Oracle® Warehouse Builder

User's Guide

10*g* Release 2 (10.2.0.2)

B28223-05

April 2009

ORACLE®

Oracle Warehouse Builder User's Guide 10*g* Release 2 (10.2.0.2)

B28223-05

Contents

4 Designing Target Schemas

7 Designing Process Flows

8 Understanding Performance and Advanced ETL Concepts

9 Using Oracle Warehouse Builder Transformations

10 Understanding Data Quality Management

13 Defining Dimensional Objects

14 Defining Flat Files and External Tables

15 Defining Business Intelligence Objects

16 Importing Data Definitions

17 Integrating Metadata Through the Warehouse Builder Transfer Wizard

18 Importing Data From Third Party Applications

21 Data Quality Operators

Part IV ETL Design Reference

22 Using Activities in Process Flows

23 Moving Large Volumes of Data

24 ETL Objects Configuration

25 Source and Target Operators

26 Data Flow Operators

27 Transformations

30 Auditing Deployments and Executions

33 Importing and Exporting with the Metadata Loader (MDL)

34 Extending the Warehouse Builder Repository

Preface

This preface includes the following topics:

Audience

This manual is written for Oracle Database administrators and others who create warehouses using Oracle Warehouse Builder.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at http://www.fcc.gov/cgb/consumerfacts/trs.html, and a list of phone numbers is available at http://www.fcc.gov/cgb/dro/trsphonebk.html.

Conventions

In this manual, Windows refers to the Windows NT, Windows 2000, and Windows XP operating systems. The SQL*Plus interface to Oracle Database may be referred to as SQL.

In the examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following table lists the conventions used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
...	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted.
boldface text	Boldface type in text refers to interface buttons and links. Boldface type also serves as emphasis to set apart main ideas.
italicized text	Italicized text applies to new terms introduced for the first time. Italicized text also serves as an emphasis on key concepts.
`unicode text`	Unicode text denotes exact code, file directories and names, and literal commands.
`italicized unicode text`	Italicized unicode text refers to parameters whose value is specified by the user.
[]	Brackets enclose optional clauses from which you can choose one or none.

Getting Help

Help is readily available throughout Warehouse Builder:

- **Menus**: Menu bars throughout Warehouse Builder contain a Help menu. For context-sensitive information, choose **Topic** from the Help menu.
- **Wizards and Dialog Boxes**: Detailed instructions are provided on the pages of the wizards, which take you step-by-step through the process of creating an object. Click the **Help** button for additional information about completing a specific dialog box or a page of a wizard.
- **Tools**: You can identify the tools on a toolbar by the tooltips that appear when you rest the mouse over the icon.

  Some toolbars include a Help icon, which displays the Contents page of the Help system.

- **Lists**: For items presented in lists, a description of the selected item displays beneath the list.
- **Popup menus**: Click the arrow icon on the right side of the title bar for a window. Then choose **Help** from the popup menu for context-sensitive information.

You may also want to follow the Oracle By Example tutorials at

http://www.oracle.com/technology/products/warehouse/selfserv_edu/self_service_education.html

Related Publications

The Warehouse Builder documentation set includes these manuals:

- *Oracle Warehouse Builder User's Guide*
- *Oracle Warehouse Builder Installation and Configuration Guide*
- *Oracle Warehouse Builder Scripting Reference*

In addition to the Warehouse Builder documentation, you can reference *Oracle Database Data Warehousing Guide*.

What's New

This preface describes new features for this release in the following section:

New in Oracle Warehouse Builder 10*g* Release 2 (10.2.0.2)

Enhancements to Warehouse Builder Architecture: One Unified Repository

In previous Warehouse Builder releases, the runtime environment and the design environment resided in separate repositories: the runtime repository and the design repository. Beginning in this release, both environments share a single Warehouse Builder repository. If you prefer to separate the environments, you can achieve this by creating multiple repositories.

User Interface Enhancements

In previous releases of Warehouse Builder, editing different types of objects required using different editors. This release provides one common look and feel for all editors, including automatic layout, dockable panels, bird's eye view, and zoom capabilities. A new property inspector standardizes the properties interface for all objects.

Enhancements to Accessing Non-Oracle Data and Metadata

- **Creating Flat File Targets:** In previous releases, you used the Mapping Editor when you wanted to create a new flat file target for which there was no existing metadata to import. You added an unbound flat file operator to the mapping canvas and then used the operator editor to add and define fields for the flat file. Beginning in this release, the preferred method is to use the Create Flat File Wizard. Compared to the previous method, the Create Flat File Wizard enables a wider range of functionality such as specifying the character set and defining single or multiple record types.
- **Introducing Non ASII Flat Files:** In previous releases, importing binary files into Warehouse Builder was difficult or not possible, depending on the complexity of the file. Beginning in this release, you can use the Create Flat File Wizard to introduce non ASCII files into Warehouse Builder as described in Chapter 14, "Defining Flat Files and External Tables".
- **Sampling ASII Flat Files:** This release introduces enhancements to the Flat File Sample Wizard that include sampling and importing new data types such as GRAPHIC, RAW, and SMALLINT.
- **Custom Metadata Interface:** Beginning with this release of Warehouse Builder, you can define and use SQL- or XML-based custom metadata stores to retrieve

definitions of source and target objects such as tables and views as described in Chapter 16, "Importing Data Definitions".

- **Deploying to Remote and non-Oracle Targets:** Beginning with this release, you can deploy objects to a remote or a non-Oracle database. This means that you can configure a target operator in a mapping to refer to a remote Oracle location or a non-Oracle location.

Enhancements to Metadata Management

- **Impact Analysis and Change Management:** Warehouse Builder now provides graphical tools for analyzing the possible outcomes to proposed metadata changes. Determine the potential costs of your proposed changes as described in Chapter 31, "Managing Metadata Dependencies".
- **Defining User Defined Properties (UDPs):** In the previous release, you could assign UDPs only to existing Warehouse Builder repository objects. You defined UDPs using the OMB Plus scripting command OMBDEFINE and prefixed the new property name with *UPD_*. Beginning in this release, you can define new objects in addition to new properties. Prefix all user defined objects and properties with *UD_* . Any properties you defined in the previous release using the *UDP_* prefix are still valid; however, *UD_* is now the preferred prefix for both properties and objects. Refer to Chapter 34, "Extending the Warehouse Builder Repository" for more information.
- **New User Defined Objects:** Beginning in this release, you can define new folders, first class objects, and second class objects. For each object you can define properties and associations, manage them in the Warehouse Builder repository, and assign custom icons for unique recognition. Because user defined objects are repository objects, you can access metadata reports and lineage and impact analyses. Refer to Chapter 34, "Extending the Warehouse Builder Repository" for more information.
- **User Defined Icons:** You can import custom icons for unique representation in the user interface of any of the existing objects and user defined objects. Refer to Chapter 34, "Extending the Warehouse Builder Repository" for more information.

Enhancements to Metadata Security

Beginning in this release, Warehouse Builder offers a user interface for defining and implementing metadata security policies.

ETL Design and Performance Enhancements

- **Scheduled Process Execution:** In previous releases, you were required to use Oracle Workflow to schedule the execution of Warehouse Builder mappings and process flows. Beginning in this release and when using Oracle database release 10*g* or higher, you can create schedules to plan when and how often to execute mappings and process flows. You can define schedules to execute once or to execute repeatedly based on an interval you define in the user interface as described in Chapter 28, "Scheduling ETL Objects".
- **High Performance Data Extraction from Remote Sources:** In previous releases, if you designed mappings that extracted data from remote sources, you could expect performance to be slow as data was accessed via database links. Beginning in this release, you can use Transportable Modules to replicate remote Oracle databases to the local Oracle database and thereby achieve rapid data extraction as described in Chapter 23, "Moving Large Volumes of Data".

- **Pluggable Mappings:** This new feature in Warehouse Builder increases your design productivity through reusable logic that you can incorporate into various ETL processes or share between many designers. For details, refer to Chapter 6, "Creating Mappings".
- **Set Based Updates:** In previous versions, when you set an Oracle target operator to load using an UPDATE loading type, Warehouse Builder updated the target in row based mode. Beginning with this release, if you configure the Oracle target module to generate 10*g* PL/SQL code, Warehouse Builder performs UPDATES in set based mode. For modules configured to generate 9*i* and earlier versions of PL/SQL code, Warehouse Builder performs UPDATES on targets in row based mode.
- **User defined Data Types:** Beginning in this release, Warehouse Builder offers a user interface for creating user defined data types and using them for mapping in Warehouse Builder. You can use the user defined types to model real-world entities such as customers and purchase orders as objects in the database.

Enhancements to Enable Quality Information

- **Slowly Changing Dimensions (SCDs):** Warehouse Builder now provides support for designing, deploying, and loading relational Type 1, 2, and 3 SCDs. For information on defining SCDs, refer to Chapter 13, "Defining Dimensional Objects". For information on loading data into or sourcing data from SCDs using the Dimension operator, refer to Chapter 25, "Source and Target Operators".
- **New Sources and Targets:** You can now read data from any source or write data to any target using PL/SQL and Java APIs. You can also read data from or write data to table functions and Oracle streams.
- **Data Profiling:** This new feature in Warehouse Builder enables you to discover the structural content of your data, capture its semantics, and identify any anomalies or outliers prior to loading it in your BI system. With data profiling, you can automatically derive business rules and mappings to clean data, derive quality indices such as Six Sigma, and use auditors to continuously monitor data quality. You can integrate data profiling into your ETL process.

Enabling Business Intelligence

- **Dimensional Objects:** In the previous release, you had to use the OLAP bridge to deploy dimensional objects to an analytic workspace. Beginning with this release, the logical design of the dimensional object is separated from the storage. You can use the same metadata to create and manage both your relational and multidimensional data stores. You define the dimensional object and can then deploy them directly either to a relational schema or to an analytic workspace. For information on creating dimensional objects, refer to Chapter 13, "Defining Dimensional Objects".
- **Business Intelligence Objects:** Beginning with this release, you can define or derive business intelligence objects that can be integrated with analytical business intelligence tools such as Oracle BI Discoverer and Oracle BI Beans. You can deploy business intelligence objects defined using Warehouse Builder to these tools and then perform adhoc queries on the warehouse data. For information on using business intelligence objects, refer to Chapter 15, "Defining Business Intelligence Objects".

Enhancements to Enable Expertise Capture

Beginning with this release, you can create experts that enable you to build your own applications by reusing Warehouse Builder components. Experts are solutions that enable advanced users to design solutions that simplify routine or complex tasks that can be performed by end users. For more information using experts, see Chapter 35, "Implementing Best Practices".

Terminology Changes

- **Synchronize replaces Reconcile:** For an operator such as a source and target operator or a key lookup, Warehouse Builder maintains a version of the object in the repository. Propagating changes between an operator and its repository object is known as synchronizing. In previous, releases, this process was known as reconciling or reconciliation. These terms are no longer used.
- **Refresh replaces Synchronize:** When multiple users access the same repository, use the refresh command to update the Design Center display. In previous releases, this command was called synchronize. Synchronize now refers to the action of updating an operator with an associated repository object.
- **User Defined Process replaces External Process:** In previous releases, an activity that you defined in a process flow to launch an external process was known as an External Process activity. In this release, the term is now User Defined Process.

Improvements to the Documentation Set

The documentation set has been reorganized and revised.

The book formerly entitled the *Oracle Warehouse Builder Installation and Configuration Guide* is now entitled the *Oracle Warehouse Builder Installation and Administration Guide* and includes administration information such as implementing security.

Oracle Warehouse Builder User's Guide now includes enhanced introductory and conceptual information.

Oracle Warehouse Builder API and Scripting Reference now includes information on using experts and the Expert Editor, which was formerly contained in the User's Guide.

Part I

Introduction and Concepts

This part contains the following chapters:

1

Overview

Oracle Warehouse Builder provides enterprise solutions for end-to-end data management. This chapter introduces you to the range of functionality provided by Warehouse Builder.

This chapter includes the following topics:

About Oracle Warehouse Builder

Oracle Warehouse Builder is a single, comprehensive tool for all aspects of data management. Warehouse Builder leverages the Oracle Database to transform data into high-quality information. It provides data quality, data auditing, fully integrated relational and dimensional modeling, and full life cycle management of data and metadata. Warehouse Builder enables you to create data warehouses, migrate data from legacy systems, consolidate data from disparate data sources, clean and transform data to provide quality information, and manage corporate metadata.

Data Consolidation and Integration

Many global corporations have data dispersed on different platforms using a wide variety of data reporting and analysis tools. Customer and supplier data may be stored in applications, databases, spreadsheets, flat files, and legacy systems. This diversity may be caused by organizational units working independently over the course of time, or it may be the result of business mergers. Whatever the cause for diversity, this diversity typically results in poor quality data that provides an incomplete and inconsistent view of the business.

Transforming poor quality data into high quality information requires:

- **Access to a wide variety of data sources**

  Warehouse Builder leverages the Oracle Database to establish transparent connections to numerous third-party databases, applications, files, and data stores as listed in "Supported Sources and Targets" on page 5-2.

- **Ability to profile, transform and cleanse data**

  Warehouse Builder provides an extensive library of data transformations for data types such as text, numerical, date, and others. Use these transformations to reconcile the data from many different sources as described in "Using Oracle Warehouse Builder Transformations" on page 9-1.

Before loading data into a new data store, you can optionally profile the data to evaluate its quality and appropriateness. Subsequently, you can match and merge records using rules that you devise. You can validate name and address data against postal databases. This process of changing poor quality data into high quality information is introduced in "About the Data Quality Management Process" on page 10-1.

- **Ability to implement designs for diverse applications**

  Using Warehouse Builder, you can design and implement any data store required by your applications, whether relational or dimensional.

- **Audit trails**

  After consolidating data from a variety of sources into a single data store, you likely face the challenge of proving the validity of the output information. For instance, can you track and proof how a particular number was derived? This is a question often posed by decision makers within your organization and by government regulators.

Depending on how you utilize Warehouse Builder, you may require licenses for additional database options and, or technologies such as Oracle Partitioning, Oracle OLAP and the Oracle Transparent Gateways.

See Also: *Oracle Database Licensing Guide*

2

Creating an Oracle Data Warehouse

Oracle Warehouse Builder is a flexible tool that enables you to design and deploy various types of data management strategies, including traditional data warehouses.This chapter provides a brief introduction to the basic, minimal steps for creating an Oracle data warehouse. It provides a starting point for using Warehouse Builder for the first time and serves as a road map to the documentation.

This chapter includes the following topics:

Understanding the Basic Concepts

Oracle Warehouse Builder is comprised of a set of graphical user interfaces to assist you in implementing complex data system designs. Your designs are saved as metadata in a centralized repository.

The centralized repository, known as the **Warehouse Builder repository,** is hosted on an Oracle Database. The **Design Center** is the interface that provides a visual representation of the Warehouse Builder repository. Use the Design Center to import source objects such as tables and views, design ETL processes such as mappings, and ultimately design the target warehouse.

A **mapping** is an object in which you define the flow of data from sources to targets. Based on a mapping design, Warehouse Builder generates the code required to implement the ETL logic.Warehouse Builder can generate PL/SQL, SQL*Loader, or ABAP code for mappings.

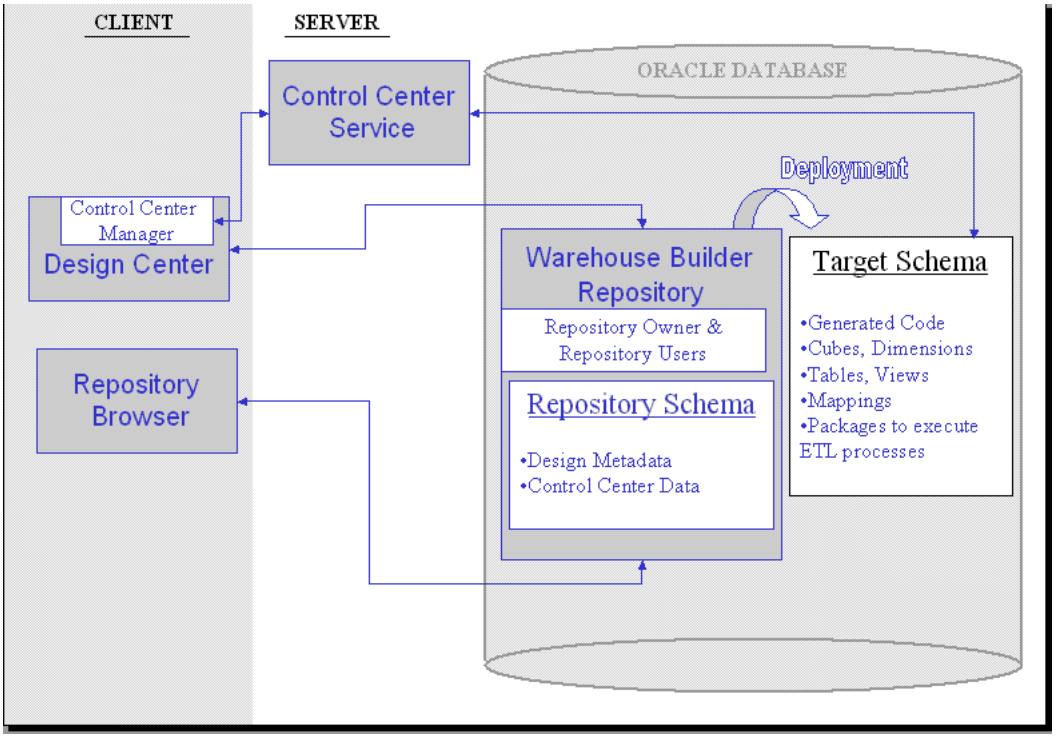
After you complete the design of a mapping, for example, and prompt Warehouse Builder to generate the code, the next step is to deploy the mapping. **Deployment** is the process of copying the relevant metadata and code you generated in the Design Center to a target schema. The **target schema** is generically defined as the database which will execute the ETL logic you designed in the Design Center. Specifically, in a traditional data warehousing implementation, the data warehouse is the target schema and the two terms are interchangeable.

Figure 2–1 illustrates the deployment process. For the purposes of this illustration, the target schema and the repository exist on the same Oracle Database; however, in practice, target schemas often exist on separate databases. To deploy design objects and subsequently execute the generated code, use the **Control Center Manager,** which

is the client interface that interacts with the target schema through the control center service.

As previously noted, the Design Center is the primary user interface. It is also a centralized interface in that you can launch from it all the client based tools, including the Control Center Manager. A secondary user interface is the web-based **Repository Browser.** In addition to browsing design metadata and auditing execution data, you can view and create reports. You can also perform limited lineage impact analysis in the Repository Browser. However, the Design Center has more advanced, interactive capabilities for lineage and impact analysis.

Figure 2–1 Warehouse Builder Components



General Steps for Creating an Oracle Data Warehouse

Use Warehouse Builder to create a data warehouse in the following recommended stages:

- Before You Begin
- Stage 1: Preparing the Warehouse Builder Design Center
- Stage 2: Importing the Source Metadata
- Stage 3: Designing the Oracle Data Warehouse
- Stage 4: Deploying the Design and Implementing the Data Warehouse

Before You Begin

Before you can use any of the Warehouse Builder client components, first ensure access to a Warehouse Builder repository, which is an Oracle Database schema that stores the metadata for the system you design.

To begin using Warehouse Builder, take the following steps:

1. Install the Warehouse Builder software and create a repository as described in the Oracle Warehouse Builder Installation and Administration Guide.

   If an administrator has previously completed the server and client installation, contact that person for the connection information required to log on to the repository.

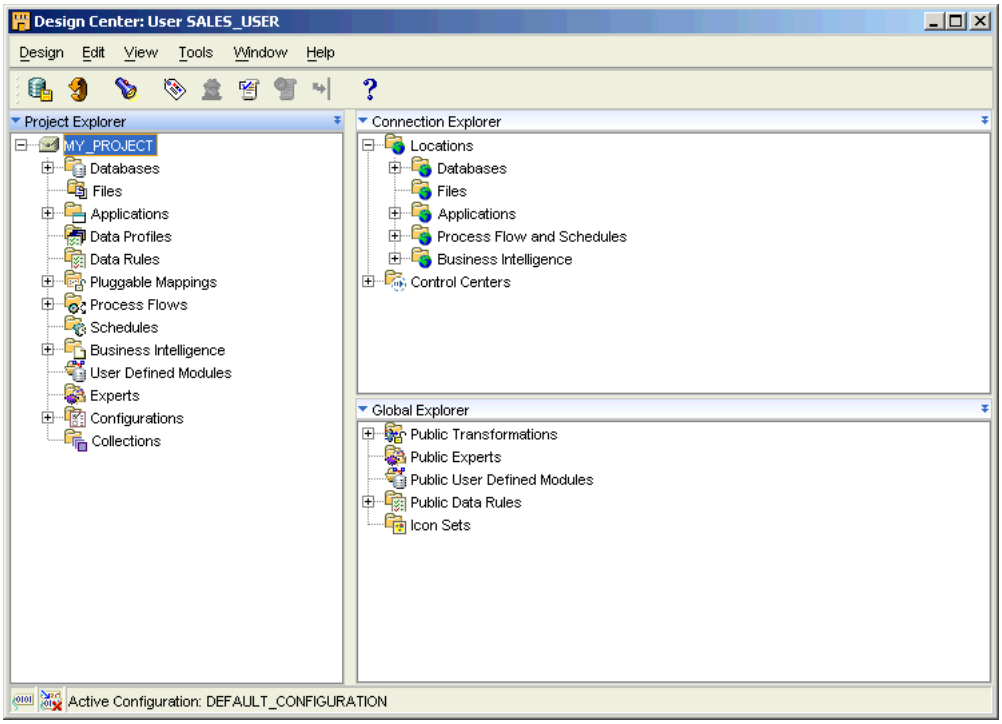
2. Launch the Design Center, shown in Figure 2–2 on page 2-3.

   On a Windows platform, from the **Start** menu, select **Programs.** Select the Oracle home in which Warehouse Builder is installed, then **Warehouse Builder,** and then **Design Center.**

   On a Linux platform, launch `owbclient.sh` located in the owb/bin/unix directory in the Oracle home for Warehouse Builder.

Figure 2–2 shows the Design Center with the top level folders in each of its 3 explorers expanded.

Figure 2–2 The Design Center



Stage 1: Preparing the Warehouse Builder Design Center

To integrate data and design a data warehouse, you primarily utilize the Project Explorer and the Connection Explorer shown in Figure 2–2.

In the Project Explorer, Warehouse Builder creates a single default project, MY_PROJECT. As a project, it contains nodes for each type of design object that you can create or import.

The Connection Explorer is the window you use to establish connections between the Warehouse Builder repository to databases, data files, and applications.

To prepare the Design Center, complete the following steps:

1. Adjust the client preference settings as desired or accept the default preference settings and proceed to the next step.

From the main menu in the Design Center, select **Tools** and then **Preferences.**

As a new user, you may be interested in setting the Environment Preferences, the locale under Appearance Preferences, and the naming mode under Naming Preferences. For information on all the preferences, see Setting Preferences on page 3-4.

2. Connect to the source and target data objects.

   You establish these connections by defining *locations* in the Connection Explorer. Expand the **Location** node and the nodes within it to gain a general understanding of the types of source and targets you can access from Warehouse Builder.

   To create a location, right-click the appropriate node and select **New.** Fill in the requested connection information and select **Test Connection.** In this step, you merely establish connections to sources and targets. You do not move data or metadata until subsequent steps.

   For more information about locations, see "About Locations" on page 2-7.

3. Organize the design environment.

   If you are satisfied with the default project, MY_PROJECT, continue with the next step.

   Alternatively, you can define more projects at any time. Each project you define is organized in the same fashion as shown in MY_PROJECT with nodes for databases, files, applications, and so on. For a different organization, consider creating optional collections as described in "Organizing Design Objects into Projects and Collections" on page 3-1.

4. Identify the Oracle Database target.

   Although you can use a flat file as a target, the most common and recommended scenario is to use an Oracle schema as the data warehouse target as described in these steps.

   To define the Oracle target, begin by creating a *module.* Modules are grouping mechanisms in the Project Explorer that correspond to locations in the Connection Explorer. The Oracle target module is the first of several modules you create in Warehouse Builder.

   In the Project Explorer, expand the **Databases** node. Right-click **Oracle** and select **New.** The Create Module wizard displays. Set the module type to **target** and specify whether the module will be used in development, quality assurance, or production. This module status is purely descriptive and has no bearing on subsequent steps you take.

   When you complete the wizard, the target module displays with nodes for mappings, transformations, tables, cubes and the various other types of objects you utilize to design the target warehouse.

5. Create a separate Oracle module for the data sources. (Optional)

   At your discretion, you can either create another Oracle module to contain Oracle source data or simply proceed to the next step.

6. Identify the execution environment.

   Under the Connection Explorer, notice the Control Centers node. A control center is an Oracle Database schema that manages the execution of the ETL jobs you design in the Design Center in subsequent steps.

During installation, Warehouse Builder creates the `DEFAULT_CONTROL_CENTER` schema on the same database as the repository.

If you choose to utilize the default execution environment, continue to the next step. Alternatively, you can define new control centers at any time. For more information and instructions, see "Deploying to Target Schemas and Executing ETL Logic" on page 11-1.

7. Prepare development, test, and production environments. (Optional)

   Thus far, these instructions describe the creation of a single project corresponding to a single execution environment. You can, however, reuse the logical design of this project in different physical environments such as testing or production environments.

   Deploy a single data system to several different host systems or to various environments, by "Creating Additional Configurations" on page 3-11.

Stage 2: Importing the Source Metadata

1. Import metadata from the various data sources.

   In the Project Explorer, select a node and determine the locations from which you intend to ultimately extract data. Now create a module for each relevant location. After you create a module, right-click the module and select **Import** to extract metadata from the associated location. Warehouse Builder launches a wizard to guide you through the process of importing data.

   For an example and additional information on importing data objects, see "Identifying Data Sources and Importing Metadata" on page 5-1.

2. For the metadata you imported, profile its corresponding data. (Optional)

   Before continuing to the next step, consider using the data profiling option to ensure data quality as described in "Understanding Data Quality Management" on page 10-1.

Stage 3: Designing the Oracle Data Warehouse

1. Create and design the data objects for the Oracle target module.

   In previous steps, you may have already imported existing target objects. For new target objects, design any of the dimensional or relational objects listed in Table 4–1 on page 4-2.

   To create data objects, you can either launch the appropriate wizard or use the Data Object Editor. To use a wizard, right-click the node for the desired object and select **New.** After using a wizard, you may want to modify the object in the editor. In that case, right-click the object and select **Open Editor.**

   For additional information, see Chapter 4, "Designing Target Schemas".

2. As you design your data warehouse, be sure to frequently validate the design objects.

   You can validate objects as you create them, or validate a group of objects together. In the Project Explorer, select one or more objects or modules, then click the Validate icon.

   Examine the messages in the Validation Results window. Correct any errors and try validating again.

   To redisplay the most recent validation results at a later time, choose **Validation Messages** from the View menu.

3. When satisfied with the design of the target objects, generate the code.

   Generation produces a DDL or PL/SQL script to be used in subsequent steps to create the data objects in the target schema.

   In the Data Editor, you can generate code for a single object by clicking the Generate icon.

   In the Project Explorer, select one or more objects or modules, then click the Generate icon. Examine the messages in the Generation Results window. To redisplay the most recent generation results at a later time, choose **Generated Scripts** from the View menu.

   You can save the generated script as a file and optionally deploy it outside Warehouse Builder.

4. Design mappings that define the flow of data from a source to target objects.

   In the Project Explorer, expand the Oracle target module, right-click the Mappings node and select **New.**

   The Mapping Editor enables you to define the flow of data visually. You can drag-and-drop operators onto the canvas, and draw lines that connect the operators.

   Follow the Instructions for Defining Mappings, concluding with generating the code for the mapping.

5. To manage dependencies between mappings, refer to "Designing Process Flows" on page 7-1.

Stage 4: Deploying the Design and Implementing the Data Warehouse

Recall that deployment is the process of copying the relevant metadata and code you generated in the Design Center to a target schema. This step is necessary to enable the target schema to execute ETL logic such as mappings.

To deploy and execute the data warehouse design, complete the following steps:

1. Deploy objects from either the Design Center or Control Center Manager.

   In this step, you define the objects in the target schema. You need do this only once.

   The simplest approach is to deploy directly from the Design Center by selecting an object and clicking the Deploy icon. In this case, Warehouse Builder deploys the objects with the default deployment settings.

   Alternatively, if you want more control and feedback on how Warehouse Builder deploys objects, select Tools from the Design Center menu and select Control Center Manager.

   Whether you deploy objects from the Design Center or the Control Center Manager, be sure to deploy all associated objects. For example, when deploying a mapping, also deploy the target data objects such as tables that you defined and any associated process flows or other mappings.

   For more information, see "Deploying to Target Schemas and Executing ETL Logic".

2. Execute the ETL logic to populate the target warehouse.

   In this step, you move data for the first time. Repeat this step each time you want to refresh the target with new data.

You have two options for executing the ETL logic in mappings and process flows. You can create and deploy a schedule as described in "Process for Defining and Using Schedules" on page 11-4. Or you can execute jobs manually as described in "Starting ETL Jobs" on page 11-5.

About Locations

Locations enable you to store the connection information to the various files, databases, and applications that Warehouse Builder accesses for extracting and loading data. Similarly, locations also store connection information to ETL management tools and Business Intelligence tools. For a detailed listing, see "Supported Sources and Targets" on page 5-2.

Oracle Database locations and file locations can be sources, targets, or both. For example, you can use a location as a target for storing temporary or staging tables. Later, you can reuse that location as a source to populate the final target schema.

In some cases, such as with flat file data, the data and metadata for a given source are stored separately. In such a case, create a location for the data and another for the metadata.

About Locations, Passwords, and Security

Considering that all Warehouse Builder users can view connection information in a location, note that the passwords are always encrypted. Furthermore, Warehouse Builder administrators can determine whether or not to allow locations to be shared across users and persisted across design sessions. By default, locations are not shared or persisted.

To implement either of these scenarios, see the topic "Managing Passwords in Warehouse Builder" in the *Oracle Warehouse Builder Installation and Administration Guide*.

Automatically Created Locations

Warehouse Builder automatically creates an Oracle location for each user you register in the repository. For example, if you register user `SCOTT`, then the repository includes an Oracle location named `SCOTT_LOCATION` that stores the connection details for to the `SCOTT` schema.

During installation, Warehouse Builder creates an Oracle location named `OWB_REPOSITORY_LOCATION`. This location provides the connection details to the Warehouse Builder repository. You cannot rename or delete the repository location. Only a database administrator can change the password. To prevent unauthorized access to the database administrator password, all users are restricted from deploying to the repository location.

Deleting Locations

To delete a location, right-click the location in the Connection Explorer and select **Delete.** If the delete option is not available here, this indicates that the location has been registered in a control center and is likely being utilized. Verify that the location is not in use, unregister the location in the control center, and then you can delete the location from the Connection Explorer.

About Modules

Modules are grouping mechanisms in the Project Explorer that correspond to locations in the Connection Explorer. A single location can correspond to one or more modules. However, a given module can correspond to only a single location at a time.

The association of a module to a location enables you to perform certain actions more easily in Warehouse Builder. For example, you can re-import metadata by reusing an existing module. Furthermore, when you deploy ETL processes in subsequent steps, modules enable you to deploy related objects together such as process flows.

To create a module:

1. Expand the Project Explorer until you find the node for the appropriate metadata type.

   For example, if the source data is stored in an Oracle Database, then expand the **Databases** node to view the **Oracle** node. Or if the source data is in an SAP R/3 system, expand the applications node to view the **SAP** node.

2. Right-click the desired node and choose **New**.

   The Create Module wizard opens.The wizard determines the correct integrator to use to enable access to the data store you selected.

3. Follow the prompts in the wizard.

   The wizard prompts you to name and describe the module and associate it with a location.

During the course of using Warehouse Builder, you may need to associate a module with a new location. For example, assuming your production environment utilizes different locations than your development environment, you need to reassociate the modules.

To change the location associated with a module:

1. In the Project Explorer, select the module.
2. Click the Configure icon.

   The Configuration Properties dialog box appears.

3. In the Identification folder, select a new value for the Locations property.

3

Setting Up Warehouse Builder

This chapter includes additional and optional steps that you may take when initially designing your data system. This chapter covers the following topics:

- Organizing Design Objects into Projects and Collections
- Defining Collections
- Setting Preferences
- About Connectors

Organizing Design Objects into Projects and Collections

Projects are the largest storage objects within the Warehouse Builder design repository. Projects store and organize related metadata definitions. You should include all the objects in a project that you think can or will share information. These definitions include data objects, mappings, and transformation operations. The definitions are organized into folders within the project. By creating multiple projects, you can organize the design and deployment of a large system.

To create a project:

1. In the Project Explorer, right-click a project such as `MY_PROJECT` and select **New**.

   The Create Project dialog box is displayed.

2. Click **Help** for additional instructions.

Each Warehouse Builder repository has a default project called `MY_PROJECT`. You can rename `MY_PROJECT`, or you can delete it after you create other projects. However, a repository must contain at least one project at all times.

Because projects are the main design component in Warehouse Builder, some restrictions are enforced to prevent you from deleting them unintentionally. You cannot delete:

- The currently active or expanded project.
- The only project in a repository.

To delete a project:

1. In the **Project Explorer**, collapse the project that you want to delete. You cannot delete the project when it is expanded.
2. Select and expand any other project.
3. Highlight the project you want to delete and, from the **Edit** menu, select **Delete.**

   or

Right-click the project and select **Delete.**

The Deletion Confirmation dialog box provides the option of putting the project in the recycle bin.

Defining Collections

Collections are structures in Warehouse Builder that store the metadata you want to export to other tools and systems. Collections enable you to perform the following tasks:

- Organize a large logical warehouse.
- Validate and generate a group of objects.

When you create a collection, you do not create new objects or copies of existing objects. You create shortcuts pointing to objects already existing in the project. Use a shortcut to quickly access a base object and make changes to it.

You can define more than one collection within a project and an object can be referenced by more than one collection. For example, each user that accesses a project can create his own collection of frequently used objects. The users can also add the same objects (such as mappings, tables, or process flows) to their separate collections.

Each user can also delete either the shortcut or the base object. Shortcuts to deleted objects are deleted in the collection.

When you open an object in a collection, you obtain a lock on that object. Warehouse Builder prevents other users from editing the same object from another collection.

Creating a Collection

Use the New Collection Wizard to define a collection.

To define a new collection:

1. Select and expand a project node on the Project Explorer.
2. Right-click the **Collections** node and select **New.**

   Warehouse Builder displays the Welcome page for the Create Collections Wizard. This page lists the steps to create a collection. Click **Next** to proceed.
3. Provide information on the following pages of the Create Collection wizard:
   - Name and Description Page on page 3-2
   - Contents Page on page 3-2
   - Summary Page on page 3-3

Name and Description Page

Use the Name and Description page to provide a name and an optional description for the collection. The name should be unique within the module. In physical naming mode, type a name between 1 to 200 characters. Spaces are not allowed. In logical mode, the maximum number of characters is 200 and spaces are allowed.

Contents Page

The Contents page enables you to select the data objects that you want to refer to in the collection. Use the following steps:

1. Select and expand the project node in the left panel.

The wizard displays a list of objects you can add to the collection.

2. Select objects from **Available:** section in the left panel.

   Use the **Ctrl** key to select multiple objects. You can select objects at the object level or the module level. For example, under the Files node, you can add a specific file or add all the files in a given flat file module.

   If you add a module or another collection, Warehouse Builder creates references to the module or collection and also creates references to objects contained in the module or collection.

3. Click the left to right arrow.

   The wizard displays the list of objects under **Selected:** section on the right panel. You can remove objects from the list by selecting objects and clicking the right to left arrow.

Summary Page

The Summary page displays the objects selected for the collection. Review the objects and click **Back** to make changes to your selections. Click **Finish** to complete the collection definition. Warehouse Builder creates the collection and adds it to the Project Explorer.

Editing Collection Definitions

Use the Edit Collection dialog to edit a collection. You can perform the following actions when you edit a collection definition:

- Rename the collection
- Add data objects to the collection
- Remove data objects that the collection references.

From the Project Explorer, right-click the collection and select Open Editor. Warehouse Builder displays the Edit Collection dialog that contains the following two tabs: **Name Tab** and **Contents Tab**. Use these tabs to edit the collection definition.

Name Tab

Use the Name tab to rename a collection or modify its description. To rename a collection, select the name in the **Name** field and enter the new name. The name must be unique within the project. In physical naming mode, type a name between 1 to 200 characters. Spaces are not allowed. In logical mode, the maximum number of characters is 200 and spaces are allowed.

You can also modify the description of the collection using the **Description** field.

Contents Tab

Use the Contents page to select objects that you want to reference in the collection.

To complete the Contents page:

1. Select and expand the project node in the left panel.

   The wizard displays a list of objects you can add to the collection.

2. Select objects from **Available Objects** in the left panel.

Use the **Ctrl** key to select multiple objects. You can select objects at the object level or the module level. For example, under the Files node, you can select to add a specific flat file or add all the flat files in a given flat file module.

3. Click the left to right arrow.

   The wizard displays the list of objects under **Selected Objects** on the right panel. You can remove objects from the list by selecting objects and clicking the right to left arrow.

Setting Preferences

You use the Preferences dialog to set user preferences. The Preferences dialog contains two sections. The section on the left of this dialog lists the categories of preferences. Click a category to display the preferences it contains and its value in the section on the right.

Warehouse Builder enables you to set the following types of preferences:

- Appearance Preferences
- Control Center Monitor Preferences
- Data Profiling Preferences
- Deployment Preferences
- Environment Preferences
- Generation/Validation Preferences
- Logging Preferences
- Naming Preferences
- Security Preferences

Appearance Preferences

The Appearance category contains the Locale preference. Use **Locale** to set the language you want the client text to display. The drop-down list for the Locale preferences displays the language options. Warehouse Builder prompts you to restart the computer in order to use the new language setting.

The Locale selection does not define the character set of your repository; it only affects the text and menu options on the client user interface. The repository character set is determined by the database.

Control Center Monitor Preferences

Use the Control Center Monitor category to set preferences that control the display of components in the control center. When you use the control center to deploy or execute objects, the Job Details dialog displays the results of deployment or execution. The Control Center Monitor preferences enable you to control the display of components in the object tree of the Job Details dialog.

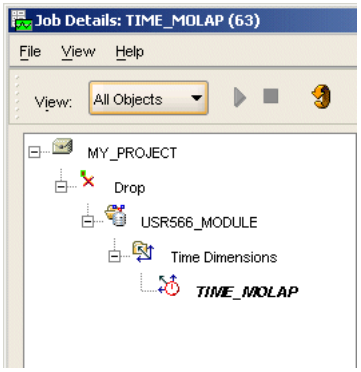
Note: Warehouse Builder displays the Job Details dialog only if you select the **Show Monitor** preference under the Process node of the Deployment category. If this option is not selected, you can display the Job Details dialog by double-clicking the row that represents the job whose details you want to display in the Control Center Jobs panel of the Control Center.

You can set the following control center monitor preferences:

- **Show Project:** Select this option to display the project name in the Job Details dialog object tree. When this option is selected, the object tree displays a node for the project name. All the objects are displayed under the project node.
- **Show Module:** Select this option to display the name of the module to which the object being deployed or executed belongs in the Job Details panel. When this option is selected, the object tree displays a node for the module. Expand the module node to view the object details.
- **Show Location:** Select this option to display the location name in the object tree of the Job Details dialog.
- **Show Action:** Select this option to display the action performed on the object in the object tree of the Job Details dialog. The actions performed include Create, Drop, Deploy, and Upgrade.
- **Show Type:** Select this option to display the type of object in the object tree of the Job Details dialog. When you select this option, a node is displayed for the type of object and the objects are listed under the respective nodes.

Figure 3–1 displays the object tree of the Job Details dialog in which the following Control Center preferences were selected: Show Project, Show Module, Show Action, and Show Type.

Figure 3–1 Job Details Dialog with Control Center Monitor Preferences



Data Profiling Preferences

Use the Data Profiling category to set the preferences for data profiling. This section contains the following preferences:

- **Data Rule Folder Name:** Use this option to set the name of the folder that contains the data rules as a result of data profiling.
- **Default Profile Location:** Use this option to set the default location that is used to store the data profiling results. You can override this setting by selecting a

different location as your profile location. In the Data Profile Editor, select the Properties option from the Edit menu. Use the Data Locations tab to change the default profile location.

Deployment Preferences

The Deployment category enables you to set deployment preferences such as displaying the deployment monitor, prompting for execution parameters, and showing completion messages. This enables you to control some of the popup windows that are displayed by the Control Center Manager during object deployment.

Deployment preferences are divided into two sections: Process and Tools. Expand the **Deployment** node in the Preferences dialog. Two nodes called **Process** and **Tools** are displayed. Click the node for which you want to set preferences.

Process

Set the following deployment options in this section:

- **Allow Undo/Redo:** Select this option to allow the user to undo and redo a deployment upgrade job. You can undo or redo a deployment upgrade job using the Job Details dialog. To display the Job Details dialog for a job, double-click the job in the Control Center Jobs panel of the Control Center Manager.
- **Pause After Compile:** Select this option to pause deployment after script generation. This means that you will need to explicitly deploy an object after it is successfully generated.
- **Prompt for Commit:** Select this option to prompt the user to commit design time changes before a deployment. When you deploy objects from the Design Center, if there are any unsaved design changes, Warehouse Builder prompts you to save these changes by displaying the Warehouse Builder Warning dialog. Click **Save** to commit unsaved design changes. Click Cancel to terminate the deployment.

  If you do not set this option, Warehouse Builder saves any design changes before the deployment job.
- **Prompt for Job Name:** Select this option to prompt the user for the name of a deployment job. When this option is not selected, Warehouse Builder assigns a default job name.
- **Prompt for Execution Parameters:** Select this option to prompt the user for the values of execution parameters. If you do not select this option, Warehouse Builder uses the default value of parameters during the execution. The user is not prompted to provide the parameter values.
- **Show Monitor:** Select this option to display the Job Details dialog when you deploy or execute an object. This dialog displays details of the objects being deployed, deployment progress, and deployment status.
- **Show Deployment Completion Message:** Select this option to display an alert indicating that the deployment job has completed.
- **Show Design Center Deployment Job:** Select this option to display the Control Center Jobs dialog when you deploy an object from the Design Center. The control Center Jobs dialog, which is similar to the Jobs panel of the Control Center Manager, contains the Deployment, Execution, and Scheduled tabs. Use this option to view the status of a deployment job while deploying using the Design Center.

Tools

Set the following deployment options:

- **Show Monitor Tree:** Select this option to show the job Details dialog when you perform a deployment or execution.
- **Show Monitor Results:** Select this option to display the deployment or execution results in Control Center Manager.
- **Show Monitor Logfile:** Select this option to display the log file in the Control Center Manager.

Environment Preferences

The Environment category enables you to set generic preferences that control the client environment such as displaying welcome pages for wizards and recycle bin preferences.

Set the following environment preferences:

- **Personality:** For the standard installation, set the value of this preference to Default. For a customized installation, this preference tailors the types of objects shown in the Project Explorer tree. Oracle recommends that you change the value of this preference, from Default, only after discussion with your Oracle system administrator. This feature is reserved for future use.
- **Allow Optimize Repository Warning on Startup:** Select this option to collect schema statistics when you log in to Warehouse Builder. Collecting schema statistics improves repository performance. If this option is selected, at log on, Warehouse Builder determines if statistics need to be gathered for the repository schema. If statistics need to be gathered, a warning dialog is displayed asking if you want to gather statistics now. Click **Yes** to collect schema statistics and optimize the repository.

  If you do not select this option, you can still collect schema statistics from the Design Center. Select the **Optimize Repository** option from the **Tools** menu.
- **Hide All Wizard Welcome pages:** Select this option to hide the welcome page of all wizards. Every wizard in Warehouse Builder starts with a Welcome page that summarizes the steps to follow to complete that task. To display the Welcome page for all wizards, deselect this preference.
- **Show Delete Confirmation Dialog:** Select this option to display a dialog that asks for a confirmation before deleting an object. When this option is selected, if you delete an object, the Warehouse Builder Warning dialog is displayed. Click **Yes** to delete the object. Click **No** to cancel the Delete operation and retain the object.
- **Recycle Deleted Objects:** Select this option to move deleted objects to the recycle bin. If this option is not selected, any objects you delete are lost and you have no way of recovering them.
- **Empty Recycle Bin on Exit:** Select this option to empty the contents of the recycle bin when you exit the Warehouse Builder client. Deselect this option to save the recycle bin objects across sessions.

Generation/Validation Preferences

The Generation/Validation category enables you to set preferences related to generation and validation of Warehouse Builder design objects. Use these preferences to control what is displayed in the Generation Results dialog or Validation Results

dialog. These dialogs are displayed when you generate or validate an object from the design center. You can set the following preferences:

- **Show Project:** Select this option to display the project node in Validation Results dialog or the Generation Results dialog.
- **Show Module:** Select this option to display the module node in Validation Results dialog or the Generation Results dialog.
- **Show Location:** Select this option to display the location node in Validation Results dialog or the Generation Results dialog.
- **Show Action:** Select this option to display the action node in Validation Results dialog or the Generation Results dialog.
- **Show Type:** Select this option to display the type node in Validation Results dialog or the Generation Results dialog.

Logging Preferences

The Logging Preferences category enables you to set log file options such as file location, file size, and types of messages saved to any log file. The log file contains messages relating to your design process. By default a message log is saved to the default location. Following are the logging preferences that you can set:

- **File Path:** Represents the location where the log files are saved. Type the complete path or use the Browse button to select the location. The default location is <OWB_ORACLE_HOME>\owb\bin\admin.
- **File Name:** Represents the name of the log file. Do not include a file extension when you specify a file name.
- **Maximum Size (Kb):** Indicate the maximum file size for the log file(s) in KB. There are two log files: <logfilename>.0, and <logfilename>.1. When the maximum size of the first log file <logfilename>.0 is reached, Warehouse Builder starts writing to the second, <logfilename>.1. When the maximum size of the second one is reached, Warehouse Builder starts overwriting the first.
- **Log Error Messages:** Select this option to write all error messages to the log file.
- **Log Warning Messages:** Select this option to write all warning messages to the log file.
- **Log Information Messages:** Select this option to write all information messages to the log file.

Naming Preferences

The Naming Preferences category enables you to set naming preferences by selecting whether you want to view objects in business or physical name mode. You can also set up how you want to propagate object name changes.

Set the following naming preferences:

- **Naming Mode:** Select whether to display objects using their physical or business names. For more information about naming modes, refer to "About Naming Modes" on page 3-9.
- **Propagate Name Changes:** Propagate Name Changes from the current naming mode to the other naming mode.

About Naming Modes

Warehouse Builder maintains a business and a physical name for each object stored in the repository. A business name is a descriptive logical name for an object.

When you generate DDL scripts for a named object, the physical names are used. Physical names must conform to the syntax rules for basic elements as defined in the *Oracle Database SQL Reference*. Names must be unique within their category:

- Module names must be unique within a project.
- Warehouse object names must be unique within a warehouse module. This includes the names of tables, dimensions, cubes, mappings, materialized views, sequences, views and indexes.
- Transformation names must be unique within a transformation package.

Business Name Mode You can create a business name for an object or change the business name of an existing object when Warehouse Builder is in business name mode. Warehouse Builder editors, wizards, and property sheets display the business names of objects in this mode.

A business name must conform to these rules:

- The length of a name cannot exceed 200 characters.
- The name must be unique within its category.
- All source modules reflect the case of the imported source and are subject to the double-quotes rules as defined in the *Oracle Database SQL Reference*.

Copy operations from a source to a target in a mapping do not propagate case.

When you create a business name, Warehouse Builder generates a valid physical name that resembles the business name. If you create a business name that duplicates an existing physical name, Warehouse Builder appends an underscore and a number in order to create a unique name.

Physical Name Mode You can create a physical name for an object or change the physical name of an existing object when Warehouse Builder is in the Physical name mode. Warehouse Builder editors, wizards, and property sheets display physical names of objects in this mode. Physical names are converted to uppercase.

A physical name must:

- Contain no more than 30 characters.
- Conform with the basic syntax rules for schema objects defined by the *Oracle Database SQL Reference*

> **Note:** A collection can have a physical name containing up to 200 characters.

Warehouse Builder prevents you from entering an invalid physical name. For example, you cannot enter a duplicate name, a name with too many characters, or a name that is a reserved word.

Setting the Name Mode To create or change a business name for an object, Warehouse Builder must be in business name mode. To create or change a physical name for an object, Warehouse Builder must be in physical name mode.

The default naming preferences for Warehouse Builder are as follows:

Mode: The default setting for the mode is physical name mode.

Propagation: The default propagation setting is to propagate physical name to business name.

Icons for the name mode and name propagation settings are located in the lower-left corner of the editors. These icons indicate the current naming preference setting.

Warehouse Builder saves your naming preferences across sessions. The name mode preference is stored in a file on the client workstation. If you use Warehouse Builder from another workstation, your preferences may be different.

Security Preferences

Only administrators can edit the security preferences. Administrators can set the following preferences:

Persist Location Password in Metadata

This option determines whether or not location passwords are persisted across Warehouse Builder design sessions.

By default, this option is deselected, which is the more secure option. Warehouse Builder retains location passwords for the length of the design session only. That is, the first time you launch tools such as the Data Viewer or Debugger, you must enter the appropriate location passwords.

If selected, Warehouse Builder persists encrypted versions of location passwords in repository. You can launch tools such as the Data Viewer and Debugger without entering passwords each time.

For additional information about the encryption methodology, see "Managing Passwords in Warehouse Builder" in the Oracle Warehouse Builder Installation and Administration Guide.

Share Location Password During Runtime

This preference determines whether or not the location passwords users enter during the design phase can be shared with other users. For example, assume that user `Dev1` designs mapping MAP1. To access the sources and targets for this mapping, `Dev1` defines the locations to each source and target including a username and password. When other users subsequently attempt to execute MAP1 or view data associated with it, the Share Location Password During Runtime preference determines whether or not each user must enter the password each time in the Design Center or the first time in the Control Center.

Share Location Password During Runtime works in conjunction with Persist Location Password in Metadata. The most secure mode, and therefore the default behavior, is for both options to be deactivated. In this case, each user including Dev1 must enter their password once for each Design Center session and the first time they attempt to use that location in the Control Center. Depending on your security requirements, you may want each user to define their own location for a given source or target

If both Share Location Password During Runtime and Persist Location Password in Metadata are activated, then any user can access a schema given that any user previously defined the location. Therefore, user Oper2 can execute MAP1 given that Dev1 or any other user previously defined the location with valid credentials.

Default Metadata Security Policy

Specify the default security policy to be applied. Minimum security allows all users full control over objects any newly registered user creates. Maximum security,

however, restricts access to the newly registered user that created the object and Warehouse Builder administrators.

This setting is not retroactive. That is, if you change this setting in an existing Warehouse Builder implementation, the setting does not affect existing users and existing objects. You must change the security on existing objects manually as described in "Implementing Security in Warehouse Builder" and "Applying Security Properties on Specific Metadata Objects" in the Oracle Warehouse Builder Installation and Administration Guide.

Creating Additional Configurations

A configuration contains all of the physical details required to deploy a data system. One of the steps in creating an object is to set its configuration parameters, as described in "Configuring Data Objects" on page 12-31.

A configuration consists of:

- The configuration settings of all objects
- A Control Center

You can create multiple configurations for a single project. Only one configuration is active at a time. The default configuration is named `DEFAULT_CONFIGURATION`. This is the configuration that you have been working in.

By switching among different configurations, you can change the physical design without making any changes to the logical design. You can easily deploy a single data system to several different host systems or to various environments, such as development and production. You only need to create additional configurations when working with multiple environments.

For example, suppose the Tablespace parameter for a table is set to `SMALL_TBSP` in `DEFAULT_CONFIGURATION`. You create `MY_NEW_CONFIGURATION` and make it the active configuration. Then you change the Tablespace parameter for the table to `BIG_TBSP`. When you switch to back `DEFAULT_CONFIGURATION`, the Tablespace parameter is set to `SMALL_TBSP`.

Creating a New Configuration

To create a new configuration:

1. In the Project Explorer, select a project and expand the navigation tree.
2. Select **Configurations**.
3. From the Design menu, select **New**.

   The Create Configuration wizard opens.
4. Follow the steps of the wizard.

   Click **Help** for more information.

The new configuration appears in the Configurations folder.

Activating a Configuration

To activate a configuration:

1. In the Project Explorer, select a project and expand the navigation tree.
2. Expand the Configurations folder.

3. Select a configuration.
4. From the Design menu, select **Set As Active Configuration**.

Any changes that you make to the configuration parameters of objects are saved in this configuration. If you switch to the previous configuration, then these parameters will have their previous settings.

The Control Center associated with this configuration is now active and stores all new information about validation, generation, and deployment of objects in the project.

About Connectors

A connector is a logical link created by a mapping between a source location and a target location. The connector between schemas in two different Oracle Databases is implemented as a database link, and the connector between a schema and an operating system directory is implemented as a database directory.

You do not need to create connectors manually if your repository user ID has the credentials for creating these database objects. Warehouse Builder will create them automatically the first time you deploy the mapping. Otherwise, a privileged user must create the objects and grant you access to use them. You can then create the connectors manually and select the database object from a list.

> **See Also:** *Oracle Database SQL Reference* for information about the `CREATE DATABASE LINK` and `CREATE DIRECTORY` SQL commands.

To create a database connector:

1. In the Connection Explorer, expand the Locations folder and the subfolder for the *target* location.
2. Right-click **DB Connectors** and choose **New**.

   The Create Connector wizard opens.
3. Follow the steps of the wizard. Click the **Help** button for specific information.

To create a directory connector:

1. In the Connection Explorer, expand the Locations folder and the subfolder for the *target* location.
2. Right-click **Directories** and choose **New**.

   The Create Connector dialog box opens.
3. Click the **Help** button for specific information about completing this dialog box.

About Validation

Validation is the process of verifying metadata definitions and configuration parameters. These definitions must be valid before you proceed to generation and deployment of scripts. Warehouse Builder runs a series of validation tests to ensure that data object definitions are complete and that scripts can be generated and deployed. When these tests are complete, the results display. Warehouse Builder enables you to open object editors and make corrections to any invalid objects before continuing. In addition to being a standalone operation, validation also takes place implicitly when you generate or deploy objects.

To detect possible problems and deal with them as they arise, you can validate in two stages: after creating data object definitions, and after configuring objects for deployment. In this case, validating objects after configuration is more extensive than validating object definitions.

> **Tip:** Validate objects as you create and configure them to resolve problems as they arise. The same error-checking processes are run whether you are validating the design or configuration.

When you validate an object after it has been defined, the metadata definitions for the objects you have designed are checked for errors. For example, if you create a Table, Warehouse Builder requires that columns be defined. When this object is validated, Warehouse Builder verifies that all components of the Table have been defined. If these components are missing, validation messages display in the Validation Results window.

If you validate an object after it has been configured, metadata definitions are re-checked for errors and configuration parameters are checked to ensure that the object will be generated and deployed without any problems. You can then make edits to invalid objects.

4

Designing Target Schemas

Target schemas include all the necessary data objects such as tables, views, dimensions, and cubes. In a traditional data warehousing implementation, there is typically only one target schema, which is the data warehouse target. In Warehouse Builder, you design target schemas using the Data Object Editor.

This chapter includes the following topics:

Creating Oracle Data Objects

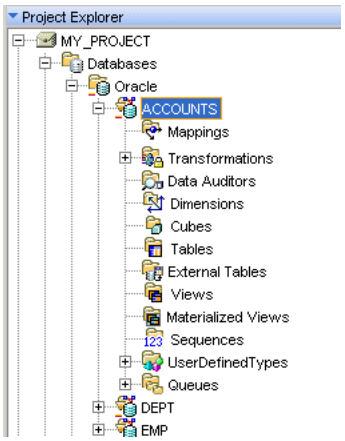
In previous steps, you may have already imported existing target objects. For new target objects, design any of the dimensional or relational objects listed in Table 4–1 on page 4-2.

To create data objects, you can either launch the appropriate wizard or use the Data Object Editor. To use a wizard, right-click the node for the desired object and select **New.** After using a wizard, you may want to modify the object in the editor. In that case, right-click the object and select **Open Editor.**

About Data Objects

The Oracle Database module contains nodes for each type of data object you can define in Warehouse Builder. In the Project Explorer under the Oracle node, expand the module node to view all the supported data objects, as shown in Figure 4–1.

Figure 4–1 Data Objects within a Warehouse Module



Warehouse Builder supports relational and dimensional data objects. Relational objects, like relational databases, rely on tables and table-derived objects to store and link all of their data. The relational objects you define are physical containers in the database that are used to store data. It is from these relational objects that you run queries after the warehouse has been created. Relational objects include tables, views, materialized views, and sequences. This chapter provides specific information about each type of relational object and how it is used in Warehouse Builder.

Dimensional objects contain additional metadata to identify and categorize your data. When you define dimensional objects, you describe the logical relationships that help store the data in a more structured format. Dimensional objects include dimensions and cubes. This chapter provides specific information about each type of dimensional object and how they are used in Warehouse Builder.

In addition to relational and dimensional objects, Warehouse Builder supports intelligence objects. Intelligence objects are not part of Oracle modules. They are displayed under the Business Intelligence node in the Explorer. Intelligence objects enable you to store definitions of business views and report structures. You can deploy these definitions to analytical tools such as Oracle Discoverer and BI Beans and perform ad hoc queries on the warehouse. For more information about intelligence objects, see Chapter 15, "Defining Business Intelligence Objects".

Table 4–1 describes the types of data objects you can use in Warehouse Builder.

Table 4–1 Data Objects in Warehouse Builder

Data Object	Type	Description
Tables	Relational	The basic unit of storage in a relational database management system. Once a table is created, valid rows of data can be inserted into it. Table information can then be queried, deleted, or updated. To enforce defined business rules on a table's data, integrity constraints can be defined for a table. See "Using Tables" on page 12-13 for more information.
External Tables	Relational	External tables are tables that represent data from non-relational flat files in a relational format. Use an external table as an alternative to using a flat file operator and SQL* Loader. See "Using External Tables" on page 14-26 for more information.

Table 4–1 (Cont.) Data Objects in Warehouse Builder

Data Object	Type	Description
Views	Relational	A view is a custom-tailored presentation of data in one or more tables. Views do not actually contain or store data; they derive their data from the tables on which they are based. Like tables, views can be queried, updated, inserted into, and deleted from, with some restrictions. All operations performed on a view affect the base tables of the view. Use views to simplify the presentation of data or to restrict access to data. See "Using Views" on page 12-17 for more information.
Materialized Views	Relational	Materialized views are pre-computed tables comprising aggregated or joined data from fact and possibly dimension tables. Also known as a summary or aggregate table. Use materialized views to improve query performance. See "Using Materialized Views" on page 12-20 for more information.
Sequences	Relational	Sequences are database objects that generate lists of unique numbers. You can use sequences to generate unique surrogate key values. See "Using Sequences" on page 12-25 for more information.
Dimensions	Dimensional	A general term for any characteristic that is used to specify the members of a data set. The 3 most common dimensions in sales-oriented data warehouses are time, geography, and product. Most dimensions have hierarchies. See "About Dimensions" on page 4-30 for more information.
Cubes	Dimensional	Cubes contain measures and links to one or more dimension tables. They are also known as Facts. See "About Cubes" on page 4-41 for more information.
Advanced Queues	Relational	Advanced Queues enable message management and communication required for application integration. Currently, you cannot create advanced queues using Warehouse Builder. You can only import advanced queues that were exported into a .mdl file using a previous version of Warehouse Builder.
Queue Tables	Relational	Queue tables are tables that store queues. Each queue table contains a payload whose data type can be an object type or RAW. You cannot create a queue table using Warehouse Builder. A queue table is imported as part of an advanced queue payload.
Object Types	Relational	An object type is made up of one or more user defined types or scalar types. See "About Object Types" on page 12-26 for more information.
Varrays	Relational	A varray is an ordered collection of elements. See "About Varrays" on page 12-28 for more information.
Nested Tables	Relational	A nested table complements the functionality of the varray data type. A nested table permits a row to have multiple 'mini-rows' of related data contained within the one object. See "About Nested Tables" on page 12-29 for more information.

Naming Conventions for Data Objects

The rules for naming data objects depend on the naming mode you set for Warehouse Builder in the Naming Preferences section of the Preferences dialog. Warehouse Builder maintains a business and a physical name for each object stored in the repository. The business name for an object is its descriptive logical name and the physical name is the name used when Warehouse Builder generates code. See "Naming Preferences" on page 3-8 for details on how to specify a naming mode.

When you name or rename data objects, use the following naming conventions.

Naming Data Objects

In the physical naming mode, the name can be from 1 to 30 alphanumeric characters. Blank spaces are not allowed. Do not use any of the Reserved Words on page 37-1 as a name of an object.

In the business naming mode, the limit is 200 characters. The name should be unique across the object category that owns the object. For example, since all tables belong to a module, table names should be unique across the module to which they belong. Similarly, module names should be unique across the project to which they belong.

Describing Data Objects

Edit the description of the data object as necessary. The description can be between 2 and 2,000 alphanumeric characters and can contain blank spaces. Specifying a description for a data object is optional.

Supported Data Types

Table 4–2 shows the data types you can use to create and edit columns.

Table 4–2 Data Types

Data Type	Description
BINARY_DOUBLE	Stores double-precision IEEE 754-format single precision floating point numbers. Used primarily for high-speed scientific computation. Literals of this type end with `d`. For example, 3.0235`d`.
BINARY_FLOAT	Stores single-precision IEEE 754-format single precision floating point numbers. Used primarily for high-speed scientific computation. Literals of this type end with `f`. For example, 2.07`f`.
BLOB	Stores large binary objects in the database, in-line or out-of-line. Every BLOB variable stores a locator, which points to a large binary object. The size of a BLOB cannot exceed four gigabytes.
CHAR	Stores fixed-length character data to a maximum size of 4000 characters. How the data is represented internally depends on the database character set. You can specify the size in terms of bytes or characters, where each character contains one or more bytes, depending on the character set encoding.
CLOB	Stores large blocks of character data in the database, in-line or out-of-line. Both fixed-width and variable-width character sets are supported. Every CLOB variable stores a locator, which points to a large block of character data. The size of a CLOB cannot exceed four gigabytes.

Table 4–2 (Cont.) Data Types

Data Type	Description
DATE	Stores fixed-length date times, which include the time of day in seconds since midnight. The date defaults to the first day of the current month; the time defaults to midnight. The date function SYSDATE returns the current date and time.
FLOAT	Stores a single-precision, floating-point, number. FLOAT can be loaded with correct results only between systems where the representation of a FLOAT is compatible and of the same length.
INTEGER	A NUMBER subtype that stores integer values with a maximum precision of 38 decimal digits.
INTERVAL DAY TO SECOND	Stores intervals of days, hours, minutes, and seconds.
INTERVAL YEAR TO MONTH	Stores intervals of years and months.
LONG	Stores fixed-length character strings. The LONG datatype is like the VARCHAR2 datatype, except that the maximum length of a LONG value is 2147483647 bytes (two gigabytes).
MDSYS.SDOAGGRTYPE	Stores the geometric description of a spatial object and the tolerance. Tolerance is used to determine when two points are close enough to be considered as the same point.
MDSYS.SDO_DIM_ARRAY	Stores an array of type MDSYS.SDO_DIM_ELEMENT.
MDSYS.SDO_DIM_ELEMENT	Stores the dimension name, lower boundary, upper boundary and tolerance.
MDSYS.SDO_ELEM_INFO_ARRAY	Stores an array of typeMDSYS.SDO_ORDINATE_ ARRAY.
MDSYS.SDO_GEOMETRY	Stores Geographical Information System (GIS) or spatial data in the database. For more information, refer to the *Oracle Spatial Users Guide and Reference*.
MDSYS.SDO_ORDINATE_ARRAY	Stores the list of all vertices that define the geometry.
MDSYS.SDO_POINT_TYPE	Stores two dimensional and three dimensional points.
NCHAR	Stores fixed-length (blank-padded, if necessary) national character data. Because this type can always accommodate multibyte characters, you can use it to hold any Unicode character data. How the data is represented internally depends on the national character set specified when the database was created, which might use a variable-width encoding (UTF8) or a fixed-width encoding (AL16UTF16).
NCLOB	Stores large blocks of NCHAR data in the database, in-line or out-of-line.
NUMBER	Stores real numbers in a fixed-point or floating-point format. Numbers using this data type are guaranteed to be portable among different Oracle platforms, and offer up to 38 decimal digits of precision. You can store positive and negative numbers, as well as zero, in a NUMBER column.

Table 4–2 (Cont.) Data Types

Data Type	Description
NVARCHAR2	Stores variable-length Unicode character data. Because this type can always accommodate multibyte characters, you can use it to hold any Unicode character data. How the data is represented internally depends on the national character set specified when the database was created, which might use a variable-width encoding (UTF8) or a fixed-width encoding (AL16UTF16).
RAW	Stores binary data or byte strings. For example, a RAW variable might store a sequence of graphics characters or a digitized picture. Raw data is like VARCHAR2 data, except that PL/SQL does not interpret raw data.
SYS.ANYDATA	An Oracle-supplied type that can contain an instance of a given type, with data, plus a description of the type. ANYDATA can be used as a table column data type and lets you store heterogeneous values in a single column. The values can be of SQL built-in types as well as user-defined types.
SYS.LCR$_ROW_RECORD	This type represents a data manipulation language (DML) change to a row in a table. This type uses the LCR$_ROW_LIST type.
TIMESTAMP	Extends the DATE data type and stores the year, month, day, hour, minute, and second. The default timestamp format is set by the Oracle initialization parameter NLS_TIMESTAMP_FORMAT.
TIMESTAMP WITH LOCAL TIMEZONE	Extends the TIMESTAMP data type and includes a time-zone displacement. The time-zone displacement is the difference (in hours and minutes) between local time and Coordinated Universal Time (UTC)—formerly Greenwich Mean Time. You can also use named time zones, as with TIMESTAMP WITH TIME ZONE.
TIMESTAMP WITH TIMEZONE	Extends the data type TIMESTAMP and includes a time-zone displacement. The time-zone displacement is the difference (in hours and minutes) between local time and Coordinated Universal Time (UTC)—formerly Greenwich Mean Time.
VARCHAR	Stores a length-value data type consisting of a binary length subfield followed by a character string of the specified length. The length is in bytes unless character-length semantics are used for the data file. In that case, the length is in characters.
VARCHAR2	Stores variable-length character data. How the data is represented internally depends on the database character set. The VARCHAR2 data type takes a required parameter that specifies a maximum size up to 4000 characters.
XMLFORMAT	This is an object type that is used to specify formatting arguments for SYS_XMLGEN() and SYS_XMLAGG() functions.
XMLTYPE	An Oracle-supplied type that can be used to store and query XML data in the database. It has member functions you can use to access, extract, and query the XML data using XPath expressions. XPath is another standard developed by the W3C committee to traverse XML documents.

About the Data Object Editor

The Data Object Editor provides a centralized interface to create, edit, configure, validate, and deploy Oracle data objects. You can use the Data Object Editor with relational, dimensional, and business intelligence objects. You can also view the data stored in these objects.

The Data Object Editor enables you to build your warehouse schema designs. It also provides an intuitive user interface that supports fast entry of design details.

Use the Data Object Editor to:

- Create, edit, and delete relational and dimensional objects.
- Create, edit, and delete the following business intelligence objects: Business Areas and Item Folders.
- Define relationships between Oracle data objects.
- Validate, generate, and deploy Oracle data objects.
- Define and edit all aspects of a data object such as its columns, constraints, indexes, partitions, data rules, and attribute sets.
- View impact analysis and lineage information for a data object.
- Define implementation details for dimensional objects with a relational implementation.
- View the data stored in a data object.

Launching the Data Object Editor

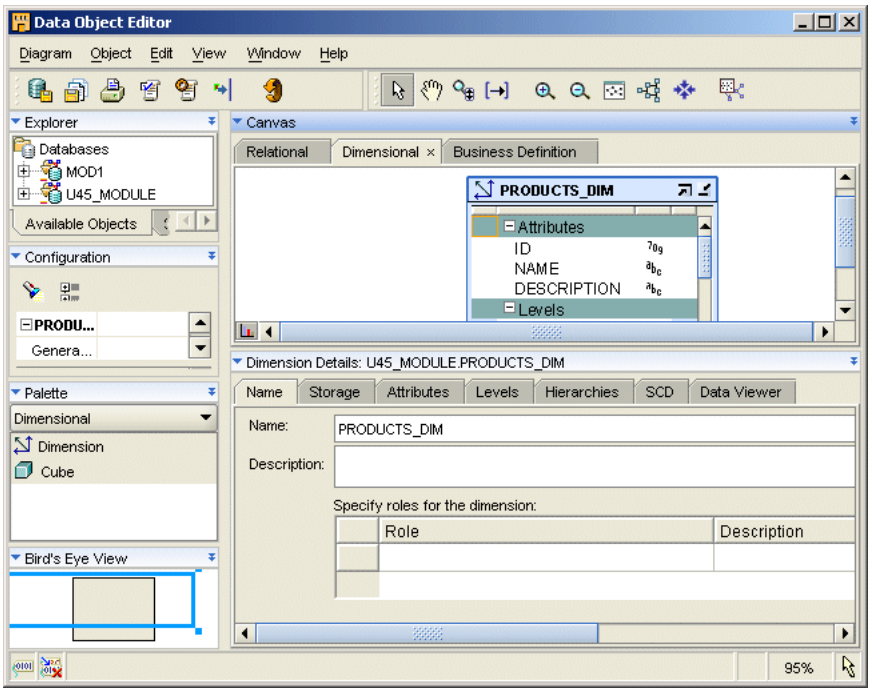
You can use one of the following methods to launch the Data Object Editor:

- Select a data object in the Project Explorer and choose **Edit**, then **Open Editor** from the Design Center menu.
- Right-click a data object in the Project Explorer and select **Open Editor**.
- Double-click a data object in the Project Explorer.

Data Object Editor Components

Figure 4–2 displays the Data Object Editor.

Figure 4–2 Data Object Editor Window



The Data Object Editor contains a menu bar, multiple toolbars, and multiple panels. All the panels are dockable. You can resize the panels or relocate them anywhere in the editor window. You can also choose to display or hide any of the panels.

To relocate a panel, hold down the mouse button on the panel title, drag to the new location and release the mouse button. Resize a panel by placing your mouse on the panel border, pressing the mouse button when the double sided arrow appears, and dragging your mouse to indicate the desired size.

The Data Object Editor has the following components:

- Title Bar
- Menu Bar
- Toolbar
- Explorer
- Palette
- Configuration
- Bird's Eye View
- Canvas
- Object Details
- Generation
- Indicator Bar

Title Bar

The title bar is located at the top of editor window. It displays the title Data Object Editor.

Menu Bar

The menu bar, situated below the title bar, provides access to the editor commands. You can access the editor commands either by clicking the menu bar options or by using hot keys. For example, to access the Diagram menu, you can click **Diagram** or type **Alt+D**.

The menu bar options enable you to create, edit, validate, and deploy data objects. The menu bar contains the following menu items:

- Diagram
- Object
- Edit
- View
- Window
- Help

Diagram

The Diagram menu provides options to create, validate, generate, and deploy relational, dimensional, and business intelligence objects. You can also perform binding and unbinding for dimensional objects. The menu items in the Diagram menu apply to the currently selected tab on the canvas.

The Diagram menu contains the following menu options:

- **Close Window:** Closes the Data Object Editor.
- **Export Diagram:** Saves the current diagram displayed on the canvas as a file on your machine either as an SVG file or JPEG file.
- **Save All:** Saves changes made to the repository.
- **Add:** Enables you to add data objects to the editor canvas. Use the options in the Add menu item to select the type of object that you want to add. The options in the Add menu item are Table, View, Materialized View, Object Type, Varray, Nested Table, Dimension, Cube, Item Folder, and Business Area.

  Notice that only some of the options in the Add menu item are enabled. The Add menu options that are enabled depend on the canvas tab that is currently selected. For example, when the Dimensional tab is selected on the canvas, the options Dimension and Cube of the Add menu are enabled. For more information on the Data Object Editor canvas, see "Canvas" on page 4-14.
- **Bind All:** Performs binding for all the dimensional objects on the canvas. This option is only enabled when the Dimensional tab is selected.

  Binding is applicable only to dimensional objects that have a a relational implementation. For more information on binding, see "Binding" on page 4-27.
- **Unbind All:** Unbinds all the dimensional objects on the canvas. Unbinding removes the mapping between a dimensional object and the database objects that store its data. This option is enabled only when the Dimensional tab of the canvas is selected.

  Unbinding is applicable only to dimensional objects that have a relational implementation.
- **Validate All:** Validates all the data objects on the canvas. The results of the validation are displayed in the Generation panel.

- **Generate All:** Generates all the data objects on the canvas. The generation results are displayed in the Generation panel.
- **Derive All:** Derives business intelligence objects for all the relational or dimensional objects on the canvas.
- **Deploy All:** Deploys all the data objects on the canvas.
- **Print Preview:** Provides a preview of the contents of the canvas before printing.
- **Print:** Prints the contents of the canvas.

Object

The Object menu contains options to validate, generate, and deploy the currently selected data object(s). You can also bind and unbind dimensional objects. You can select multiple data objects by holding down the Shift key.

The Object menu contains the following options:

- **Bind:** Binds the dimensional objects selected on the canvas to the database tables that store its data. Warehouse Builder creates the database tables that store the dimensional object data.

  You can perform binding only for dimensional objects that use a relational implementation.
- **Unbind:** Removes the bindings between the selected dimensional objects and the database tables that store their data. Warehouse Builder does not delete the tables that have been unbound. You will need to explicitly delete the tables.

  You can perform unbinding only for dimensional objects that use a relational implementation.
- **Validate:** Validates the data object selected on the canvas. The results of the validation are displayed in the Generation window.
- **Generate:** Generates the data object selected on the canvas. The results of the generation are displayed in the Generation window.
- **Derive:** Derives a business intelligence object from the selected data object.
- **Deploy:** Deploys the data object selected on the canvas.
- **Show Related:** Displays the data objects to which the selected data object is related on the canvas. For example, for a cube, selecting this option displays the dimensions that the cube references on the canvas.

Edit

The Edit menu enables you to remove, hide, copy, and paste data objects on the canvas. The Edit menu contains the following options:

- **Copy:** Copies the selected object.
- **Paste:** Pastes the data object from the clipboard on to the canvas.
- **Delete:** Deletes the data object from the repository.
- **Hide:** Removes a data object from the canvas without deleting it from the repository.
- **Select All:** Selects all the data objects on the editor canvas.
- **Refresh:** Refreshes the data displayed on the editor panels. For example, you open the Data Object Editor for the table EMPL. In the Design Center, you rename the

view EMP_VIEW to EMPL_VIEW. When you refresh the editor, the new name for the view is displayed in the Explorer panel of the Data Object Editor.

View

The View option enables you to change viewing preferences in the editor such as the zoom setting, show or hide toolbars, layout of the objects in the editor (center, fit in window, auto layout). You can also maximize or minimize data objects on the canvas.

- **Toolbars:** Enables you to select the types of toolbars that should be displayed in the editor. The options you can select are **Generic** and **Graphic**. For more information on toolbars, see "Toolbar" on page 4-12.
- **Zoom:** Increases or decreases the size of the objects displayed in the canvas. The options you can select to zoom the canvas are 400%, 200%, 100%, 75%, 50%, and 25%.
- **Fit in Window:** Arranges the objects on the canvas so that they fit on the editor canvas. The size of the objects could be increased or decreased depending on the number of objects on the canvas.
- **Auto Layout:** Displays the data objects on the canvas in their default size and layout.
- **Center:** Centers the data objects on the canvas.
- **Minimize:** Minimizes the selected data object on the canvas.
- **Restore Size:** Restores the size of a minimized data object and displays it on the canvas.
- **Maximize:** Maximizes the selected data object on the canvas.
- **Set Size to Default:** Displays the selected data object in its default size. For example, you may have resized the data object using the double-sided arrow that appears when you position the cursor at the edge of a data object. This option displays the data object in its default size.
- **Clipboard Contents:** Displays the Clipboard Contents dialog that displays details about the contents of the clipboard.
- **Enable Horizontal Scrolling:** Enables or disables horizontal scrolling for all the data objects displayed on the canvas.

Window

Use the Window menu to show or hide the panels in the Data Object Editor. The window menu contains the following options:

- **Explorer:** Shows or hides the Explorer panel of the Data Object Editor.
- **Palette:** Shows or hides the Data Object Editor Palette.
- **Bird's Eye View:** Shows or hides the Bird's Eye View panel of the Data Object Editor.
- **Generation Results:** Shows or hides the Generation Results panel of the Data Object Editor.
- **Object Details:** Shows or hides the Object Details panel of the Data Object Editor.
- **Configuration Panel:** Shows or hides the Configuration panel of the Data Object Editor.
- **Arrange All:** Arranges all the panels in the editor window.

All the options are toggle options. For example, select **Window** and then **Object Details** to show or hide the object details panel.

Help

The Help menu options provide access to the online help, Oracle Technology Network, and information about the version of Warehouse Builder. Use the **About Warehouse Builder** option to view the version of the Warehouse Builder client and the repository version.

Use the **Session Properties** option to display the Session Properties dialog. This dialog contains information about the current Warehouse Builder session such as the user details, version of the Oracle Database, and roles granted to the currently logged in user.

Toolbar

The toolbar, located below the menu bar, provides icons for the commonly used commands. It provides shortcuts to access the editor menu items. For example, the toolbar contains an icon to deploy a data object.

The Data Object Editor contains two tool bars: Generic and Graphic. The generic toolbar enables you to invoke common operations such as validation, generation, deployment, and synchronization of data objects. The graphic toolbar enables you to navigate the canvas and change the magnification of objects on the canvas.

You can hide the toolbars using the **View** option of the editor. You can also move the toolbars to any part of the editor window by clicking the drag handle on the left of the toolbar and dragging to the new position.

Explorer

When you first open the Data Object Editor, the Explorer displays on the top left corner. You can relocate the Explorer anywhere on the Data Object Editor.

The Explorer provides a view, similar to a folder structure, of the data objects that are available to be viewed or edited on the canvas. The Explorer is synchronized with the currently active tab of the Data Object Editor canvas. Thus the objects displayed by the Explorer depend on the tab that is selected on the canvas. For more information on the different tabs on the canvas, see "Canvas" on page 4-14.

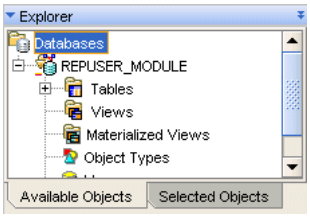
The Explorer also enables you to view the data objects that are currently displayed on the canvas along with details of their definition. For example, a table node can be expanded to display details of the constraints, indexes, and so on.

The Explorer contains two tabs: Available Objects and Selected Objects. Data objects that are selected on the Selected Objects tree are also selected on the canvas.

Available Objects This tab displays the data objects that you can view or edit on the canvas. You can view or edit a data object by dragging the object from this tab and dropping it on to the canvas. Warehouse Builder displays a node for each module. Expand the module node to view the data objects in that module.

Figure 4–3 displays the Available Objects tab of the Explorer panel.

Figure 4–3 Available Objects Tab of the Explorer Panel



The data objects displayed on this tab depend on the tab that is currently selected on the canvas. For example, a module called MOD1 contains three tables, four dimensions, and a cube. When the Relational tab is selected on the canvas, the Available Objects tab displays the three tables under the MOD1 node. when the Dimensional tab is selected in the canvas, the Available Objects tab displays the four dimensions and the cube under the MOD1 node of the Explorer.

Selected Objects The Selected Objects tab displays a node for each data object displayed on the canvas. Expand the node to view the details of the object definition. For example, when you expand a node for a database table, you will see the columns, keys, partitions, and indexes defined on the table.

Palette

When you open the Data Object Editor, the Palette is displayed on the left side of the editor window. You can relocate it anywhere in the editor window.

The Data Object Editor Palette displays icons for the type of data objects that you can drag and drop onto the canvas. Some icons in the palette may be disabled. The Palette, like the Explorer, is synchronized with the canvas. The data objects that are enabled in the Palette depend on the currently active tab on the canvas. For example, when the Relational tab is currently active in the canvas, all the relational data objects in the Palette are enabled.

At the top of the palette is a drop-down list that you can use to display only a particular type of data objects in the palette. The options in the drop-down list are: All, Relational, Dimensional, and Business Definition. Selecting a type from this drop-down list displays the icons for the objects belonging to that type only.

For example, when you select Dimensional from this drop-down list, only icons for dimensional objects are displayed in the palette.

Configuration

The Configuration panel displays the configuration properties of the data object selected on the canvas or the Selected Objects tab of the Explorer. Use this window to set or edit configuration properties for the selected data object. When you first open the Data Object Editor, the Configuration panel is displayed on the top left. You can relocate it anywhere on the editor window.

Bird's Eye View

The Bird's Eye View enables you to move the view of the canvas with a single mouse dragging operation. You can thus reposition your view of the canvas without using the scroll bars.

The Bird's Eye View displays a miniature version of the entire canvas. It contains a blue colored box that represents the portion of the canvas that is currently in focus. In

the case of mappings that span more than the canvas size, you can click the blue box and drag it to the portion of the canvas that you want to focus on.

Canvas

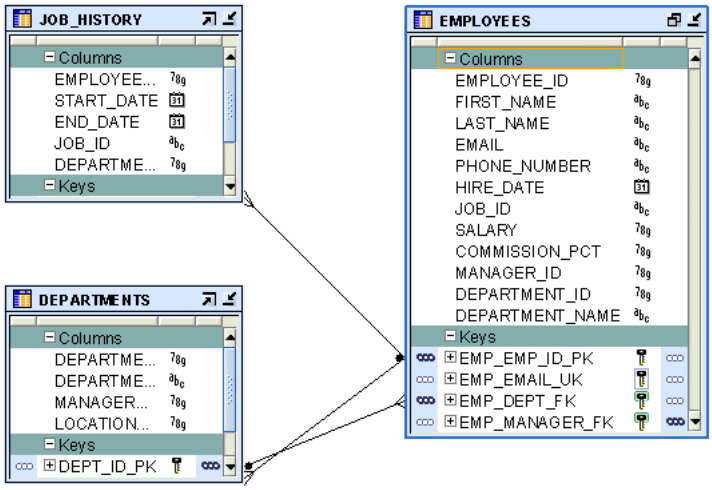
The canvas is the central graphical area that displays relational, dimensional, and business intelligence objects. The canvas is the area where data objects and their associations, if any, are represented graphically. You can also display two objects on the canvas that have no relationship to each other.

When you open the Data Object Editor for a particular object, the canvas displays the data object along with the other data objects that are related to it and represents the objects and their relationship graphically. For example, when you open the Data Object Editor for a cube, the canvas displays the cube and the dimensions that the cube references.

Each object on the canvas is represented by a node. Figure 4–4 displays the canvas for the EMPLOYEES table. The node that represents this table contains two groups: Columns and Keys. The columns group represents the columns in the table. For each column, the column name and the data type are displayed on the canvas. The Keys group displays the primary, foreign, and unique keys in the table along with the columns that make up the key.

The canvas also contains nodes for the tables DEPARTMENTS and JOB_HISTORY. The foreign key relationship between these tables can be seen.

Figure 4–4 Data Object Editor Canvas



Canvas Tabs

The canvas uses the following tabs to group data objects of similar type:

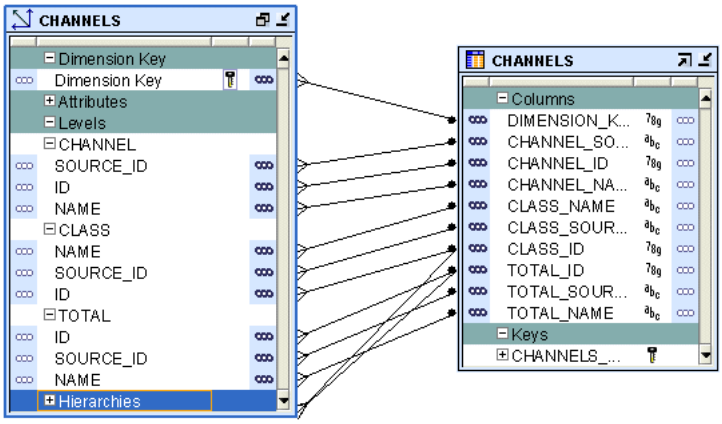
- Relational
- Dimensional
- Business Definition

All these tabs are displayed when the editor is first opened. Each tab is used to display information for the type of object that it represents. For example, when you are creating or editing a dimension, the dimension is represented graphically on the Dimensional tab. The Relational tab is used to represent tables, views, or materialized views.

In addition to these tabs, the canvas uses an additional tab that displays a detailed representation for a data object. You can view a detailed representation for any data object on the canvas. On the canvas, right-click the node that represents the data object and select **Detail View**.

Figure 4–5 displays the detail view of the CHANNELS dimension. The dimension data is stored in the table CHANNELS. The relationship depicts the table column that stores the dimension attribute data.

Figure 4–5 Detail View of a Dimension



Canvas Icons

At the bottom of the Canvas panel is an icon that you use to display the graph. In addition to this, the detail view contains two icons to display the lineage and impact analysis.

Performing Operations on a Data Object Using the Canvas

The shortcut menu on the canvas enables you to perform operations such as creating, validating, generating, and deploying data objects. The Menu Bar section on page 4-9 described how you perform the same operations using the menu bar.

Right-click the title bar of the node that represents the data object. Warehouse Builder displays a shortcut menu with the following options:

- **Generate:** Generates the selected data object. The results are displayed in the Generation panel.
- **Validate:** Validates the selected data object. The results are displayed in the Generation panel.
- **Deploy:** Deploys the selected data object.
- **Derive:** Derives a business intelligence object using the selected data object. For more information on deriving intelligence objects, see "Deriving Business Intelligence Objects" on page 15-20.
- **Copy:** Copies the selected data object. You can then paste this data object in a different module or project. You can then paste this object on the canvas thus creating a new object in the same schema as the original.
- **Hide Object:** Hides the data object on the canvas. The data object is not deleted.
- **Remove from repository:** Deletes the data object from the repository.

- **Detail View:** Creates a new tab in the canvas that contains a detail view of the selected data object. For dimensional objects with a relational implementation, this tab displays the implementation details. Use this tab to perform manual binding for dimensional objects.
- **Show Related:** Displays objects related to the selected data object on the canvas.
- **Auto Bind:** Performs auto binding for dimensional objects. This options is enabled only for dimensional objects that use a relational implementation. For more information on auto binding, see "Binding" on page 4-27.
- **Unbind:** Removes the bindings between the selected dimensional object and the data object that stores the dimensional object data. For more information on auto binding, see "Binding" on page 4-27.

You can also use the shortcut menu of the canvas to add data objects to the canvas. Right-click any blank area (whitespace) on the canvas. This displays a list of objects that you can add to the canvas in the current context. The list of items differs based on the tab that is currently selected on the canvas. For example, when you right-click a blank area on the Dimensional tab, you see options to add cubes and dimensions. The shortcut menu of the Relational tab provides options to add tables, views, materialized views, object types, varrays, and nested tables.

Object Details

The Details panel contains a set of tabs that you use to define the data object. The title of the Details panel contains the type, module name, and name of the data object. These details are suffixed by the mode in which the data object is open. For example, if you open the Data Object Editor for a table called DEPT which is stored in the HR module, the name of the window is Table Details: HR.DEPT "Read/Write". If no object is selected on the canvas, the Details panel displays "Details: No object selected".

The tabs displayed in the Details panel depend on the type of data object selected on the canvas. For example, the number and names of tabs for a table are different from that for a dimension. Use the tabs on the Details panel to define or edit the data object definition. Any changes you make on the Details panel are stored in the repository. The canvas and Explorer are also refreshed to reflect the changes.

For more information on the tabs for each data object, refer to the following sections:

- Creating Dimensions on page 13-1
- Creating Cubes on page 13-20
- Editing a Business Area on page 15-12
- Editing an Item Folder on page 15-5
- Creating Table Definitions on page 12-13
- Creating View Definitions on page 12-19
- Creating Materialized View Definitions on page 12-20
- Creating Object Types on page 12-27
- Creating Varrays on page 12-29
- Creating Nested Tables on page 12-30

Generation

The Generation panel displays the generation and validation results for a data object. This panel is hidden when you first open the editor window. It is displayed the first time you generate or validate a data object. You can to show or hide the Generation panel by selecting **Window** and then **Generation Results** from the editor menu.

The Generation window contains two tabs: Script and Message. The Script tab displays the scripts generated by Warehouse Builder to implement the data object selected in the canvas. The Message tab displays the validation messages for the selected data object. Double-click a message to view the complete message text.

Indicator Bar

Along the lower edge of the editor you see mode icons and indicators as shown in Figure 4–6. The left corner contains the Naming Mode and the Rename Mode icon indicators. Position your mouse on an icons to display the current setting for the mode. The right side displays the zoom indicator and the navigation mode.

Figure 4–6 Indicator Bar of the Data Object Editor

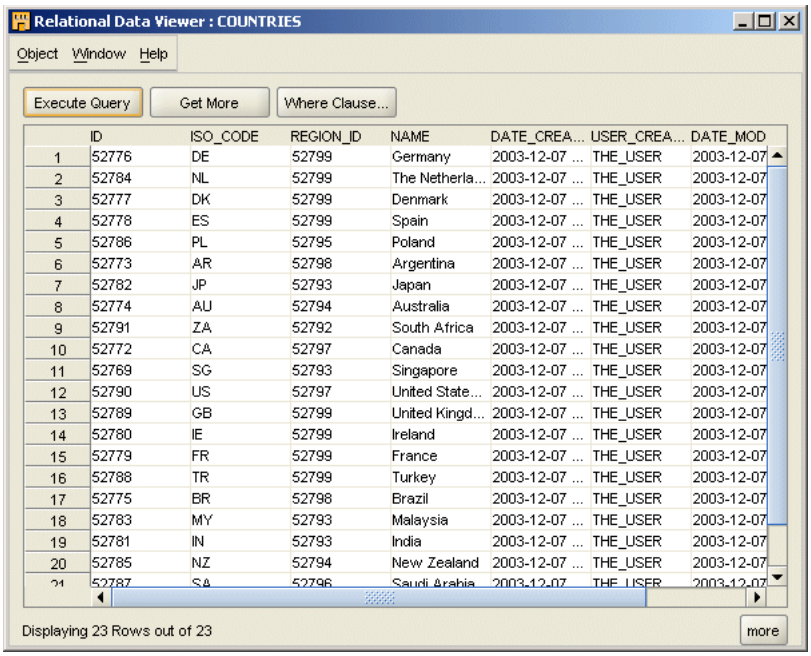
Data Viewer

The Data Viewer enables you to view the data stored in the data object. For example, the data viewer for a table enables you to view the table data. You can access the Data Viewer using one of the following methods:

- From the Project Explorer, right-click a data object and select **Data**.
- In the Data Object Editor of the data object, navigate to the Data Viewer tab of the Details panel. Click the **Execute Query** button.

Figure 4–7 displays the data viewer for the COUNTRIES table.

Figure 4–7 Data Viewer for a Table



	ID	ISO_CODE	REGION_ID	NAME	DATE_CREA...	USER_CREA...	DATE_MOD
1	52776	DE	52799	Germany	2003-12-07 ...	THE_USER	2003-12-07
2	52784	NL	52799	The Netherla...	2003-12-07 ...	THE_USER	2003-12-07
3	52777	DK	52799	Denmark	2003-12-07 ...	THE_USER	2003-12-07
4	52778	ES	52799	Spain	2003-12-07 ...	THE_USER	2003-12-07
5	52786	PL	52795	Poland	2003-12-07 ...	THE_USER	2003-12-07
6	52773	AR	52798	Argentina	2003-12-07 ...	THE_USER	2003-12-07
7	52782	JP	52793	Japan	2003-12-07 ...	THE_USER	2003-12-07
8	52774	AU	52794	Australia	2003-12-07 ...	THE_USER	2003-12-07
9	52791	ZA	52792	South Africa	2003-12-07 ...	THE_USER	2003-12-07
10	52772	CA	52797	Canada	2003-12-07 ...	THE_USER	2003-12-07
11	52769	SG	52793	Singapore	2003-12-07 ...	THE_USER	2003-12-07
12	52790	US	52797	United State...	2003-12-07 ...	THE_USER	2003-12-07
13	52789	GB	52799	United Kingd...	2003-12-07 ...	THE_USER	2003-12-07
14	52780	IE	52799	Ireland	2003-12-07 ...	THE_USER	2003-12-07
15	52779	FR	52799	France	2003-12-07 ...	THE_USER	2003-12-07
16	52788	TR	52799	Turkey	2003-12-07 ...	THE_USER	2003-12-07
17	52775	BR	52798	Brazil	2003-12-07 ...	THE_USER	2003-12-07
18	52783	MY	52793	Malaysia	2003-12-07 ...	THE_USER	2003-12-07
19	52781	IN	52793	India	2003-12-07 ...	THE_USER	2003-12-07
20	52785	NZ	52794	New Zealand	2003-12-07 ...	THE_USER	2003-12-07
21	52787	SA	52796	Saudi Arabia	2003-12-07 ...	THE_USER	2003-12-07

The Data Viewer tab contains three buttons: Execute Query, Get More, and Where Clause. When you navigate to this tab, it displays only these buttons. Click the **Execute Query** button to execute a query on the data object and fetch its data.

Use the **Get More** button to fetch more data from the data object. By default, the Data Viewer displays the first hundred rows of data. To see the remaining data use the Get More button. Alternatively, you can use the More button at the bottom of the Data Viewer to perform the same action.

The **Where Clause** button is used to specify a condition that restricts the data displayed by the Data Viewer. This option is applicable for tables and views only. Clicking this button displays the Where Clause dialog. Use this dialog to specify the condition used to filter data.

Using the Data Object Editor

Use the Data Object Editor to create relational, dimensional, and certain business intelligence objects. There are multiple methods of creating data objects using the Data Object Editor. The Data Object Editor should be open to use any of these methods.

Use one of the following editor components to create a data object:

- Menu Bar

  See "Creating a Data Object Using the Menu Bar" on page 4-19.

- Canvas

  See "Creating a Data Object Using the Canvas" on page 4-19.

- Data Object Editor Palette

  See "Creating a Data Object Using the Data Object Editor Palette" on page 4-20.

Creating a Data Object Using the Menu Bar

To create a data object using the menu bar:

1. Open the Data Object Editor.

   Use any of the methods described in "Launching the Data Object Editor" on page 4-7.

2. Navigate to the tab that corresponds to the type of data object that you want to create.

   For example, to create a table, select the Relational tab. To create a business area, select the Business Intelligence tab.

3. From the **Diagram** menu, select **Add**, then select the type of data object to create.

   Warehouse Builder displays the Add a New or Existing <Object> dialog. For details on this dialog, see "Add a New or Existing Data Object Dialog" on page 4-20.

   Notice that the list of data objects in the Add menu contains some disabled items. Only the data objects that you can create from the current editor context are enabled.

4. Select the **Create a new <object>** option.

   For example, to add a table, select the **Create a new Table** option.

5. Specify the name of the data object using the New <Object> Name field.

   The New <Object> Name field displays a default name for the object. You can choose to retain this default name or specify a different name.

6. Click **OK**.

   Warehouse Builder adds a node for the new data object to the canvas.

7. Use the tabs of the Details panel to define the data object.

Creating a Data Object Using the Canvas

To create a data object using the canvas:

1. Open the Data Object Editor.

   Use any of the methods described in "Launching the Data Object Editor" on page 4-7.

2. Navigate to the tab that corresponds to the type of data object that you want to create.

   For example, to create a materialized view, select the Relational tab. To create a dimension, select the Dimensional tab.

3. Right-click whitespace (blank area) on the canvas.

   Warehouse Builder displays a shortcut menu that contains the type of data objects you can create. For details on this dialog, see "Add a New or Existing Data Object Dialog" on page 4-20.

4. Select the option corresponding to the type of object you want to create.

   For example, to create a materialized view, select the **Add a Materialized View** option. Warehouse Builder displays the Add a New or Existing <Object> dialog.

5. Select the **Create a new <object>** option.

For example, to add a cube, select the **Create a new Cube** option.

6. Specify the name of the data object using the New <Object> Name field.

   The New <Object> Name field displays a default name for the object. You can choose to retain this default name or specify a different name.

7. Click **OK**.

   Warehouse Builder adds a node for the new data object to the canvas.

8. Use the tabs of the Details panel to define the data object.

Creating a Data Object Using the Data Object Editor Palette

To create a data object using the Palette:

1. Open the Data Object Editor.

   Use any of the methods described in "Launching the Data Object Editor" on page 4-7.

2. Navigate to the tab that corresponds to the type of data object that you want to create.

   For example, to create a view, select the Relational tab. To create a cube, select the Dimensional tab.

3. Drag and drop the operator that corresponds to the type of object that you want to create on to the canvas.

   For example, to create a view, drag and drop the View operator from the Palette on to the canvas.

   Warehouse Builder displays a shortcut menu that contains the type of data objects you can create. For details on this dialog, see "Add a New or Existing Data Object Dialog" on page 4-20.

4. Select the **Create a new <object>** option.

   For example, to add a cube, select the **Create a new Cube** option.

5. Specify the name of the data object using the New <Object> Name field.

   The New <Object> Name field displays a default name for the object. You can choose to retain this default name or specify a different name.

6. Click **OK**.

   Warehouse Builder adds a node for the new data object to the canvas.

7. Use the tabs of the Details panel to define the data object.

Add a New or Existing Data Object Dialog

The Add a New or Existing <data object> dialog enables you to add new or existing data objects to the Data Object Editor canvas. The new data objects added using this dialog are stored in the repository.

Select one of the following options:

- Create a New <data object>
- Select an existing <data object>

Create a New <data object> Use this option to create a new Oracle data object such as a table, dimension, or item folder. Enter the name of the data object in the **New <data object> Name** field. Warehouse Builder displays <data object type>_1 as the default name for the data object. You can change this default name by selecting the name and entering the new name.

Use the **Oracle Module** drop-down list to select the Oracle module to which the data object should belong. For relational and dimensional objects, Warehouse Builder initially displays the module to which the data object that is currently open in the editor belongs. You can change this selection. For business intelligence object, Warehouse Builder displays the business definition module to which the data object belongs.

For Item Folders, an additional drop-down list called **Item Folder Type** is displayed. Use this drop-down list to indicate whether you want to create a simple or a complex item folder.

After specifying these details for the data object, click **OK**. Warehouse Builder creates the data object and adds a node for this data object on the editor canvas. Use the tabs in the Details window to define the data object.

Select an existing <data object> Use this option to add an existing repository data object to the editor canvas. You may want to do this to define relationships between data objects. For example, you may want to manually bind the dimension attributes to the database columns that store their data.

To search for a data object by name, type the name of the data object in the **Search For** field and click **Go**. Alternatively, you can select the name of the data object from the selection tree displayed below the Search For field.

After you select the data object, click **OK**. Warehouse Builder adds a node for this data object on the editor canvas.

Configuring Data Objects

Configuration defines the physical characteristics of data objects. For example, you can define a tablespace and set performance parameters in the configuration of a table.

You can change the configuration of an object any time prior to deployment.

You can define multiple configurations for the same set of objects. This feature is useful when deploying to multiple environments, such as test and production.

All objects have a Deployable parameter, which is selected by default. To prevent an object from being deployed, clear this parameter.

To configure an object:

1. In the Project Explorer, select the object and click the Configure icon.

   The Configuration Properties dialog box is displayed.

2. Select a parameter to display its description at the bottom of the right panel. Click **Help** for additional information.

3. Enter your changes and click **OK**.

About Attribute Sets

An attribute set contains a chosen set of columns in the order you specify. Attribute sets are useful while defining a mapping or during data import and export. Warehouse

Builder enables you to define attribute sets for tables, views, and materialized views. For the sake of brevity, in the following sections, the word *table* refers to all objects for which you can define attribute sets

For each table, Warehouse Builder generates a predefined attribute set containing all the columns in that table. In addition, Warehouse Builder generates predefined attribute sets for each defined constraint. Predefined attribute sets cannot be modified or deleted.

For more information about creating and editing attribute sets, refer to the following:

- Creating Attribute Sets on page 12-23
- Editing Attribute Sets on page 12-24

About Constraints

Constraints are used to enforce the business rules you want to associate with the information in a database. Constraints prevent the entry of invalid data into tables. Business rules specify conditions and relationships that must always be true, or must always be false. In Warehouse Builder, you can create constraints for tables, views, and materialized views.

For example, if you define a constraint for the salary column of the employees table as Salary < 10,000, this constraint enforces the rule that no row in this table can contain a numeric value greater than 10,000 in this column. If an INSERT or UPDATE statement attempts to violate this integrity constraint, then Oracle displays an error message. Keep in mind that constraints slow down load performance.

You can define the following constraints for tables, views, and materialized views:

- **Unique Key (UK):** A UNIQUE key constraint requires that every value in a column or set of columns (key) be unique. No two rows of a table can have duplicate values in a specified column or set of columns. A UK column can also contain a null value.
- **Primary Key (PK):** A value defined on a key (column or set of columns) specifying that each row in the table can be uniquely identified by the values in the key (column or set of columns). No two rows of a table can have duplicate values in the specified column or set of columns. Each table in the database can have only one PK constraint. A PK column cannot contain a null value.
- **Foreign Key (FK):** A rule defined on a key (column or set of columns) in one table that guarantees that the values in that key match the values in a PK or UK key (column or set of columns) of a referenced table.
- **Check Constraint:** A user-defined rule for a column (or set of columns) that restricts inserts and updates of a row based on the value it contains for the column (or set of columns). A Check condition must be a Boolean expression which is evaluated using the values in the row being inserted or updated. For example, the condition *Order Date < Ship Date* will check that the value of the Order Date column is always less than that of the Ship Date column. If not, there will be an error when the table is loaded and the record will be rejected. A check condition cannot contain subqueries and sequences or SYSDATE, UID, USER, or USERENV SQL functions. While check constraints are useful for data validation, they slow down load performance.

For information on using constraints in a table, refer to the following sections:

- Creating Constraints on page 12-2

- Creating Constraints on page 12-2

About Indexes

Use indexes to enhance query performance of your data warehouse. In Warehouse Builder, you can create indexes for tables, materialized views, dimensions, and cubes. For the sake of brevity, in the following sections, the word *table* refers to all objects for which you can define indexes.

Indexes are important for speeding queries by quickly accessing data processed in a warehouse. You can create indexes on one or more columns of a table to speed SQL statement execution on that table. Indexes have the following characteristics:

- Indexes provide pointers to the rows in a table that contain a given key value.
- Index column values are stored presorted.
- Because the database stores indexes in a separate area of the database, you can create and drop indexes at any time without effecting on the underlying table.
- Indexes are independent of the data in the table. When you delete, add, or update data, the indexes are maintained automatically.

To learn more about indexes and indexing strategies, see the Oracle 10*g Data Warehousing Guide.*

Creating Indexes

In general, you define indexes by using the Indexes tab in the Data Object Editor. To launch the Data Object Editor, navigate to the table or other data object on the Project Explorer and double-click it or right-click and select **Open Editor.** When you select an index type, Warehouse Builder displays the appropriate template enabling you to define the index.

For all types of indexes except bitmap indexes, you can determine whether or it inherits the partitioning method of the underlying table. An index that inherits its partitioning method is known as a *local index* while an index with its own partitioning method is known as a *global index.* For additional information, see "Index Partitioning" on page 12-11.

You can create the following types of indexes in Warehouse Builder:

- **Unique:** These indexes ensure that no two rows of a table have duplicate values in the key column or composite key columns.
- **Normal:** Also known as non unique indexes, these are b-tree type indexes that do not impose restrictions against duplicate values in the key column or composite key columns.
- **Bitmap:** These indexes are primarily used for data warehousing applications to enable the querying of large amounts of data. These indexes use bitmaps as key values instead of a list of row ids. Bitmaps are effective when the values for the index key comprise a small list. For example, AGE_GROUP could be a good index key but AGE would not.

  Bitmaps enable star query transformations which are cost-based query transformations aimed at efficiently executing star queries. A prerequisite of the star transformation is that a bitmap index must be built on each of the foreign key columns of the cube or cubes.

When you define a bitmap index in Warehouse Builder, set its scope to LOCAL and partitioning to NONE.

- **Function-based:** These indexes compute and store the value of a function or expression you define on one or more columns in a table. The function can be an arithmetic expression that contains a PL/SQL function, package function, C callout, or SQL function.
- **Composite:** Also known as concatenated indexes, these are indexes on multiple columns. The columns can be in any order in the table and need not be adjacent to each other.

  To define a composite index in Warehouse Builder, create the index as you would any other index and assign between 2 and 32 index columns.
- **Reverse:** For each indexed column except for the rowid column, this index reverses the bytes in the columns. Since rowid is not reversed, this index maintains the column order.

  To define a reverse index in Warehouse Builder, create the index as you would any other index and then go to the configurations window and set Index Sorting listed under the Performance Parameters to REVERSE.

About Partitions

Partitions enable you to efficiently manage very large tables and indexes by dividing them into smaller, more manageable parts. Partitions improve query and load performance because operations work on subsets of data. Use partitions to enhance data access and improve overall application performance- especially for applications that access tables and indexes with millions of rows and many gigabytes of data.

In Warehouse Builder, you can define partitions for tables, indexes, materialized views, and MOLAP cubes. For the sake of brevity, in the following sections, the word *table* refers to all objects for which you can define partitions. The following sections discuss partitioning for all the objects previously listed with the exception of partitions MOLAP cubes which is described separately.

You define partitions for these objects by using the Partitions tab in the Data Object Editor. To launch the Data Object Editor, navigate to the object on the Project Explorer and double-click it or right-click and select **Edit.** Depending on the type of partition you create, you may also need to configure tablespaces for the partitions in the configuration window.

You can create the following types of partitions:

- **Range Partitioning** on page 12-5: Use range partitioning to create partitions based on a range of values in a column. When you use range partitioning with a date column as the partition key, you can design mappings that instantly update target tables, as described in "Improved Performance Through Partition Exchange Loading" on page 8-24.
- **Hash Partitioning** on page 12-6: Use hash partitioning to direct the Oracle Database to evenly divide the data across a recommended even number of partitions. This type of partitioning is useful when data is not historical and there is no obvious column or column list.
- **Hash By Quantity Partitioning** on page 12-7: As a means of quickly defining hash partitioning, use Hash by Quantity partitioning. This the same as hash partitioning except that you specify only a partition key and the number of

partitions and Warehouse Builder creates and names the partitions for you. You can then configure the partitions to share the same tablespace list.

- **List Partitioning** on page 12-7: Use list partitioning to explicitly assign rows to partitions based on a partition key you select. This enables you to organize the data in a structure not available in the table.
- **Composite Partitioning** on page 12-9: You can use Warehouse Builder to specify a composite of either range-hash, range-hash by quantity, or range-list partitioning. The Oracle Database first performs the range partitioning and then further divides the data using the second partitioning you select. For example, in range-list partitioning, you can base partitions on the sales transaction date and then further divide the data based on lists of states where transactions occurred.
- **Index Partitioning** on page 12-11: You can define an index that inherits the partitioning method of its underlying table. Or, you can partition an index with its own partitioning strategy.

About Dimensional Objects

Objects that contain additional metadata to identify and categorize data are called dimensional objects. Warehouse Builder enables you to design, deploy, and load two types of dimensional objects: dimensions and cubes. In this chapter, the word dimensional object refers to both dimensions and cubes.

Most analytic queries require the use of a time dimension. Warehouse Builder provides tools that enable you to easily create and populate time dimensions by answering simple questions.

Design versus Implementation

Warehouse Builder separates the logical design of dimensional objects from their storage. The logical design (business rules) allow you to focus on the structure and the content of the dimensional object first. You can then choose a relational, ROLAP, or MOLAP implementation for the dimensional object.

ROLAP and relational implementations store the dimensional object in a relational schema in the database.

A MOLAP implementation stores the dimensional object in analytic workspaces in the database.

Warehouse Builder enables you to use the same metadata to create and manage both your relational and multidimensional data stores. Separating the design from the implementation has the following advantages:

- Implementation is easier, because you first design and then implement.
- ETL is transparent as it is always the same for any type of implementation.

Uses of OLAP

Business organizations typically have complex analytic, forecast, and planning requirements. Analytic Business Intelligence (BI) applications provide solutions by answering critical business questions using the data available in your database.

Dimensional objects provide complex analytic power to your data warehouse. After you load data into dimensional objects, you can use tools and applications to run complex analytical queries that answer your business questions. These analytic queries include time-series analysis, inter-row calculations, access to aggregated historical and

current data, and forecasts. Multidimensional objects are more effective in answering these types of queries quickly.

About Creating Dimensional Objects

Creating dimensional objects consists of four high-level tasks:

1. Defining Dimensional Objects
2. Implementing Dimensional Objects
3. Deploying Dimensional Objects
4. Loading Dimensional Objects

Defining Dimensional Objects

When you define dimensional objects, you describe the logical relationships that help store data in a more structured format. For example, to define a dimension, you describe its attributes, levels, and hierarchies.

Warehouse Builder provides the following two methods to define dimensional objects:

- **Wizards:** Use wizards to create dimensional objects easily. The wizard creates a fully functional dimensional object along with the implementation objects that store the dimensional object data. Many options are defaulted to the most common settings. You can change these settings later using the editors.

  You use the Create Dimension wizard to create dimensions, the Create Time Dimension wizard to create time dimensions, and the Create Cube wizard to create cubes.

- **Editors:** Use editors to create or edit dimensional objects. Use editors to create a dimensional object when you want to specify settings that are different from the default settings used by the wizards. Also use editors to create dimensional objects that use certain advanced options that are not available when you use wizards. For example, to create a relational dimension that uses a snowflake schema implementation, you must use the editor. When you use the wizard, the default implementation method used is the star schema.

Implementing Dimensional Objects

To implement a dimensional object is to create the physical structure of the dimensional object. Warehouse Builder provides the following implementations for dimensional objects:

- Relational Implementation of Dimensional Objects
- ROLAP Implementation of Dimensional Objects
- MOLAP Implementation of Dimensional Objects

You set the Deployment Option configuration property to specify the type of implementation for a dimensional object. For more setting this property, see "Configuring Dimensions" on page 13-19 and "Configuring Cubes" on page 13-35.

Relational Implementation of Dimensional Objects

A relational implementation stores the dimensional object and its data in a relational form in the database. The dimensional object data is stored in implementation objects that are typically tables. Any queries that are executed on the dimensional object obtain data from these tables.

Warehouse Builder creates the DDL scripts that create the dimensional object. You can then deploy these scripts to the database using the Control Center.

When you use the wizard to define dimensional objects, Warehouse Builder creates the database tables that store the dimensional object data. When you define a dimensional object using the Data Object Editor, you can decide whether you want Warehouse Builder to create the implementation tables or you want to store the dimensional object data in your own tables and views. The following section on binding describes how you specify the relationship between the dimensional object and its implementation objects.

Binding Binding is the process of connecting the attributes and relationships of the dimensional object to the columns in the table or view that store their data. You perform binding only for dimensional objects that have a relational implementation. For multidimensional objects, binding is implicit and is resolved in the analytic workspace.

Warehouse Builder provides two methods of binding:

- Auto binding
- Manual binding

Auto Binding In auto binding, Warehouse Builder binds the attributes and relationships of the dimensional object to the database columns that store their data. You can perform auto binding using both the wizards and the editors.

When you use the wizard to create dimensional objects, Warehouse Builder creates the implementation tables and then performs auto binding. In the case of a dimension, the number of tables used to store the dimension data depends on the options you select for the storage. For more information on these options, see "Relational and ROLAP Implementation of a Dimension" on page 4-35.

When you use the editors to create dimensional objects, you can perform both auto binding and manual binding.

Manual Binding In manual binding, you must explicitly bind the attributes of the dimensional objects to the database columns that store their data. You use manual binding when you want to bind a dimensional object to existing tables or views.

Unbinding Warehouse Builder also enables you to unbind a dimensional object. Unbinding removes the connections between the dimensional object and the tables that store its data.

To unbind a dimensional object from its current implementation, right-click the dimensional object on the Relational tab of the Canvas and select **Unbind**. Unbinding removes the bindings between the dimensional object and its implementation objects. However, it does not modify the implementation objects.

ROLAP Implementation of Dimensional Objects

A ROLAP implementation, like a relational implementation, stores the dimensional object and its data in a relational form in the database. In addition to creating DDL scripts that can be deployed to a database, a ROLAP implementation enables you to create CWM2 metadata for the dimensional object in the OLAP catalog.

When you use the wizard to define dimensional objects, Warehouse Builder creates the database tables that store the dimensional object data. When you define a dimensional object using the Data Object Editor, you can decide whether you want Warehouse Builder to create the implementation tables or you want to store the dimensional object

data in your own tables and views. For more information about how you specify the relationship between the dimensional object and its implementation objects, see "Binding" on page 4-27.

MOLAP Implementation of Dimensional Objects

In a MOLAP implementation, the dimensional object data is stored in an analytic workspace in Oracle Database 10*g*. This analytic workspace, in turn, is stored in the database.

Note: To use a MOLAP implementation, you must have the following:

- Oracle Database 10*g* Enterprise Edition with the OLAP option
- OLAP 10.1.0.4 or higher

Analytic Workspace An analytic workspace is a container within the Oracle Database that stores data in a multidimensional format. Analytic workspaces provide the best support to OLAP processing. An analytic workspace can contain a variety of objects such as dimensions and variables.

An analytic workspace is stored in a relational database table, which can be partitioned across multiple disk drives like any other table. You can create many analytic workspaces within a single schema to share among users. An analytic workspace is owned by a particular user and other users can be granted access to it. The name of a dimensional object must be unique within the owner's schema. For more information about analytic workspaces, refer to Chapter 6, Understanding Data Storage, of the *Oracle OLAP Application Developer's Guide 10g Release 2(10.2)*.

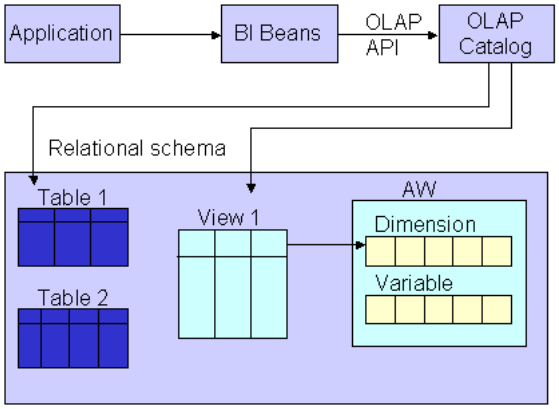
OLAP Catalog The OLAP catalog is the metadata repository provided for the OLAP option in the Oracle Database. This metadata describes the data stored in both relational tables and in analytic workspaces.

When you deploy a dimensional object using Warehouse Builder, you can specify if the dimensional object metadata should be stored in the OLAP catalog.

OLAP metadata is dynamically projected through a series of views called the active catalog views (views whose names begin with ALL_OLAP2_AW).

In Oracle Database 10*g*, the OLAP catalog metadata is used by OLAP tools and applications to access data stored in relational star and snowflake schemas. External application such as BI Beans and Discoverer use the OLAP catalog to query relational and multidimensional data. The application does not need to be aware of whether the data is located in relational tables or in analytic workspaces, nor does it need to know the mechanism for accessing it. Figure 4–8 describes how the OLAP catalog enables applications to access data stored in relational tables and analytic workspaces.

Figure 4–8 Using the OLAP Catalog to Access Dimensional Objects



The OLAP catalog uses the metadata it stores to access data stored in relational tables or views. The OLAP catalog defines logical multidimensional objects and maps them to the physical data sources. The logical objects are dimensions and cubes. The physical data sources are columns of a relational table or view.

Deploying Dimensional Objects

To instantiate the dimensional objects in the database, you must deploy them. Warehouse Builder provides the following deployment options for dimensional objects.

- **Deploy All:** For a relational or ROLAP implementation, the dimensional object is deployed to the database and a CWM definition to the OLAP catalog. For a MOLAP implementation, the dimensional object is deployed to the analytic workspace.
- **Deploy Data Objects Only:** Deploys the dimensional object only to the database. You can select this option only for dimensional objects that use a relational implementation.
- **Deploy to Catalog Only:** Deploys the CWM definition to the OLAP catalog only. Use this option if you want applications such as BI Beans or Discoverer for OLAP to access the dimensional object data after you deploy data only.
- **Deploy Aggregation:** Deploys the aggregations defined on the cube measures. This option is available only for cubes.

Deploying Dimensional Objects that Use a MOLAP Implementation

Dimensional objects that use a MOLAP implementation can be deployed just after you define them. You can use the Design Center or the Control Center Manager to deploy a dimensional object.

Deploying Dimensional Objects that Use a Relational or ROLAP Implementation

Before you deploy a relational or ROLAP dimensional object, ensure that the implementation details are specified. This means that the dimensional object should be bound to its implementation objects. Also ensure that the dimensional object is valid. For more information on implementing dimensional objects, see "Relational Implementation of Dimensional Objects" on page 4-26. For more information on performing binding, see "Binding" on page 4-27.

After you perform binding, deploy the dimensional object. Before you deploy a dimensional object, ensure that all its implementation objects are deployed. For a dimension, this includes the sequence that is used to generate the surrogate identifier of the dimension levels. Alternatively, you can deploy the implementation objects together with the dimensional object.

Loading Dimensional Objects

After you deploy a dimensional object, you load data into it by creating a mapping. Use the Mapping Editor to create the mapping that loads data from the source objects into the dimensional object. You then deploy and execute this mapping.

For more information on loading dimensions, see "Dimension Operator as a Target" on page 25-17. For information on loading cubes, see "Cube Operator" on page 25-10.

About Dimensions

A dimension is a structure that organizes data. Examples of commonly used dimensions are Customers, Time, and Products.

For relational dimensions, using dimensions improves query performance because users often analyze data by drilling down on known hierarchies. An example of a hierarchy is the Time hierarchy of year, quarter, month, day. The Oracle Database uses these defined hierarchies by rewriting queries that retrieve data from materialized views rather than detail tables.

Typical relational dimension tables have the following characteristics:

- A single column primary key populated with values called warehouse keys.

  Warehouse keys that provide administrative control over the dimension, support techniques that preserve dimension history, and reduce the size of cubes.

- One or more hierarchies that are explicitly defined as dimension objects. Hierarchies maximize the number of query rewrites by the Oracle server.

Rules for Dimension Objects

When you create a dimension object using Warehouse Builder, the dimension must conform to the following rules:

- A dimension must have a surrogate identifier and a business identifier.
- The surrogate identifier can consist of only one attribute. However, the business identifier can consist of more than one attribute.
- Every dimension level must have at least one attribute.
- A dimension attribute can play only one of the following role at a time: effective date, expiration date, triggering attribute, surrogate identifier, business identifier, or parent identifier.
- A dimension that uses a relational or ROLAP implementation must have at least one level.
- Any database table or view that implements a dimension that uses a relational or ROLAP implementation must have only one LONG, LONG RAW, or NCLOB column.
- For a dimension that uses a relational or ROLAP implementation, all level attributes must bind to database tables or views only.

- A dimension that uses a relational or ROLAP implementation must be associated with a sequence that is used to load the dimension key attribute.
- The dimension key attribute of a dimension that uses a relational or ROLAP implementation must bind to the primary key of a table.
- A Type 2 SCD must have the effective date, expiration date, and at least one triggering attribute.
- A Type 3 SCD must have the effective date and at least one triggering attribute.

Defining a Dimension

A dimension consists of a set of levels and a set of hierarchies defined over these levels. To create a dimension, you must define the following:

- Dimension Attributes
- Levels
- Level attributes
- Hierarchies

Defining Dimension Attributes

A dimension attribute is a descriptive characteristic of a dimension member. It has a name and a data type. A dimension attribute is applicable to one or more levels in the dimension. They are implemented as level attributes to store data.

In Warehouse Builder, you define dimension attributes when you define a dimension. The list of dimension attributes must include all the attributes that you may need for any of the levels in the dimension. Dimension attributes are the only attributes that are visible in Discoverer and other OLAP tools.

For example, the Products dimension has a dimension attribute called Description. This attribute is applicable to all the levels Total, Groups, and Products and stores the description for each of the members of these levels.

Defining Levels

The levels in a dimension represent the level of aggregation of data. A dimension must contain at least one level, except in the case of a dimension that contains a value-based hierarchy. Every level must have level attributes and a level identifier.

For example, the dimension Products can have the following levels: Total, Groups, and Product.

Surrogate, Business, and Parent Identifiers

Every level must have two identifiers: a surrogate identifier and a business identifier. When you create a dimension, each level must implement the dimension attributes marked as the surrogate identifier and business identifier (attributes, in the case of a composite business identifier) of the dimension.

Surrogate Identifiers A surrogate identifier uniquely identifies each level record across all the levels of the dimension. It must be composed of a single attribute. Surrogate identifiers enable you to hook facts to any dimension level as opposed to the lowest dimension level only.

For a dimension that has a relational or ROLAP implementation, the surrogate identifier should be of the data type `NUMBER`. Because the value of the surrogate

identifier must be unique across all dimension levels, you use the same sequence to generate the surrogate identifier of all the dimension levels.

For a relational implementation, the surrogate identifier serves the following purposes:

- If a child level is stored in a different table from the parent level, each child level record stores the surrogate identifier of the parent record.
- In a fact table, each cube record stores only the surrogate identifier of the dimension record to which it refers. By storing the surrogate identifier, the size of the fact table that implements the cube is reduced.

Business Identifiers A business identifier consists of a user-selected list of attributes. The business identifier must be unique across the level and is always derived from the natural key of the data source. The business identifier uniquely identifies the member. For example, the business identifier of a Product level can be its Universal Product Code (UPC), which is a unique code for each product.

> **Note:** For a dimension that has a MOLAP implementation, the business identifier can consist of only one attribute.

The business identifier does the following:

- Identifies a record in business terms.
- Provides a logical link between the fact and the dimension or between two levels.
- Enables the lookup of a surrogate key.

When you populate a child level in a dimension, you must specify the business identifier of its parent level. When you populate a cube, you must specify the business identifier of the dimension level to which the cube refers.

Parent Identifier A parent identifier is used to annotate the parent reference in a value-based hierarchy. For more information on value-based hierarchies, see "Value-based Hierarchies" on page 4-34.

For example, an EMPLOYEE dimension with a value-based hierarchy, has the following dimension attributes: ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE, JOB_ID, HIRE_DATE, and MANAGER_ID. In this dimension, ID is the surrogate identifier and MANAGER_ID is the parent identifier.

Defining Level Attributes

A level attribute is a descriptive characteristic of a level member. Each level in the dimension has a set of level attributes. To define level attributes, you just select the dimension attributes that the level will implement. A level attribute has a distinct name and a data type. The data type is inherited from the dimension attribute that the level attribute implements. The name of the level attribute can be modified to be different from that of the dimension attribute that it implements.

Every level must implement the attribute marked as the surrogate identifier and the business identifier in the set of the dimension attributes.

Defining Hierarchies

A dimension hierarchy is a logical structure that uses ordered levels or a set of data values (for a value-based hierarchy) as a means of organizing data. A hierarchy

describes parent-child relationships among a set of levels. A level-based hierarchy must have at least one level. A level can be part of more than one hierarchy.

For example, the Time dimension can have the following two hierarchies:

Fiscal Hierarchy: Fiscal Year > Fiscal Quarter > Fiscal Month > Fiscal Week > Day

Calendar Hierarchy: Calendar Year > Calendar Quarter > Calendar Month > Day

All hierarchies must be strict 1:n relationships. One record in a parent level corresponds to multiple records in a child level. But one record in a child level corresponds to only one parent record within a hierarchy.

Dimension Roles

A dimension role is an alias for a dimension. In a data warehouse, a cube can refer to the same dimension multiple times, without requiring the dimension to be stored multiple times. Multiple references to the same dimension may cause confusion. So you create an alias for each reference to the dimension, thus allowing the joins to be instantly understandable. In such cases, the same dimension performs different dimension roles in the cube.

For example, a sales record can have the following three time values:

- Time the order is booked
- Time the order is shipped
- Time the order is fulfilled

Instead of creating three time dimensions and populating them with data, you can use dimension roles. Model one time dimension and create the following three roles for the time dimension: order booked time, order shipped time, and order fulfilment time. The sales cube can refer to order time, ship time and payment time dimension.

When the dimension is stored in the database, only one dimension is created and each dimension role references this dimension. But when the dimension is stored in the OLAP catalog, Warehouse Builder creates a dimension for each dimension role. Thus, if a time dimension has three roles, three dimensions are created in the OLAP catalog. However, all three dimensions are mapped to the same underlying table. This is a workaround because the OLAP catalog does not support dimension roles.

Note: Dimension roles can be created for dimensions that have a relational implementation only.

Level Relationships

A level relationship is an association between levels in a dimension hierarchy. Level relationships are implemented using level attributes that store the reference to the parent level in the hierarchy.

For example, the Products dimension has the following hierarchy: Total > Groups > Product. Warehouse Builder creates two level relationships: Product to Groups and Groups to Total. Two new attributes implement this level relationship: one in the Product level and one in the Groups level. These attributes store the surrogate ID of the parent level.

Dimension Example

An example of a dimension is the Products dimension that you use to organize product data. Table 4–3 lists the levels in the Products dimension and the surrogate identifier and business identifier for each of the levels in the dimension.

Table 4–3 Products Dimension Level Details

Level	Attribute Name	Identifier
Total	ID	Surrogate
	Name	Business
	Description	
Groups	ID	Surrogate
	Name	Business
	Description	
Product	ID	Surrogate
	UPC	Business
	Name	
	Description	
	Package Type	
	Package Size	

The Products dimension contains the following hierarchy:

Hierarchy 1: Total > Groups > Product

Value-based Hierarchies

A value-based hierarchy is a dimension in which hierarchical relationships are defined by a parent dimension attribute and a child dimension attribute. This is different from a level-based hierarchy, referred to as a hierarchy in this chapter, in which the hierarchical relationships are defined between levels.

You create a value-based hierarchy when the parent-child relationships cannot be grouped into meaningful levels. A value-based hierarchy has no levels. When you create the dimension attributes, you must specify which dimension attribute is the parent attribute.

For example, consider an EMPLOYEE dimension that has the following dimension attributes: ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE, JOB_ID, HIRE_DATE, DESCRIPTION, and MANAGER_ID. This dimension contains a parent-child relationship in which the MANAGER_ID attribute identifies the manager of each employee. But these relationship may not form meaningful levels across the organization. This is because the number of levels between an employee and the CEO is not the same for all employees. There may be 4 levels between employee A and the CEO, whereas, there may be 6 levels between employee B and the CEO. In such cases, you create a value-based hierarchy with MANAGER_ID as the parent identifier.

You can create value-based hierarchies using the Data Object Editor only. For more information about specifying a parent attribute, see "Attributes Tab" on page 13-40.

Note: Value-based hierarchies can be created only in dimensions that use a MOLAP implementation.

Implementing a Dimension

Implementing a dimension consists of specifying how the dimension and its data are physically stored. You can choose either a relational implementation, ROLAP implementation, or MOLAP implementation for a dimension. For more information about setting the implementation method, see "Implementing Dimensional Objects" on page 4-26.

> **Note:** For information about certain limitations of deploying dimensions to the OLAP catalog, see the *Oracle Warehouse Builder Release Notes*.

Relational and ROLAP Implementation of a Dimension

When you store dimension data in a relational form, you can implement the dimension using one of the following two methods:

- Star Schema
- Snowflake Schema

Star Schema In a star schema implementation, Warehouse Builder stores the dimension data in a single table. Because the same table or view stores data for more than one dimension level, you must specify a dimension key column in the table. The dimension key column is the primary key for the dimension. This column also forms the foreign key reference to the cube.

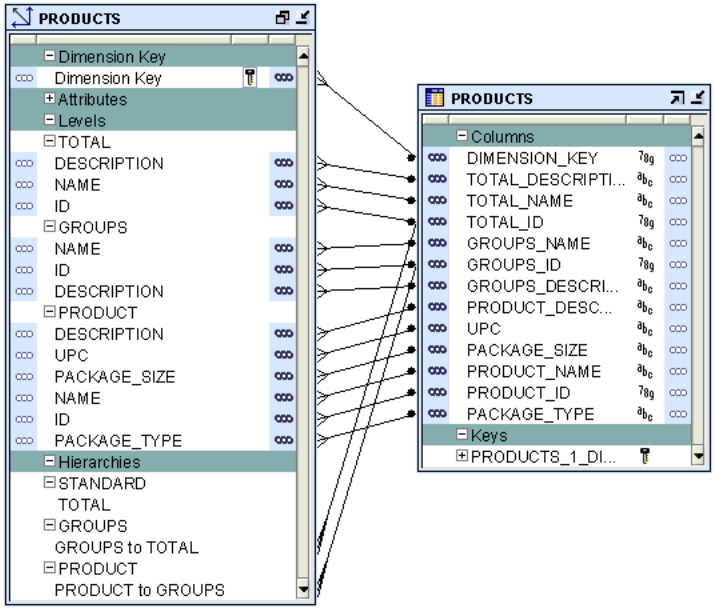
Each level implements a subset of dimension attributes. By default, the level attribute name is the same as the dimension attribute name. To avoid name conflicts caused by all level data being stored in the same table, Warehouse Builder uses the following guidelines for naming in a star table:

- If the level attribute name is not unique, Warehouse Builder prefixes it with the name of the level.
- If the level attribute name is unique, Warehouse Builder does not use any prefix.

> **Note:** To ensure that no prefixes are used, you must explicitly change the level attribute name in the Create Dimension wizard or the Data Object Editor.

For example, if you implement the Products dimension using a star schema, Warehouse Builder uses a single table to implement all the levels in the dimension. Figure 4–9 displays the star schema implementation of the Products dimension. The attributes in all the levels are mapped to different columns in a single table called PRODUCTS. The column called DIMENSION_KEY stores the surrogate ID for the dimension and is the primary key of the table.

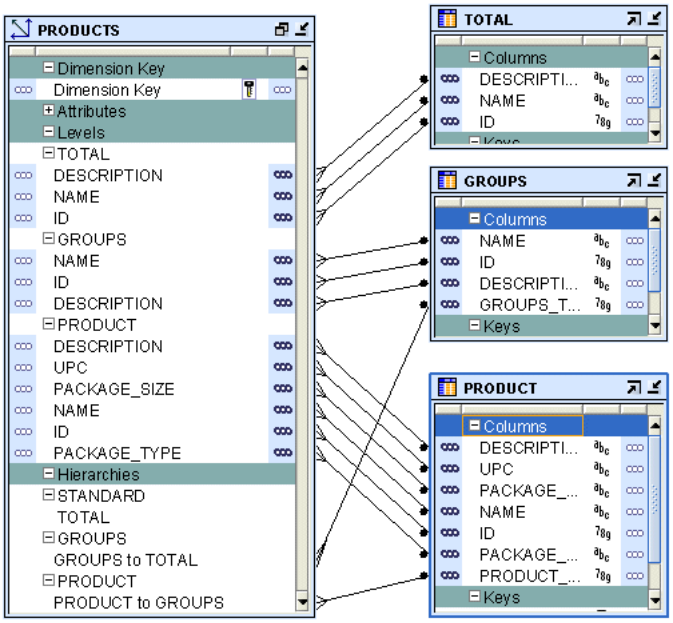
Figure 4–9 Star Schema Implementation of Products Dimension



Snowflake Schema In a snowflake schema implementation, Warehouse Builder uses more than one table to store the dimension data. Separate database tables or views store the data pertaining to each level in the dimension.

Figure 4–10 displays the snowflake implementation of the Products dimension. Each level in the dimension is mapped to a different table.

Figure 4–10 Snowflake Schema Implementation of the Products Dimension



Binding

When you perform binding, you specify the database columns that will store the data of each attribute and level relationship in the dimension. You can perform either auto

binding or manual binding for a dimension. For more information about binding, see "Binding" on page 4-36.

Auto Binding When you perform auto binding, Warehouse Builder binds the dimension object attributes to the database columns that store their data. When you perform auto binding for the first time, Warehouse Builder also creates the tables that are used to store the dimension data.

When you perform auto binding on a dimension that is already bound, Warehouse Builder uses the following rules:

- If the implementation method of the dimension remains the same, Warehouse Builder rebinds the dimensional object to the existing implementation objects. The implementation method can be either Star or Snowflake. For more information on implementation methods, see "Relational and ROLAP Implementation of a Dimension" on page 4-35.

  For example, you create a Products dimension using the star schema implementation method and perform auto binding. The dimension data is stored in a table called Products. You modify the dimension definition at a later date but retain the implementation method as star. When you now auto bind the Products dimension, Warehouse Builder rebinds the Products dimension attributes to the same implementation tables.

- If the implementation method of a dimension is changed, Warehouse Builder deletes the old implementation objects and creates a new set of implementation tables. If you want to retain the old implementation objects, you must first unbind the dimensional object and then perform auto binding. For more information on implementation methods, see "Relational and ROLAP Implementation of a Dimension" on page 4-35.

  For example, you create a Products dimension using the star schema implementation method and bind it to the implementation table. You now edit this dimension and change its implementation method to snowflake. When you now perform auto binding for the modified Products dimension, Warehouse Builder deletes the table that stores the dimension data, creates new implementation tables, and binds the dimension attributes and relationships to the new implementation tables.

To perform auto binding:

1. In the Project Explorer, right-click the dimension and select **Open Editor**.

   The Data Object Editor for this dimension is displayed.

2. On the Dimensional tab, right-click the Dimension node and select **Bind**.

   Alternatively, select the dimension node on the canvas and from the **Object** menu choose **Bind**.

Auto binding uses the implementation settings described in "Relational and ROLAP Implementation of a Dimension" on page 4-35.

Manual Binding You would typically use manual binding to bind existing tables to a dimension. Use manual binding if no auto binding or rebinding is required.

To perform manual binding for a dimension:

1. Create the implementation objects (tables or views) that you will use to store the dimension data.

In the case of relational or ROLAP dimensions, create the sequence used to load the surrogate identifier of the dimension. You can also choose to use an existing sequence.

2. In the Project Explorer, right-click the dimension and select **Open Editor**.

   The Data Object Editor for the dimension opens. On the canvas, the Dimensional tab is active.

3. Right-click the dimension and select **Detail View**.

   Warehouse Builder opens a new tab that has the same name as the dimension.

4. From the Palette, drag and drop the operator that represents the implementation object onto the canvas.

   Warehouse Builder displays the Add a New or Existing <Object> dialog. For example, if the dimension data is stored in a table, drag a Table operator from the Palette and drop it onto the canvas. The Add a New or Existing Table dialog is displayed.

5. Choose the **Select an existing <Object>** option and then select the data object from the list of objects displayed in the selection tree.

6. Click **OK**.

   A node representing the object that you just added is displayed on the canvas.

7. If more than one data object is used to store the dimension data, perform steps 4 to 6 for each data object.

8. Map the attributes in each level of the dimension to the columns that store their data. To do this, hold down your mouse on the dimension attribute, drag, and then drop on the column that stores the attribute value.

   Also map the level relationships to the database column that store their data.

   For example, for the Products dimension described in "Dimension Example" on page 4-34, the attribute Name in the Groups level of the Products dimension is stored in the Group_name attribute of the Products_tab table. Hold down the mouse on the Name attribute, drag, and drop on the Group_name attribute of the Products_tab table.

MOLAP Implementation

When a dimension is implemented in a MOLAP environment, the dimension definition and data are stored in an analytic workspace. This is done using analytic workspace objects such as dimensions, relationships, and so on. You can store multiple cubes in the same analytic workspace. For more information on MOLAP implementation, see "MOLAP Implementation of Dimensional Objects" on page 4-28.

About Slowly Changing Dimensions

A Slowly Changing Dimension (SCD) is a dimension that stores and manages both current and historical data over time in a data warehouse. In data warehousing, there are three commonly recognized types of SCDs.

With the appropriate licensing, you can use Warehouse Builder to define, deploy, and load all three types of SCDs. You can create slowly changing dimensions only for dimensions that use a relational implementation.

Note: Type 1 does not require additional licensing; however, type 2 and 3 SCDs require the Warehouse Builder Enterprise ETL Option.

Table 4–4 describes the three types of SCDs.

Table 4–4 Types of Slowly Changing Dimensions

Type	Use	Description	Preserves History?
Type 1	Overwriting	Only one version of the dimension record exists. When a change is made, the record is overwritten and no historic data is stored.	No
Type 2	Creating a new version of a dimension record	There are multiple versions of the same dimension record, and new versions are created while the old ones are still kept upon modification.	Yes
Type 3	Creating a current value field	There is one version of the dimension record. This record stores the previous value and current value of selected attributes.	Yes

To create a Type 2 SCD or a Type 3 SCD, in addition to the regular dimension attributes, you need additional attributes that perform the following roles:

- **Triggering Attributes:** These are attribute for which historical values need to be stored. For example, in the Products dimension, the attribute package_type of the Product level can be a triggering attribute. This means that when the value of this attribute changes, the old value needs to be stored.
- **Effective Date:** This attribute stores the start date of the record's life span.
- **Expiration Date:** This attribute stores the end date of the record's life span.

An attribute can play only one of the above roles. For example, an attribute cannot be a regular attribute and an effective date attribute. When you use the wizard to create a Type 2 SCD or a Type 3 SCD, Warehouse Builder creates the required additional attributes.

About Type 1 SCDs

In a Type 1 SCD the new data overwrites the existing data. Typically, this type is not considered an SCD and most dimensions are of this type. Thus the existing data is lost as it is not stored anywhere else. This is the default type of dimension you create. You do not need to specify any additional information to create a Type 1 SCD. Unless there are specific business reasons, you must assume that a Type 1 SCD is sufficient. For more information on how to define and implement a Type 1 SCD, refer to the following:

- Defining a Dimension on page 4-31
- Implementing a Dimension on page 4-35

About Type 2 SCDs

A Type 2 SCD retains the full history of values. When the value of a triggering attribute changes, the current record is closed. A new record is created with the changed data values and this new record becomes the current record. Each record contains the effective date and expiration date to identify the time period for which the record was active. Warehouse Builder also enables you to set a specific non-null date value as the expiration date. The current record is the one with a null or the previously specified value in the expiration date.

All the levels in a dimension need not store historical data. Typically, only the lowest levels is versioned.

Note: Be aware of the impact that all levels in a dimension not storing historical data has on query tools.

Defining a Type 2 SCD

To define a Type 2 SCD:

- For the level that stores historical data, specify the attributes used as the effective date and the expiration date.
- Choose the level attribute(s) that will trigger a version of history to be created.

  You cannot choose the surrogate ID, effective date attribute or expiration date attribute as the triggering attribute.

Each version of a record is assigned a different surrogate identifier. The business ID connects the different versions together in a logical sense. Typically, if there is a business need, Type 2 SCDs are used.

Example

To create the Products dimension described in "Dimension Example" on page 4-34 as a Type 2 SCD:

- Specify that the PACKAGE_TYPE and the PACKAGE_SIZE attributes of the Product level are the triggering attributes.
- Use two additional attributes in the Product level, to store the effective date and the expiration date of the level records. When you use the Create Dimension wizard to create a Type 2 SCD, Warehouse Builder creates these additional attributes for the lowest level only. If you use the Data Object Editor to create a Type 2 SCD, you must explicitly create these attributes and apply them to the level.

When the value of any of the triggering attributes changes, Warehouse Builder performs the following:

- Marks the original dimension record as expired. The expiration date of this record is set to the current time or any value that you specify. For more information on setting the expiration date, see "Dimension Operator" on page 25-14.
- Creates a new record using the changed attribute values. The expiration date of this new record can be set to NULL or a predefined value. For more information on setting the value for the expiration date attribute, see "Dimension Operator" on page 25-14.

About Type 3 SCDs

A Type 3 SCD stores two versions of values for certain selected level attributes. Each record stores the previous value and the current value of the versioned attributes. When the value of any of the versioned attributes changes, the current value is stored as the old value and the new value becomes the current value. Each record stores the effective date that identifies the date from which the current value is active. This doubles the number of columns for the versioned attributes and is used rarely.

Defining a Type 3 SCD

To define a Type 3 SCD:

1. For each level, specify which attributes should be versioned. That is, which attributes should store the previous value as well as the current value.
2. For each versioned attribute, specify the attribute that stores the previous value.

   The following restrictions apply to attributes that can have a previous value.

   - An attribute specified as a previous value cannot have further previous values.
   - The surrogate ID cannot have previous values.
3. For each level that is versioned, specify the attribute that stores the effective date.

Warehouse Builder recommends that you do not include previous value attributes in the business identifier of a Type 3 SCD.

Example

The Products dimension described in "Dimension Example" on page 4-34 can be created as a Type 3 SCD. The attributes PACKAGE_TYPE and PACKAGE_SIZE of the Product level should be versioned. You define two additional attributes to store the previous values, say PREV_PACK_SIZE and PREV_PACK_TYPE in the Product level. Suppose the value of the PACKAGE_TYPE attribute changes, Warehouse Builder stores the current value of this attribute in PREV_PACK_TYPE and stores the new value in the PACKAGE_TYPE attribute. The effective date attribute can be set to the current system date or to any other specified date.

About Cubes

Cubes contain measures and link to one or more dimensions. The axes of a cube contain dimension members and the body of the cube contains measure values. Most measures are additive. For example, sales data can be organized into a cube whose edges contain values for Time, Products, and Customers dimensions and whose body contains values from the measures Value sales, and Dollar sales.

A cube is linked to dimension tables over foreign key constraints. Since data integrity is vital, these constraints are critical in a data warehousing environment. The constraints enforce referential integrity during the daily operations of the data warehouse.

Data analysis applications typically aggregate data across many dimensions. This enables them to look for anomalies or unusual patterns in the data. Using cubes is the most efficient way of performing these type of operations. In a relational implementation, when you design dimensions with warehouse keys, the cube row length is usually reduced. This is because warehouse keys are shorter than their natural counterparts. This results is lesser amount of storage space needed for the cube data. For a MOLAP implementation, OLAP uses VARCHAR2 keys.

A typical cube contains:

- A primary key defined on a set of foreign key reference columns or, in the case of a data list, on an artificial key or a set of warehouse key columns. When the cube is a data list, the foreign key reference columns do not uniquely identify each row in the cube.
- A set of foreign key reference columns that link the table with its dimensions.

Defining a Cube

A cube consists of the set of measures defined over a set of dimensions. To create a cube, you must define the following:

- Cube Measures
- Cube Dimensionality

Cube Measures

A measure is data, usually numeric and additive, that can be examined and analyzed. Examples of measures include sales, cost, and profit. A cube must have one or more measures. You can also perform aggregation of measures. Only numeric measures can be aggregated.

Cube Dimensionality

A cube is defined by a set of dimensions. A cube can refer to a level that is not the lowest level in a dimension.

For cubes that use a pure relational implementation, you can reuse the same dimension multiple times with the help of dimension roles. For more information on dimension roles, see "Dimension Roles" on page 4-33.

Before you validate a cube, ensure that all the dimensions that the cube references are valid.

To define a dimension reference, specify the following:

- The dimension and the level within the dimension to which the cube refers.

  For a cube that uses a relational implementation, you can refer to intermediate levels in a dimension. However, for cubes that use a multidimensional implementation, you can only reference the lowest level in the dimension. Warehouse Builder supports a reference to the non surrogate identifier of a level, for example, the business keys.

- For dimensions that use a relational or ROLAP implementation, a dimension role for each dimension to indicate what role the dimension reference is performing in the cube. Specifying the dimension role is optional.

When you define a MOLAP cube, the order in which you define the dimension references is important. The physical ordering of dimensions on disk is the same as the order in which you define the dimension references. The physical ordering is tightly coupled with the sparsity definition. Define the dimension references in the order of most dense to least dense. Time is usually a dense dimension, and listing it first expedites data loading and time-based analysis. For more information on defining dimension references, see "Dimensions Page" on page 13-21 or "Dimensions Tab" on page 13-25. For more information on sparsity, see "Advanced Dialog" on page 13-25.

Default Aggregation Method

You can define aggregations that should be performed on the cube for ROLAP cubes or cube measures for MOLAP cubes. Warehouse Builder enables you to use the same aggregation function for all the cube measures or specify different aggregate functions for each measure.

Warehouse Builder supports the following default aggregation methods: SUM, SSUM (scaled SUM), AVERAGE, HAVERAGE (hierarchical average), MAX, MIN, FIRST, LAST, AND, OR, HIERARCHICAL_FIRST and HIERARCHICAL_LAST. If you do not

want to perform aggregation, select NOAGG. The methods AND and OR are not applicable for cubes that use a multidimensional implementation.

Note: You cannot define aggregation for pure relational cubes.

Cube Example

The Sales cube stores aggregated sales data. It contains the following two measures: Value_sales and Dollar_sales.

- Value_sales: Stores the amount of the sale in terms of the quantity sold.
- Dollar_sales: Stores the amount of the sale.

Table 4–5 describes the dimensionality of the Sales cube. It lists the name of the dimension and the dimension level that the cube references.

Table 4–5 Dimensionality of the Sales Cube

Dimension Name	Level Name
Products	Product
Customers	Customer
Times	Day

Implementing a Cube

When you implement a cube, you specify the physical storage details for the cube. You can implement a cube in a relational form or a multidimensional form in the database. Storing the cube data in an analytic workspace in the database is called as MOLAP implementation.

The types if implementation you can use for a cube are as follows:

- Relational implementation
- ROLAP implementation
- MOLAP implementation

To set the type of implementation for a cube, you use the **Deployment Option** configuration property. For more details on setting this option, see "Configuring Cubes" on page 13-35.

Relational and ROLAP Implementation of a Cube

The database object used to store the cube data is called a fact table. A cube must be implemented using only one fact table. The fact table contains columns for the cube measures and dimension references. For more information on setting the implementation option for a cube, see "Implementing Dimensional Objects" on page 4-26.

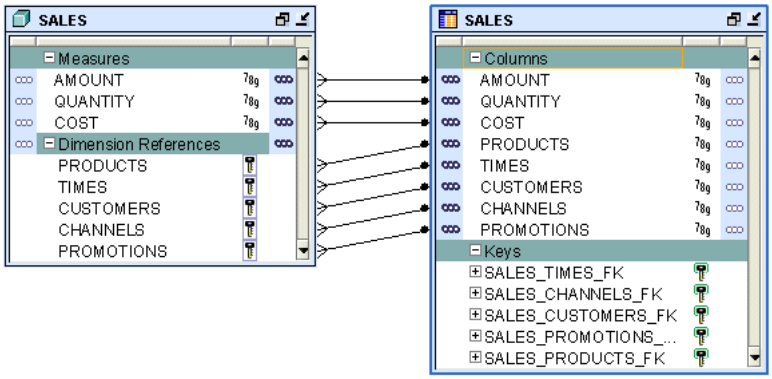
To implement a cube:

- Select a table or materialized view that will store the cube data.
- For each measure, select a column that will store the measure data.
- For each dimension reference, select a column that will store the dimension reference.

Each dimension reference corresponds to a column on the fact table and optionally a foreign key from the fact table to dimension table. The 1:n relationships from the fact tables to the dimension tables must be enforced.

Figure 4–11 displays the bindings for the relational implementation of the Sales cube. The data for the Sales cube is stored in a table called SALES.

Figure 4–11 Implementation of the Sales Cube



Binding

When you perform binding, you specify the database columns that will store the data of each measure and dimension reference of the cube. You can perform auto binding or manual binding for a cube. For more information on binding, see "Binding" on page 4-27.

Auto Binding When you perform auto binding, Warehouse Builder creates the table that stores the cube data and then binds the cube measures and references to the database columns. For detailed steps on performing auto binding, see "Auto Binding" on page 4-37.

When you perform auto binding for a cube, ensure that you auto bind the dimensions that a cube references before you auto bind the cube. You will not be able to deploy the cube if any dimension that the cube references has been auto bound after the cube was last auto bound.

For example, you create the SALES cube that references the TIMES and PRODUCTS dimensions and perform auto binding for the cube. You later modify the definition of the PRODUCTS dimension. If you now attempt to auto bind the SALES cube again, Warehouse Builder generates an error. You must first auto bind the PRODUCTS dimensions and then auto bind the cube.

Manual Binding In manual binding, you must first create the table or view that stores the cube data and then map the cube references and measures to the database columns that store their data. Alternatively, you can use an existing database table or view to store the cube data.

To perform manual binding for a cube:

1. Create the table or view that stores the cube data.
2. In the Project Explorer, right-click the cube and select **Open Editor**.

   The Data Object Editor for the cube opens. On the canvas, the Dimensional tab is active.
3. Right-click the cube and select **Detail View**.

Warehouse Builder opens a new tab that has the same name as the cube.

4. From the Palette, drag and drop the operator that represents the implementation object onto the canvas.

   Warehouse Builder displays the Add a New or Existing <Object> dialog. For example, if the cube data is stored in a table, drag a Table operator from the Palette and drop it onto the canvas. The Add a New or Existing Table dialog is displayed.

5. Choose the **Select an existing <object>** option and then select the data object from the list of objects displayed in the selection tree.

6. Click **OK**.

   A node representing the object that you just added is displayed on the canvas.

7. Map the measures and dimension references of the cube to the columns that store the cube data. To do this, hold down your mouse on the measure or dimension reference, drag, and then drop on the data object attribute that stores the measure or dimension reference.

MOLAP Implementation of a Cube

Storing the cube and its data in an analytic workspace is called a MOLAP implementation. You can store multiple cubes in the same analytic workspace. For more information on OLAP implementation, see "MOLAP Implementation of Dimensional Objects" on page 4-28.

Solve Dependency Order of Cube

Certain business scenarios may require the dimensions in a cube to be evaluated in a particular order. The order in which the dimensions are evaluated is called the solve dependency order of the cube. For example, in the Sales cube, the Time dimension may need to be evaluated before the Products dimension. For each dimension of the cube, you can specify a dependency on another dimension of the cube.

The advantage of specifying the dependency order is that it enables Warehouse Builder to optimize the query speed of calculating the joins of the dimension and cubes. For example, retrieving results from the sales cube based on Time criteria may be more selective than retrieving result based on Products criteria. In this case, you can specify that for the Sales cube, the Products dimension depends on the Time dimension.

Specifying the solve dependency order is optional. If you do not specify a dependency order, the optimizer determines the solve-order with additional flexibility.

About Time Dimensions

A time dimension is a dimension that stores temporal data. Time dimensions are used extensively in data warehouses. Warehouse Builder enables you to create and populate time dimensions. You can use Warehouse Builder to create both fiscal and calendar time dimensions.

When you create a time dimension using the wizard, Warehouse Builder creates the mapping for you to execute to populate the time dimension. Also, the data loaded into the time dimension conforms to the best practices recommended by Warehouse Builder for a time dimension.

This section contains the following topics:

Best Practices for Creating a Time Dimension

Warehouse Builder provides an accelerator to create time dimensions. It also specifies a set of rules as best practices for defining a time dimension. Warehouse Builder enforces these rules when you use Create Time Dimension wizard to create a time dimension.

The rules are as follows:

- The time dimension can contain only a subset of the predefined levels specified by Warehouse Builder.
- Each level in a time dimension must have attributes for the time span and ending date.
- A time dimension can have one or more hierarchies. Each hierarchy should be either a fiscal hierarchy or a calendar hierarchy.
- When you deploy a time dimension to the OLAP catalog, you must attach the time span and end date descriptors related to the levels to the dimension and its levels. When you create a time dimension using the Create Time Dimension wizard, Warehouse Builder performs this for you.

If you find these rules too restrictive for your business environment, you can create your own time dimension by setting the time attributes in the Data Object Editor. Ensure that you set the descriptors when you create a time dimension using the Data Object Editor.

Defining a Time Dimension

A time dimension consists of a set of levels and a set of hierarchies defined over these levels. Dimension roles are used extensively in time dimensions. For more information about dimension roles see "Dimension Roles" on page 4-33. To create a time dimension you must define the following:

- Levels
- Dimension Attributes
- Level Attributes
- Hierarchies

Levels

A level represents the level of aggregation of data. A time dimension must contain at least two levels. You can use a level only once in a time dimension. For example, a time dimension can contain only one Calendar Month level. Each level must have a surrogate identifier and a business identifier. The surrogate identifier should be the ID level attribute.

A Warehouse Builder time dimension can contain only a subset of the following levels:

- Day
- Fiscal week
- Calendar week

- Fiscal month
- Calendar month
- Fiscal quarter
- Calendar quarter
- Fiscal year
- Calendar year

Dimension Attributes

A dimension attribute is an attribute that is implemented by more than one level in the time dimension. Table 4–6 describes the dimension attributes of the Warehouse Builder time dimension.

Table 4–6 Dimension-level Attributes of the Time Dimension

Dimension Attribute	Description
ID	The ID attribute is implemented as level ID in all the levels.
Start Date	The start date for the period. It always starts at 00:00:00 of the first day of the period.
End Date	The end date for the period. It always ends on 23:59:59 of the last day of the period.
Time Span	Number of days in the period.
Description	Description of the level record.

Level Attributes

A level attribute is a descriptive characteristic of a level value. Warehouse Builder creates level attributes for the time dimension based on the levels that you decide to implement for the time dimension.

Table 4–7 lists the attributes of each level in the Warehouse Builder time dimension. For a description of each attribute, refer to Appendix B.

Table 4–7 Time Dimension Level Attributes

Level Name	Attribute Name
DAY	ID, DAY, START_DATE, END_DATE, TIME_SPAN, JULIAN_DATE, DAY_OF_CAL_WEEK, DAY_OF_ CAL_MONTH, DAY_OF_ CAL_ QUARTER, DAY_ OF_CAL_YEAR, DAY_OF_FISCAL_WEEK,DAY_ OF_FISCAL_MONTH, DAY_OF_FISCAL_ QUARTER, DAY_OF_FISCAL_YEAR. DESCRIPTION.
FISCAL WEEK	ID, WEEK_NUMBER, WEEK_OF_FISCAL_ MONTH, WEEK_OF_FISCAL_QUARTER, WEEK_ OF_FISCAL_YEAR, START_DATE, END_DATE, TIME_DATE, DESCRIPTION.
CALENDAR WEEK	ID, START_DATE, END_DATE, TIME_SPAN, DESCRIPTION.
FISCAL MONTH	ID, MONTH_NUMBER, MONTH_OF_QUARTER, MONTH_OF_YEAR, START_DATE, END_DATE, TIME_SPAN, DESCRIPTION.

Table 4–7 (Cont.) Time Dimension Level Attributes

Level Name	Attribute Name
CALENDAR MONTH	ID, MONTH_NUMBER, MONTH_OF_QUARTER, MONTH_OF_YEAR, START DATE, END_DATE, TIME_SPAN, DESCRIPTION.
FISCAL QUARTER	ID, QUARTER_NUMBER, QUARTER_OF_YEAR, START_DATE, END_DATE, TIME_SPAN, DESCRIPTION
CALENDAR QUARTER	ID, QUARTER_NUMBER, QUARTER_OF_YEAR, START_DATE, END_DATE, TIME_SPAN, DESCRIPTION.
FISCAL YEAR	ID, YESR_NUMBER, START_DATE, END_DATE, TIME_SPAN, DESCRIPTION.
CALENDAR YEAR	ID, YEAR_NUMBER, START_DATE, END_DATE, TIME_SPAN, DESCRIPTION

Hierarchies

A hierarchy is a structure that uses ordered levels to organize data. It defines hierarchical relationships between adjacent levels in a time dimension. A time dimension can have one or more hierarchies. Each hierarchy must be either a fiscal hierarchy or a calendar hierarchy. A single time dimension cannot contain both fiscal and calendar hierarchies.

Calendar Hierarchy A calendar hierarchy must contain at least two of the following levels: DAY, CALENDAR_WEEK, CALENDAR_MONTH, CALENDAR_QUARTER, CALENDAR_YEAR.

There is no drill-up path from CALENDAR_WEEK to any other levels. Thus, if a calendar hierarchy contains CALENDAR_WEEK level, it cannot contain either the CALENDAR_MONTH, CALENDAR_QUARTER, or CALENDAR_YEAR levels.

Fiscal Hierarchy A fiscal hierarchy should contain at least two of the following levels: DAY, FISCAL_WEEK, FISCAL_MONTH, FISCAL_QUARTER, FISCAL_YEAR.

When you create a fiscal hierarchy, you must specify the following:

- Start month of the fiscal year
- Start date of the fiscal year
- Start day for the fiscal week
- Fiscal Convention used by the time dimension.

  The options that you can select for fiscal convention are:

  - **455:** Select this option if the first month in the quarter has 4 weeks, the second month in the quarter has 5 weeks, and the third month in the quarter has 5 weeks.
  - **544:** Select this option if the first month in the quarter has 5 weeks, the second month in the quarter has 4 weeks, and the third month in the quarter has 4 weeks.

Implementing a Time Dimension

When you implement a time dimension, you specify how the time dimension and its data are physically stored. You can store the time dimension data either in a relational form or multidimensional form in the database.

The implementation of a time dimension is similar to the implementation of a regular dimension. For more information on implementing a dimension, see "Implementing a Dimension" on page 4-35.

Using a Time Dimension in a Cube Mapping

A time dimension created using the Time Dimension wizard uses the attribute ID as the surrogate identifier and the attribute CODE as the business identifier. The data type of both these attributes is NUMBER. When you create a cube that references a time dimension, the cube contains attributes that pertain to the surrogate identifier and the business identifier of the lowest level of the time dimension. Both these attributes have a data type of NUMBER.

When loading a cube, if you use a Warehouse Builder created time dimension as the source, both the source attributes and the cube attributes are of data type NUMBER. For example, consider a cube ALL_SALES that references two dimensions PRODUCTS and TIME_FISCAL. TIME_FISCAL is a calendar time dimension created using the Time Dimension wizard and it contains the levels Year, Month, and Day. When you create a map to load the ALL_SALES cube, you can directly map the attribute DAY_CODE of the Day level of TIME_FISCAL to the attribute ALL_SALES_DAY_CODE in the cube ALL_SALES. The data type of both these attributes is NUMBER.

Consider a scenario where you load data into the ALL_SALES cube from a source object in which the time data is stored as a DATE attribute. In this case, you cannot directly map the DATE attribute from the source to the attribute ALL_SALES_DAY_CODE of the ALL_SALES cube. Instead, you use an Expression operator in the mapping to convert the input DATE attribute to a NUMBER value and then load it into the ALL_SALES cube. In the Expression operator you convert the input using the following expression:

```
TO_NUMBER(TO_CHAR(input,'YYYYMMDD'))
```

where `input` represents the DATE attribute from the source object that needs to be converted to a NUMBER value. For information on using the Expression operator, see "Expression Operator" on page 26-11.

Populating a Time Dimension

You populate a time dimension by creating a mapping that loads data into the time dimension. When you create a time dimension using the Create Time Dimension wizard, Warehouse Builder creates a mapping that populates the time dimension. The time dimension is populated based on the values of the following parameters:

- Start year of the data
- Number of years of the data
- Start day and month of fiscal year (only for fiscal time dimensions)
- Start day of fiscal week (only for fiscal time dimensions)
- Fiscal type (only for fiscal time dimensions)

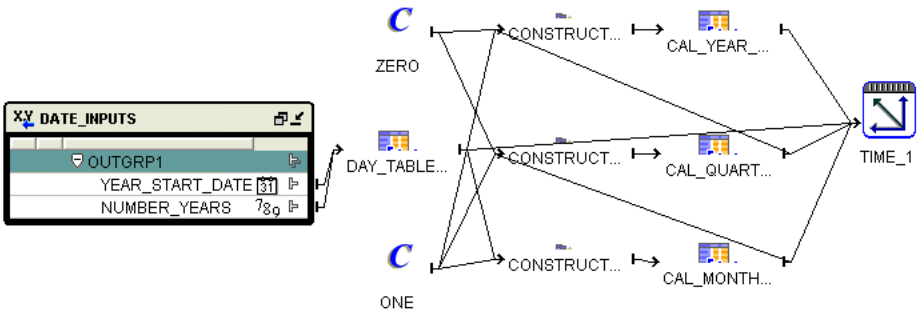
The values of these attributes are initialized at the time of creating the time dimension using the Create Time Dimension wizard. You can alter the values of these parameters

using the Data Object Editor. To change the values of the start date of the calendar year and the number of calendar years, use the Name Tab of the Data Object Editor. To change the values of the parameters pertaining to fiscal time dimensions, use the Fiscal Settings button on the Hierarchies Tab of Data Object Editor.

> **Note:** When you alter the values of any of the parameters pertaining to the data to be loaded into the time dimension, you must re-create the map that loads the time dimension. For more information on re-creating the map, see Hierarchies Tab on page 13-12.

Figure 4–12 displays a mapping to load a calendar time dimension. The Mapping Input operator DATE_INPUTS represents the attributes needed to populate the time dimension.

Figure 4–12 Mapping that Populates a Time Dimension



Overlapping Data Populations

You can run a map that populates the time dimension multiple times. During each run you specify the attributes required to populate the time dimension. It is possible that a run of the mapping may overlap with the previous runs, meaning you may attempt to load data that already exists in the time dimension. In such a case, if a record was populated by a previous run, Warehouse Builder does not populate the data again.

For example, in the first run, you populate the time dimension with data from the year 2000 for 5 years. In the second run, you populate the time dimension with data from 2003 for 3 years. Since the records from beginning 2003 to end 2004 already exist in the time dimension, they are not created again.

5

Identifying Data Sources and Importing Metadata

In Oracle Warehouse Builder you can access data from a variety of sources. As a precursor to extracting any data set, you first import its metadata.

This chapter includes the following topics:

About Source Data and Metadata

The source systems for a data warehouse are typically transaction processing applications. For example, a sales analysis data warehouse typically extracts data from an order entry system that records current order activities.

Designing the extraction process can be problematic. If the source system is complex and poorly documented, then determining which data to extract can be difficult. Moreover, the source system typically cannot be modified, nor can its performance or availability be adjusted. To address these problems, first import the metadata.

Metadata is the data that describes the contents of a given object in a data set. For example, the metadata for a table would indicate the data type for each column. After you import the metadata into Warehouse Builder, you can annotate the metadata and design an extraction strategy independently from the transaction processing application.

General Steps for Importing Metadata from Sources

1. Review the list of Supported Sources and Targets to determine if the source from which you want to extract data is supported in Warehouse Builder.
2. If you have not already done so, create a location and module for the source as described in "Creating an Oracle Data Warehouse" on page 2-1.
3. Right-click the module and select **Import.**
4. Follow the prompts in the Metadata Import Wizard.

   The wizard prompts you for information based on the type of source you selected.

Subsequent Steps

After successfully importing the metadata, you can design ETL logic to extract the data from the source, transform the data, and load it into a target schema.

Over the course of time, the source metadata may change. If this occurs, you can use Warehouse Builder to first identify the ETL logic that would be impacted and potentially made invalid due to a change in metadata as described in Chapter 31, "Managing Metadata Dependencies".

To introduce the changed metadata into Warehouse Builder, right-click the desired module and select **Import.** As described in "Re-Importing Definitions from an Oracle Database", Warehouse Builder recognizes when you are re-importing metadata.

Example: Importing Metadata from Flat Files

Whether you want to import metadata from a table, file, or application, the general process is the same and you always import metadata into a module.

In the Project Explorer, right-click the module and select **Import.** Follow the prompts in the Import Metadata Wizard.

Example: Importing Data from Flat Files

Assume that there are numerous flat files stored across three different drives and directories. In the Connection Explorer, you created 3 locations. Now in the Project Explorer, right-click the **Files** node and select **New** to create a new module. Repeat this for each of the three directories. For each of the three modules, select **Import.** A wizard directs you on how to import one or more files into each module.

Supported Sources and Targets

Table 5–1 lists the data storage systems and applications that Warehouse Builder 10.2 can access.The table lists the supported sources and targets in each **Location** node as displayed in the Connection Explorer.

Table 5–1 Sources and Targets Supported in Warehouse Builder 10.2

Location Node in the Connection Explorer	Supported Sources	Supported Targets
Databases/Oracle	Oracle db 8.1, 9.0, 9.2, 10.1, 10.2	Oracle db 9.2, 10.1, 10.2
Databases/Non-Oracle	Any database accessible through Oracle Heterogeneous Services, including but not limited to DB2, DRDA, Informix, SQL Server, Sybase, and Teradata. Any data store accessible through the ODBC Data Source Administrator, including but not limited to Excel and MS Access.	To load data into spreadsheets or third-party databases, first deploy to a comma-delimited or XML format flat file.
Files	Delimited and fixed-length flat files. See "Defining Flat Files and External Tables".	Comma-delimited and XML format flat files. See "Defining Flat Files and External Tables"

Table 5–1 (Cont.) Sources and Targets Supported in Warehouse Builder 10.2

Location Node in the Connection Explorer	Supported Sources	Supported Targets
Applications	SAP R/3 3.*x*, 4.*x* Oracle E-Business Suite PeopleSoft 8, 9 See "Importing Data From Third Party Applications"	none
Process Flows and Schedules/OEM	None	OEM Agent 9.0, 9.2
Process Flows and Schedules/Oracle Workflow	None	Oracle Workflow 2.6.2, 2.6.3, 2.6.4, 11*i*
Process Flows and Schedules/Concurrent Manager	None	Concurrent Manager 11*i*
Business Intelligence/BI Beans	None	BI Beans 10.1
Business Intelligence/Discoverer	None	Discoverer 10.1
Databases/Transportable Module Source	See "Moving Large Volumes of Data"	N/A
Databases/Transportable Module Target	N/A	See "Moving Large Volumes of Data"

Oracle Heterogeneous Services

Warehouse Builder communicates with non-Oracle systems using Oracle Database Heterogeneous Services and a complementary agent. Heterogeneous Services make a non-Oracle system appear as a remote Oracle Database server. The agent can be an Oracle Transparent Gateway or the generic connectivity agent included with Oracle Database.

- A transparent gateway agent is a system-specific source. For example, for a Sybase data source, the agent is a Sybase-specific transparent gateway. You must install and configure this agent to support the communication between the two systems.
- Generic connectivity is intended for low-end data integration solutions and the transfer of data is subject to the rules of specific ODBC or OLE DB drivers installed on the client machine. In this case, you do not need to purchase a separate transparent gateway; you can use the generic connectivity agent included with the Oracle Database server. You must still create and customize an initialization file for your generic connectivity agent.

For additional information on distributed processing systems, see *Oracle Database Distributed Database Systems.*

Integrating with Business Intelligence Tools

Warehouse Builder provides an end-to-end business intelligence solution by enabling you to integrate metadata from different data sources, designing and deploying it to a data warehouse, and making that information available to analytical tools for decision making and business reporting.

Warehouse Builder introduces Business Intelligence (BI) objects that enable you to integrate with Oracle Business Intelligence tools such as Discoverer and Business Intelligence (BI) Beans. You can define BI objects in Warehouse Builder that enable you to store definitions of business views and presentation templates. You can then deploy

these definitions to the Oracle Business Intelligence tools and extend the life-cycle of your data warehouse.

This section contains the following topics:

Introduction to Business Intelligence Objects in Warehouse Builder

Warehouse Builder enables you to derive and define Business Intelligence (BI) objects that integrate with analytical business intelligence tools, such as Oracle Discoverer and BI Beans. By deploying these BI definitions to your analytical tools, you can perform ad hoc queries on top of the relational data warehouse or define a dashboard on top of multidimensional data marts.

The BI objects you derive or define in Warehouse Builder represent equivalent objects in Oracle Discoverer and BI Beans. These definitions are stored under the Business Intelligence node on the Warehouse Builder Project Explorer.

The Business Intelligence node contains two additional nodes called Business Definitions and Business Presentations. You start by first creating a Business Definition module to store the definitions to be deployed to Discoverer. For details, see "About Business Definitions" on page 5-5. You can also create a Business Presentation module to store the presentation templates you want to deploy to BI Beans. For details, see "About Business Presentations" on page 5-5.

Introduction to Business Definitions

Business intelligence is the ability to analyze data to answer business questions and predict future trends. Oracle Discoverer is a BI tool that enables users to analyze data and retrieve information necessary to take business decisions. Discoverer also enables users to share the results of their data analysis in different formats (including charts and Excel spreadsheets).

Discoverer uses the End User Layer (EUL) metadata view to insulate its end users from the complexity and physical structure of the database. You can tailor the EUL to suit your analytical and business requirements and produce queries by generating SQL. The EUL provides a rich set of default settings to aid report building.

Through BI objects, Warehouse Builder enables you to design a data structure that facilitates this data analysis. Business Intelligence objects in Warehouse Builder provide the following benefits:

- Complete and seamless integration with Oracle Discoverer and BI Beans.
- Advanced deployment control of metadata objects using the Warehouse Builder Control Center.
- Complete, end-to-end lineage and impact analysis of Discoverer objects based on information in the Warehouse Builder repository.
- Ability to utilize Warehouse Builder metadata management features such as snap shots, multi language support, and command line interaction.

About Business Definitions

You can integrate with Discoverer by deriving business definitions directly from your warehouse design metadata. Alternatively, you can also create your own customized business definitions in Warehouse Builder.

The business definition objects in Warehouse Builder are equivalent to the Discoverer EUL objects. When you derive business definitions from your existing design metadata, Warehouse Builder organizes the definitions in Item Folders that correspond to Folders in Discoverer. You can define joins and conditions for the Items Folders and select the Items they contain using the Warehouse Builder wizards and editors. Additionally, you can define Drill Paths, Alternative Sort Orders, Drills to Detail, and Lists of Values for the Items within the Item Folders.

Warehouse Builder also enables you to define any functions registered with Discoverer. You can also sort your definitions by subject area by defining Business Areas that reference multiple Item Folders. You can then deploy these Business Areas along with the business definitions to a Discoverer EUL using the Control Center. For more information about deploying business definitions, see "Deploying Business Definitions" on page 15-23.

For information on creating, deriving, and deploying business definitions in Warehouse Builder, see Chapter 15, "Defining Business Intelligence Objects".

About Business Presentations

You can integrate with Oracle BI Beans by building and deploying the structures for Business Presentations or BI Reports directly from your warehouse design metadata.

Presentation Templates are stored within Business Presentation modules created under the Business Intelligence node on the Warehouse Builder Project Explorer. In Warehouse Builder, you can define first cut cross-tab and graphical Presentation Templates that can be deployed multiple times.

For information about creating business presentation, see Chapter 15, "Defining Business Intelligence Objects".

6

Creating Mappings

After you create and import data object definitions in Warehouse Builder, you can design extraction, transformation, and loading (ETL) operations that move data from sources to targets. In Warehouse Builder, you design these operations in a mapping.

This chapter contains the following topics that describe how to create, edit, and use mappings:

About Mappings and Operators

Mappings describe a series of operations that extract data from sources, transform it, and load it into targets. They provide a visual representation of the flow of the data and the operations performed on the data. When you design a mapping in Warehouse Builder, you use the Mapping Editor interface.

Alternatively, you can create and define mappings using OMB Plus, the scripting interface for Warehouse Builder as described in the Oracle Warehouse Builder API and Scripting Reference.

Based on the ETL logic that you define in a mapping, Warehouse Builder generates the code required to implement your design. Warehouse Builder can generate code for the following languages:

- **PL/SQL:** PL/SQL stands for Procedural Language/Standard Query Language. It extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL.

- **SQL*Loader**: SQL*Loader is an Oracle tool for loading data from files into Oracle Database tables. It is the most efficient way to load large amounts of data from flat files.
- **ABAP**: ABAP is a programming language for developing applications for the SAP R/3 system, a business application subsystem.

The basic design element for a mapping is the operator. Use operators to represent sources and targets in the data flow. Also use operators to define how to transform the data from source to target. The operators you select as sources have an impact on how you design the mapping. Based on the operators you select, Warehouse Builder assigns the mapping to one of the following Mapping Generation Languages:

- PL/SQL
- SQL*Loader
- ABAP

Each of these code languages require you to adhere to certain rules when designing a mapping.

- **PL/SQL Mappings**: For all mappings that do not contain either a flat file operator as a source or a SAP/R3 source, Warehouse Builder generates PL/SQL code. Design considerations for PL/SQL mappings depend upon whether you specify a row-based or set-based operating mode as described in Chapter 8, "Understanding Performance and Advanced ETL Concepts".
- **SQL*Loader Mappings:** When you define a flat file operator as a source, Warehouse Builder generates SQL*Loader code. To design a SQL*Loader mapping correctly, follow the guidelines described in "Flat File Source Operators" on page 25-30.
- **ABAP Mappings:** When you define a SAP/R3 source, Warehouse Builder generates ABAP code. For mapping design considerations for SAP sources, see "Creating SAP Extraction Mappings" on page 18-22.

Instructions for Defining Mappings

Before You Begin

First verify that your project contains a warehouse target module with a defined location.

Also, as described in "Example: Importing Metadata from Flat Files" on page 5-2, import any existing data you intend to use as sources or targets in the mapping.

To define a mapping, refer to the following sections:

1. Creating a Mapping on page 6-4
2. Adding Operators on page 6-11

   To design a mapping to extract from or load to a flat file, refer to "Instructions for Using Flat File Sources or Targets in a Mapping" on page 6-3.
3. Editing Operators on page 6-14
4. Connecting Operators on page 6-18
5. Using Pluggable Mappings on page 6-23
6. Setting Mapping Properties on page 6-27

7. Setting Operator, Group, and Attribute Properties on page 6-28
8. Configuring Mappings Reference on page 24-1
9. For PL/SQL mappings, you can also refer to "Best Practices for Designing PL/SQL Mappings" on page 8-1.
10. Debugging a Mapping on page 6-34
11. When you are satisfied with the mapping design, generate the code by selecting the Generate icon in the toolbar.

Subsequent Steps

After you design a mapping and generate its code, you can next create a process flow or proceed directly with deployment followed by execution.

Use process flows to interrelate mappings. For example, you can design a process flow such that the completion of one mapping triggers an email notification and launches another mapping. For more information, see Chapter 7, "Designing Process Flows".

Deploy the mapping, and any associated process flows you created, and then execute the mapping as described in Chapter 11, "Deploying to Target Schemas and Executing ETL Logic".

Instructions for Using Flat File Sources or Targets in a Mapping

In a mapping you can use flat file operators as either sources or targets but not a mix of both. You can import file definitions from existing flat files and use that data as a source or target in the mapping. Or you can create your own flat file definition in the Mapping Editor to load data into a new flat file target.

Creating a New Flat File Target

To create a new flat file definition for a target, complete the following steps:

1. If you have not already done so, create a flat file module as described in "Creating Flat File Modules" on page 14-3.

   A flat file module is necessary to enable you to create the physical flat file later in these instructions.
2. Create the mapping definition as described in "Creating a Mapping" on page 6-4.
3. Drag and drop a flat file operator onto the canvas.
4. On the Add Flat File Operator dialog, select the option Create Unbound Operator with No Attributes and assign a name to the new target operator.
5. Edit the new operator as described in "Editing Operators" on page 6-14.

   Thus far, you have defined an operator that represents a flat file but have not created the actual flat file target.
6. To create the flat file in the database, right-click the operator and select **Create and Bind...**

   The dialog prompts you to select a flat file module and enables you to assign a unique name to the flat file. When you click OK, Warehouse Builder displays the new target in the Project Explorer Files node under the module you specified.
7. Continue to define your mapping as described in "Instructions for Defining Mappings" on page 6-2.

Creating a Source or Target Based on an Existing Flat File

To use an existing flat file as a source or target, complete the following steps:

1. In the Project Explorer, right-click the File node and create a module for the flat files as described in "Creating Flat File Modules" on page 14-3.
2. Right-click the flat file module and select **Import** to import file definitions as described in "Using the Import Metadata Wizard for Flat Files" on page 14-10.
3. Decide to use the file as either a source or a target.

   If you import a file for use as a target, Warehouse Builder generates PL/SQL code for the mapping. Review the details in "Flat File Target Operators" on page 25-30 and then skip to step 7.

   If you import a file for use as a source, you must decide whether to maintain the flat structure of the file using SQL* Loader or to represent the data in PL/SQL format through an external table. Continue to the next step.
4. Refer to "External Table Operators versus Flat File Operators" to determine what type of operator to use in your mapping.

   If you select external table operator, continue to the next step.

   If you select flat file operator, skip to step 7.
5. Create the external table as described in "Creating a New External Table Definition" on page 14-26.
6. In the Project Explorer, right-click the external table and select **Configure.** On the DataFiles node, right-click and select **Create.**

   Enter the name of the flat file from which the external table inherits data. Enter the file name and the file extension such as *myflatfile.dat*.
7. Drag and drop the flat file operator or external table operator onto the canvas.
8. On the Add Operator dialog, select the option "Select from Existing Repository Object and Bind".

   You can now continue designing your mapping.

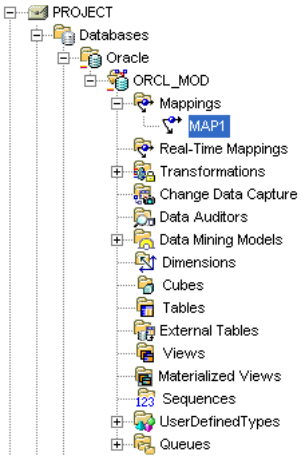
Creating a Mapping

To create a mapping:

1. Navigate to the **Mappings** node in the Project Explorer. This node is located under a warehouse target module, under the Databases folder, under the Oracle folder.

   Figure 6–1 shows the Mappings node containing maps *MAP1.* The warehouse target in this example is named *ORCL_MOD.*

Figure 6–1 Mappings Node on the Project Explorer



2. Right-click Mappings and then select **New.**

   Warehouse Builder opens the Create Mapping dialog.

3. Enter a name and an optional description for the new mapping.

   For rules on naming and describing mappings, see "Mapping Naming Conventions" on page 6-16.

4. Click **OK.**

   Warehouse Builder stores the definition for the mapping and inserts its name in the Project Explorer. Warehouse Builder opens a mapping editor for the mapping and displays the name of the mapping in the title bar.

To open a previously created mapping:

1. From the Project Explorer, locate a warehouse target module under the Databases folder and then under the Oracle Database folder as shown in Figure 6–1 on page 6-5.
2. Expand the Mappings node.
3. Open the Mapping Editor in one of the following ways:
   - Double-click a mapping.
   - Select a mapping and then from the **Edit** menu, select **Open Editor.**
   - Select a mapping and press **Ctrl + O**.
   - Right-click a mapping, and select **Open Editor.**

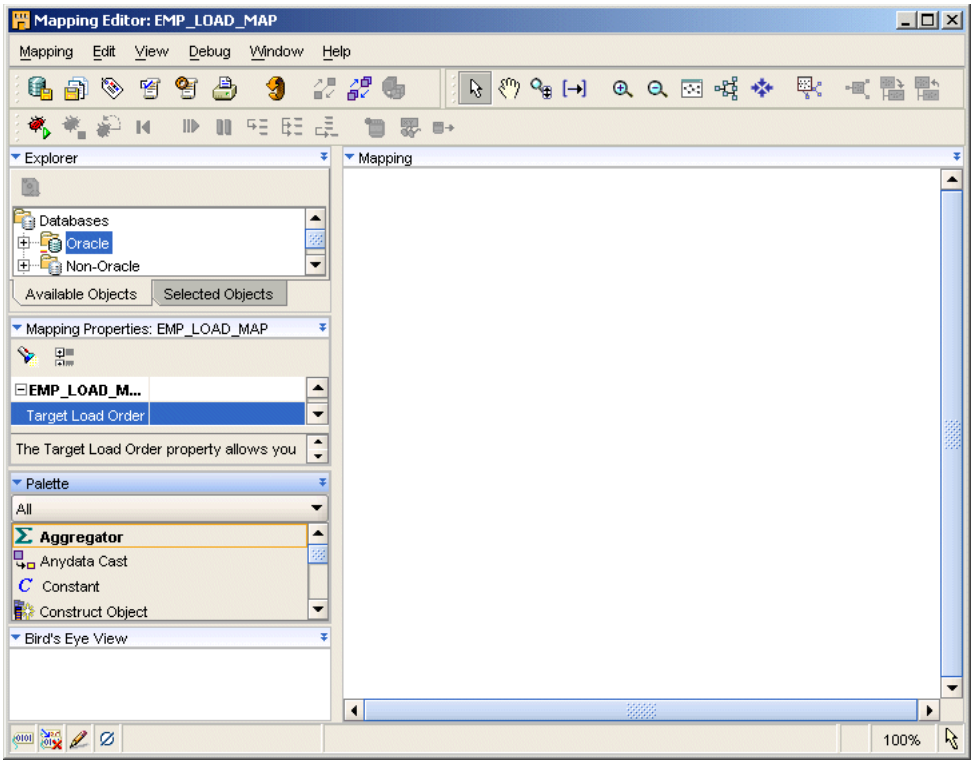
   Warehouse Builder displays the Mapping Editor.

About the Mapping Editor

The first time you open the Mapping Editor, it displays with a menu bar, multiple toolbars, multiple windows along the left side, and a canvas on the right.

Figure 6–2 displays the Mapping Editor canvas.

Figure 6–2 Mapping Editor Canvas



Standard Editor Components

The Mapping Editor has the following standard components common to most editors in Warehouse Builder:

- **Title Bar:** At the top of the editor, the title bar displays the name of the mapping and the access privileges you have on the mapping.
- **Menu Bar:** Below the title bar, the menu bar provides access to the editor commands. You can access the menu bar by clicking on one of its options or by using hot keys. For example, to access the Mapping menu, press **Alt +M.**
- **Toolbar:** Below the menu bar, the toolbar provides icons for commonly used commands.
- **Canvas:** The canvas provides the work space where you design and modify mappings.
- **Indicator Bar:** Along the lower edge of the editor you can see mode icons, indicators, and descriptions as shown in Figure 6–3.

Figure 6–3 Indicator Bar on the Mapping Editor

In the left corner are Naming Mode, Rename Mode, Read/Write, and Validation Mode.

In the right corner are the percent zoom indicator and the navigation mode. In the preceding figure, the zoom level is at 100% and the navigation mode is set to Select Mode.

Mapping Editor Windows

You can resize a window by placing your mouse on the border of the window, pressing the mouse button when the double sided arrow appears, and dragging your mouse to indicate the desired size.

You can move a window by placing the mouse on the Title Bar, and dragging the mouse to the desired location.

To show or hide windows, select **Window** from the menu bar and either activate or deactivate the check mark corresponding to the window.

Explorer

When you first launch the editor, Warehouse Builder displays the explorer in the upper left corner. The explorer provides a tree listing of all the activities on the canvas and their parameters. When you select an activity on the canvas, Warehouse Builder navigates to the activity on the explorer.

Properties Inspector

When you first launch the editor, Warehouse Builder displays the properties inspector in the lower left corner. The properties inspector displays the properties for the mapping, its operators, and attributes in the operators. Select an object either from the canvas or the explorer and Warehouse Builder displays the properties in the properties inspector.

Palette

When you first launch an editor, Warehouse Builder displays the palette along the left side and it contains activity icons that you can drag and drop onto the canvas.You can relocate the palette anywhere on the editor. You can choose to hide or display the palette by clicking on Operator Palette listed under **View** in the menu bar.

Bird Eye View

The Bird's Eye View enables you to move the view of the canvas with a single mouse dragging operation. You can thus reposition your view of the canvas without using the scroll bars.

The Bird's Eye View displays a miniature version of the entire canvas. It contains a blue colored box that represents the portion of the canvas that is currently in focus. In the case of mappings that span more than the canvas size, you can click the blue box and drag it to the portion of the canvas that you want to focus on.

Data Viewer

The Data Viewer enables you to view the data stored in the data object. Refer to the "Data Viewer" section on page 4-17 for more information.

Generation

The Generation panel displays the generation and validation results for a data object. This panel is hidden when you first open the editor window. It is displayed the first time you generate or validate a data object. You can to show or hide the Generation panel by selecting **Window** and then **Generation Results** from the editor menu.

The Generation window contains two tabs: Script and Message. The Script tab displays the scripts generated by Warehouse Builder to implement the data object

selected in the canvas. The Message tab displays the validation messages for the selected data object. Double-click a message to view the complete message text.

Mapping Editor Toolbars

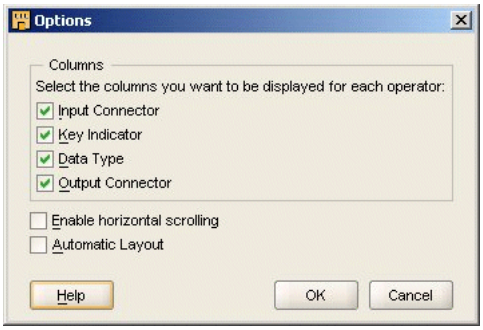
The Mapping Editor provides the following task oriented toolbars: general, graphic, generation, and palette. With the exception of the palette, the editor by default displays the toolbars below the menu bar. You can move, resize, or hide each of the toolbars.

- **General Toolbar:** Use this toolbar to invoke common operations such as save all, exporting diagram, validating, generating, and printing.
- **Diagram Toolbar:** Use this toolbar to navigate the canvas and change the magnification of objects on the canvas.
- **Debug Toolbar:** Use this toolbar to invoke commands for debugging the mapping.
- **Palette Toolbar:** The palette contains operator icons. To include an operator, drag an operator icon onto the Mapping Editor canvas. As Warehouse Builder includes over 50 operators, you may want to sort and display the operators based on type.

Mapping Editor Display Options

You can control how the editor displays the mappings on the canvas by selecting **View** from the menu bar and selecting **Options**. Figure 6–4 displays the Options dialog.

Figure 6–4 Default Viewing Options for the Mapping Editor



Types of Operators

As you design a mapping, you select operators from the Mapping Editor palette shown in Figure 6–5 and drag them onto the canvas.

This section introduces the types of operators and refers you to other chapters in this manual for detailed information.

- Oracle Source/Target Operators: These operators represent Oracle Database objects in the mapping. It also contains Flat File Source and Target operators.
- Remote and Non Oracle Source and Target Operators: The use of these operator have special requirements discussed in "Using Remote and non-Oracle Source and Target Operators" on page 25-27.
- Data Flow Operators: Data flow operators transform data.
- Pre/Post Processing Operators: Calls a function or procedure before or after executing a mapping

- Pluggable Mapping Operators: These are mappings that function as operators in other mappings.

Oracle Source/Target Operators

Use source and target operators to represent relational database objects and flat file objects. Table 6–1 lists each source and target operator alphabetically, gives a brief description.

Table 6–1 Source and Target Operators

Icon	Operator	Description
	Cube Operator	Represents a cube that you previously defined.
	Dimension Operator	Represents a dimension that you previously defined.
	External Table Operator	Represents an external table that you previously defined or imported.
	Sequence Operator	Generates sequential numbers that increment for each row.
	Constant Operator	Produces a single output group that can contain one or more constant attributes.
	Flat File Operator	Represents a flat file that you previously defined or imported.
	Construct Object Operator	Produces object types and collection types.
	Expand Object Operator	Expands an object type to obtain the individual attributes that comprise the object type.
	Varray Iterator	Iterates through the values in the table type.
	Materialized View Operator	Represents a materialized view that you previously defined.
	Table Operator	Represents a table that you previously defined or imported.
	View Operator	Represents a view that you previously defined or imported.
	Data Generator Operator	Provides information such as record number, system date, and sequence values.

Data Flow Operators

Use data flow operators to transform data in a mapping. Table 6–2 lists each data flow operator alphabetically, gives a brief description. For more information on these transformation operators, see Chapter 26, "Data Flow Operators".

Table 6–2 Data Flow Operators

Icon	Operator	Description
	Aggregator Operator	Performs data aggregations, such as SUM and AVG, and provides an output row set with aggregated data.
	Anydata Cast Operator	Converts an object of type Sys.AnyData to either a primary type or to a user defined type.

Table 6–2 (Cont.) Data Flow Operators

Icon	Operator	Description
	Deduplicator Operator	Removes duplicate data in a source by placing a DISTINCT clause in the select code represented by the mapping.
	Expressi9on Operator	Enables you to write SQL expressions that define non-procedural algorithms for one output parameter of the operator. The expression text can contain combinations of input parameter names, variable names, and library functions.
	Filter Operator	Conditionally filters out rows from a row set.
	Joiner Operator	Joins multiple row sets from different sources with different cardinalities and produces a single output row set.
	Key Lookup Operator	Performs a lookup of data from a lookup object such as a table, view, cube, or dimension.
	Match-Merge Operator	Data quality operator that identifies matching records and merges them into a single record.
	Name and Address Operator	Identifies and corrects errors and inconsistencies in name and address source data.
	Pivot Operator	Transforms a single row of attributes into multiple rows. Use this operator to transform data that contained across attributes instead of rows.
	Set Operation Operator	Performs union, union all, intersect, and minus operations in a mapping.
	Sorter Operator	Sorts attributes in ascending or descending order.
	Splitter Operator	Splits a single input row set into several output row sets using a boolean split condition.
	Table Function Operator	Enables you to develop custom code to manipulate a set of input rows and return a set of output rows of the same or different cardinality that can be queried like a physical table. You can use a table function operator as a target.
	Transformation Operator	Transforms the attribute value data of rows within a row set using a PL/SQL function or procedure.
	Unpivot Operator	Converts multiple input rows into one output row. It enables you to extract from a source once and produce one row from a set of source rows that are grouped by attributes in the source data.

Pre/Post Processing Operators

Use Pre/Post Processing operators to perform processing before or after executing a mapping. The Mapping parameter operator is used to provide values to and from a mapping. Table 6–3 lists the Pre/Post Process operators and the Mapping Parameters operator.

Table 6–3 Pre/Post Processing Operators

Icon	Operator	Description
	Pre-Mapping Process Operator	Calls a function or procedure prior to executing a mapping.

Table 6–3 (Cont.) Pre/Post Processing Operators

Icon	Operator	Description
	Post-Mapping Process Operator	Calls a function or procedure after executing a mapping.
	Mapping Input Parameter Operator	Passes parameter values into a mapping.
	Mapping Output Parameter Operator	Sends values out of a mapping.

Pluggable Mapping Operators

A pluggable mapping is a reusable grouping of mapping operators that behaves as a single operator. Table 6–4 lists the Pluggable Mappings operators.

Table 6–4 Pluggable Mapping Operators

Icon	Operator	Description
	Pluggable Mapping Operator	Represents a reusable mapping.
	Pluggable Mapping Input Signature Operator	A combination of input attributes that flow into the pluggable mapping.
	Pluggable Mapping Output Signature Operator	A combination of output attributes that flow out of the pluggable mapping.

Adding Operators

The steps you take to add an operator to a mapping depend on the type of operator you select. This is because some operators are bound to repository objects while others are not. As a general rule, when you add a data source or target operator, Warehouse Builder creates and maintains a version of that object in the Warehouse Builder repository and a separate version for the Mapping Editor. For example, when you add a table operator to a mapping, Warehouse Builder maintains a separate copy of the table in the repository. The separate versions are said to be *bound* together. That is, the version in the mapping is bound to the version in the repository.

To distinguish between the two versions, this chapter refers to objects in the repository either generically as *repository objects* or specifically as *repository tables*, *repository views*, and so on. And this chapter refers to operators in the mapping as *table operators*, *view operators*, and so on. Therefore, when you add a dimension to a mapping, refer to the dimension in the mapping as the *dimension operator* and refer to the dimension in the repository as the *repository dimension*.

Warehouse Builder maintains separate repository objects for some operators so that you can synchronize changing definitions of these objects. For example, when you re-import a new metadata definition for the repository table, you may want to propagate those changes to the table operator in the mapping. Conversely, as you make changes to a table operator in a mapping, you may want to propagate those changes back to its associated repository table. You can accomplish these tasks by a process known as synchronizing. In Warehouse Builder, you can synchronize automatically as described in Chapter 31, "Managing Metadata Dependencies". Alternatively, synchronize manually from within the Mapping Editor as described in "Synchronizing Operators and Repository Objects" on page 6-29.

To add an operator to a mapping:

1. Open the Mapping Editor.
2. From the **Mapping** menu, select **Add** and select an operator. Alternatively, you can drag an operator icon from the toolbox and drop it onto the Mapping Editor canvas.

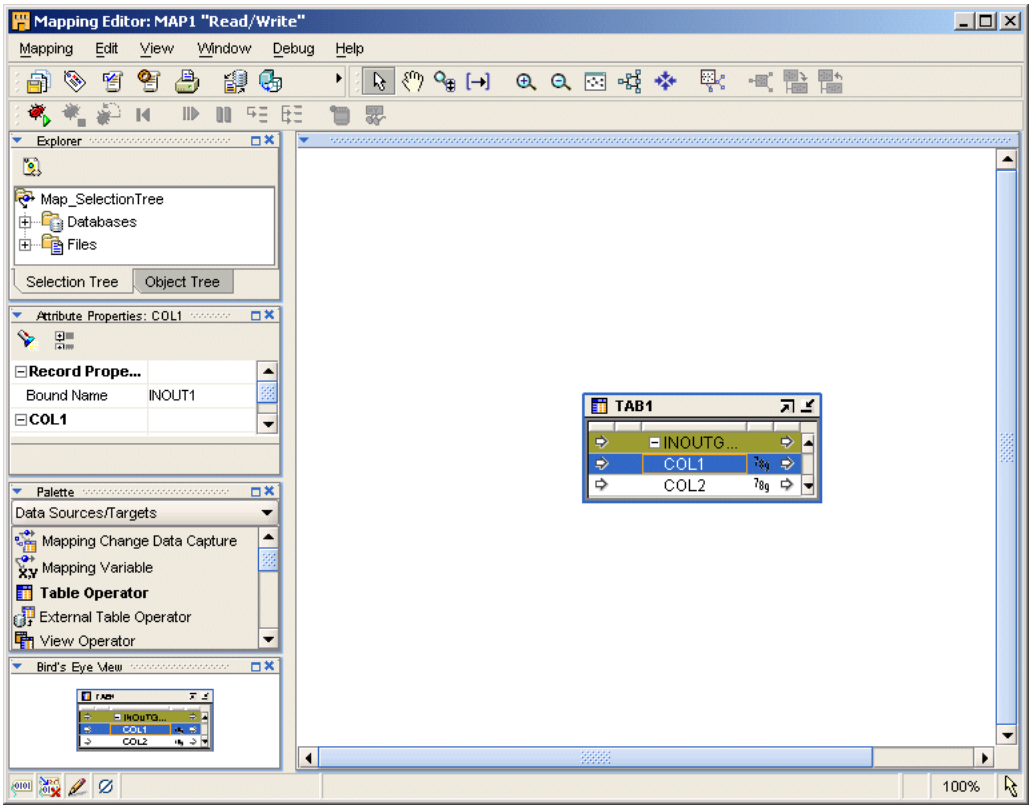
   If you select an operator that you can bind to a repository object, the Mapping Editor displays the **Add Mapping <operator name>** dialog. For details on how to use this dialog, see "Add Operator Dialog" on page 6-13.

   If you select an operator that you cannot bind to a repository object, Warehouse Builder may display a wizard or dialog to assist you in creating the operator.
3. Follow any prompts Warehouse Builder displays and click **OK.**

   The Mapping Editor displays the operator maximized on the canvas as shown in Figure 6–5. The operator name appears in the upper left corner. You can view each attribute name and data type.

   If you want to minimize the operator, click the arrow in the upper right corner and the Mapping Editor displays the operator as an icon on the canvas.

Figure 6–5 Mapping Editor Showing a Table Operator Source



Adding Operators that Bind to Repository Objects

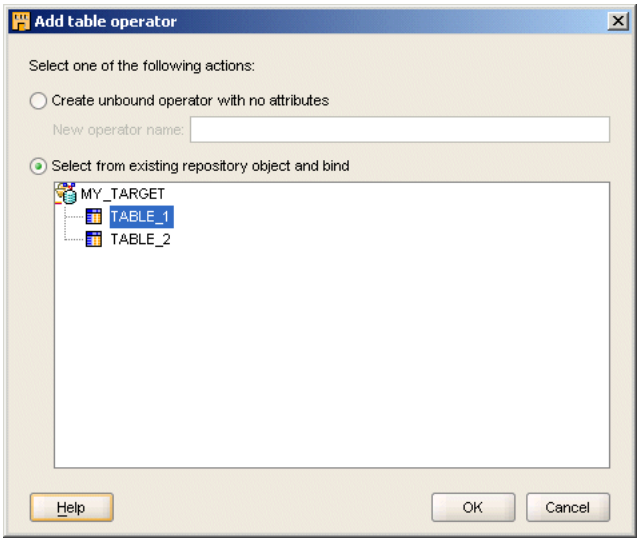
Table 6–5 lists the operators that you can bind to associated objects in the repository.

Table 6–5 Operators That You Can Bind to Repository Objects

List of Operators	List of Operators (continued)
■ Advanced Queue Operators	■ Pre Mapping Process Operators
■ Cube Operators	■ Post Mapping Process Operators
■ Dimension Operators	■ Sequence Operators
■ External Table Operators	■ Table Operators
■ Flat File Operators	■ Transformation Operators
■ Dimension Operators	■ View Operators
■ Materialized View Operators	■

Figure 6–6 shows the dialog for adding a table operator.

Figure 6–6 Add Table Operator Dialog



Add Operator Dialog

When you add an operator that you can bind to a repository object, the Mapping Editor displays the **Add Mapping <operator name>** dialog. Select one of the following options:

- Create Unbound Operator with No Attributes
- Select from Existing Repository Object and Bind

Create Unbound Operator with No Attributes

Use this option when you want to use the Mapping Editor to define a new repository object such as a new staging area table or a new target table.

After you select **Create Unbound Operator with No Attributes,** type a name for the new object. Warehouse Builder displays the operator on the canvas without any attributes.

You can now add and define attributes for the operator as described in "Editing Operators" on page 6-14. Next, to create the new repository object in a target module, right-click the operator and select **Create and Bind...**

For an example on how to use this option in a mapping design, see "Example: Using the Mapping Editor to Create Staging Area Tables" on page 6-20.

Select from Existing Repository Object and Bind

Use this option when you want to add an operator based on an object you previously defined or imported into the repository.

Either type the prefix to search for the object or select from the displayed list of objects within the selected module.

To select multiple items, press the Control key as you click each item. To select a group of items located in a series, click the first object in your selection range, press the Shift key, and then click the last object.

You can add operators based on repository objects within the same module as the mapping or from other modules. If you select a repository object from another module, the Mapping Editor creates a connector if one does not already exist. The connector establishes a path for moving data between the mapping location and the location of the repository object.

Editing Operators

Each operator has an editor associated with it. Use the operator editor to specify general and structural information for operators, groups, and attributes. In the operator editor you can add, remove, or rename groups and attributes. You can also rename an operator.

Editing operators is different from assigning loading properties and conditional behaviors. To specify loading properties and conditional behaviors, use the properties windows as described in "Setting Operator, Group, and Attribute Properties" on page 6-28.

To edit an operator, group, or attribute:

1. Select an operator from the Mapping Editor canvas.

   Or select any group or attribute within an operator.

2. Right-click and select **Open Details.**

   The Mapping Editor displays the operator editor with the Name Tab, Groups Tab, and Input and Output Tabs for each type of group in the operator.

   Some operators include additional tabs. For example, the Match Merge operator includes tabs for defining Match rules and Merge rules. For information on these additional tabs, look up the operator in Chapter 26, "Data Flow Operators".

3. Follow the prompts on each tab and click **OK** when you are finished.

Name Tab

The Name tab displays the operator name and an optional description. You can rename the operator and add a description. Name the operator according to the conventions listed in "Mapping Naming Conventions" on page 6-16.

Groups Tab

Edit group information on the Groups tab.

Each group has a name, direction, and optional description. You can rename groups for most operators but cannot change group direction for any of the operators. A group can have one of these directions: Input, Output, Input/Output.

Depending on the operator, you can add and remove groups from the Groups tab. For example, you add input groups to Joiners and output groups to Splitters.

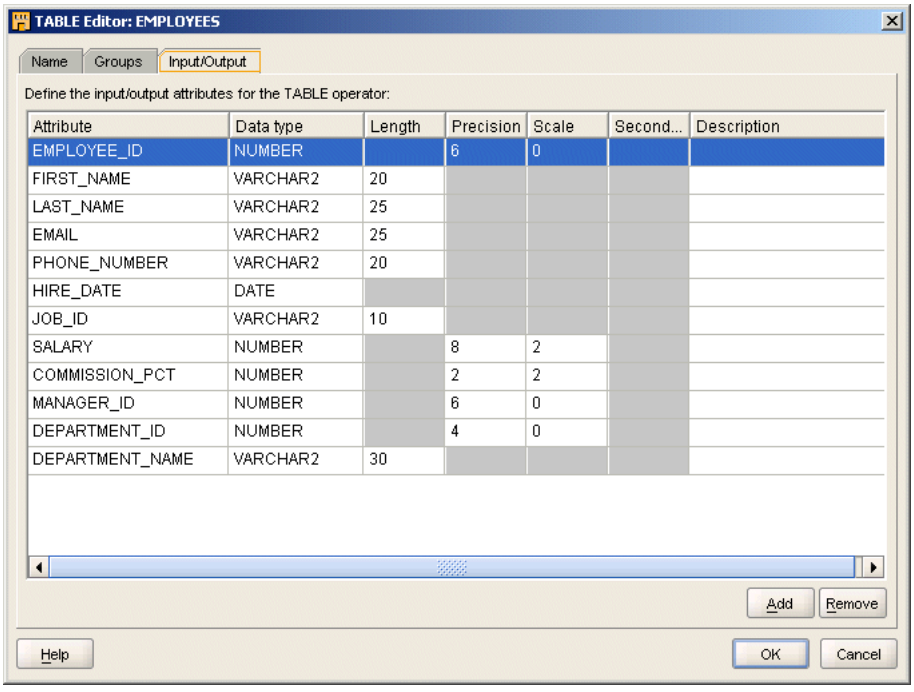
Input and Output Tabs

The operator editor displays a tab for each type of group displayed on the Groups tab. Each of these tabs displays the attribute name, data type, length, precision, scale, seconds precision, and optional description.

Edit attribute information on the each of the remaining tabs.

Figure 6–7 shows an Input/Output tab on the Operator Editor. In this example, the operator is a table and therefore has only the Input/Output tab. Other operators can have an Input tab and an Output tab.

Figure 6–7 Input/Output Tab on the Operator Editor



You can add, remove, and edit attributes. The Mapping Editor greys out properties that you cannot edit. For example, if the data type is NUMBER, you can edit the precision and scale but not the length.

To assign correct values for data type, length, precision, and scale in an attribute, follow PL/SQL rules. When you synchronize the operator, Warehouse Builder checks the attributes based on SQL rules.

Mapping Naming Conventions

The rules for naming objects in the Mapping Editor depend on the naming mode you select in Naming Preferences on page 3-8. Warehouse Builder maintains a business and a physical name for each object in the repository. The business name is its descriptive business name. The physical name is the name Warehouse Builder uses when generating code.

When you name objects while working in one naming mode, Warehouse Builder creates a default name for the other mode. Therefore, when working in the business name mode, if you assign a mapping a name that includes mixed cases, special characters and spaces, Warehouse Builder creates a default physical name for you. For example, if you save a mapping with the business name *My Mapping (refer to doc#12345),* the default physical name is *MY_MAPPING_REFER_TO_DOC#12345*.

When you name or rename objects in the Mapping Editor, use the following naming conventions.

Naming and Describing Mappings

In the physical naming mode, a mapping name can be from 1 to 30 alphanumeric characters and blank spaces are not allowed. In the business naming mode, the limit is 2000 characters and blank spaces and special characters are allowed. In both naming modes, the name should be unique across the project.

Note for scheduling mappings: If you intend to schedule the execution of the mapping as described in Chapter 28, "Scheduling ETL Objects", there is an additional consideration. For any ETL object you want to schedule, the limit is 25 characters for physical names and 1995 characters for business names. Follow this additional restriction to enable Warehouse Builder to append to the mapping name the suffix *_job* and other internal characters required for deployment and execution.

After you create the mapping definition, you can view its physical and business name on the mapping properties sheet. Right-click the mapping from the Design Center, select **Properties,** and view the names on the General tab.

Edit the description of the mapping as necessary. The description can be between 2 and 2,000 alphanumeric character and can contain blank spaces.

Naming Conventions for Attributes and Groups

You can rename groups and attributes independent of their sources. Attribute and group names are logical. Although attribute names of the object are often the same as the attribute names of the operator to which they are bound, their properties remain independent of each other. This protects any expression or use of an attribute from corruption if it is manipulated within the operator.

Naming Conventions for Operators

Business names for operator must meet the following requirements:

- The length of the operator name can be any string of 200 characters.
- The operator name must be unique on its attribute group, attribute and display set level with respect to its parent.

Physical names must meet the following requirements:

- All objects other than operators can contain a maximum of 30 characters. However, the limit is 28 for operators since Warehouse Builder reserves two characters for use when navigating using the OMB Scripting Language.

- The operator name must be unique on its group, attribute and display set level with respect to its parent.
- The operator name must conform to the syntax rules for basic elements as defined in the Oracle SQL Reference.

In addition to physical and business names, some operators also have bound names. Every operator associated with a repository object has a bound name. During code generation, Warehouse Builder uses the bound name to reference the operator to its repository object. Bound names have the following characteristics:

- Bound names need not be unique.
- Bound names must conform to the general Warehouse Builder physical naming rules.
- Typically, you do not change bound names directly but indirectly by synchronizing from an operator to the repository.
- When you rename the business name for an operator or attribute, Warehouse Builder propagates the new business name as the bound name when you synchronize. However, business names can be up to 200 character while bound names are limited to 30 characters. Therefore, Warehouse Builder uses the first 30 characters of the business name for the bound name.

Using Display Sets

A display set is a graphical representation of a subset of attributes. Use display sets to limit the number of attributes visible in an operator and simplify the display of a complex mapping.

By default, operators contain three predefined display sets, ALL, MAPPED, and UNMAPPED. Table 6–6 describes the default Display Sets.

Table 6–6 Default Sets

Display Set	Description
ALL	Includes all attributes in an operator.
MAPPED	Includes only those attributes in an operator that are connected to another operator.
UNMAPPED	Includes only those attributes that are not connected to other attributes.

Defining Display Sets

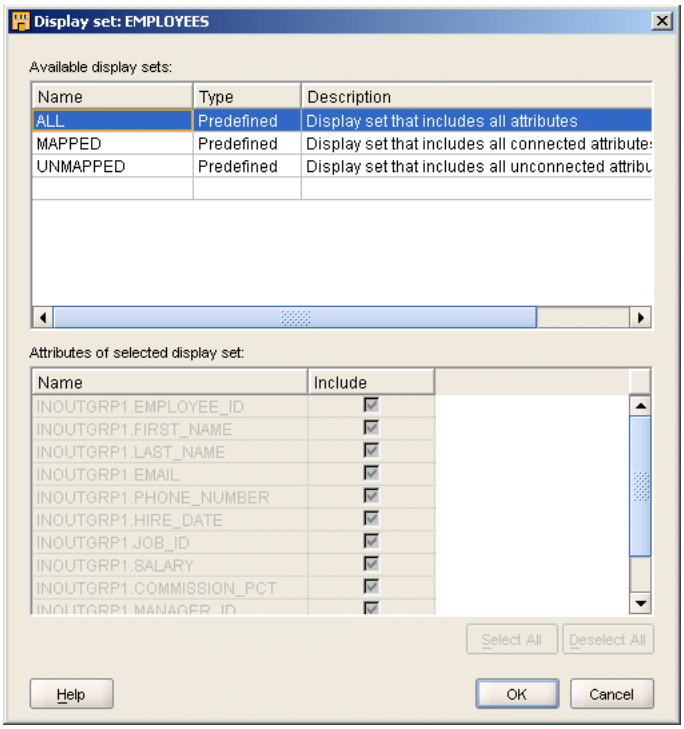
You can define display sets for any operator in a mapping.

To define a display set:

1. Right-click an operator, and select **Display Set**.

   The Display Set dialog appears, as shown in Figure 6–8.

Figure 6–8 Display Set Dialog



2. Click the row below UNMAPPED and enter a name and description for the new display set.
3. All available attributes for the operator appear in **Attributes of selected display set**. The Type column is automatically set to User defined.

   You cannot edit or delete a Predefined attribute set.
4. In the Include column, select each attribute you want to include in the display set.

   Click **Select All** to include all attributes and **Deselect All** to exclude all the attributes.
5. Click **OK.**

   The group for the operator now lists only those attributes contained within the Attribute Set selected for display.

Selecting a Display Set

If a group contains more than one display set, you can select a different display set from a list using the View menu.

To select a Display Set:

1. Right-click a group in an operator.
2. Click **Select Display Set** and select the desired display set.

Connecting Operators

After you select mapping source operators, operators that transform data, and target operators, you are ready to connect them. Data flow connections graphically represent how the data flows from a source, through operators, and to a target.

You can connect operators by one of the following methods:

- **Connecting Attributes**: Connect individual operator attributes to each other one at a time.
- **Connecting Groups**: Define criteria for connecting all the attributes between two groups.
- Using an Operator Wizard: For operators such as the Pivot operator and Name-Address operator, you can use the wizard to define data flow connections.

Connecting Attributes

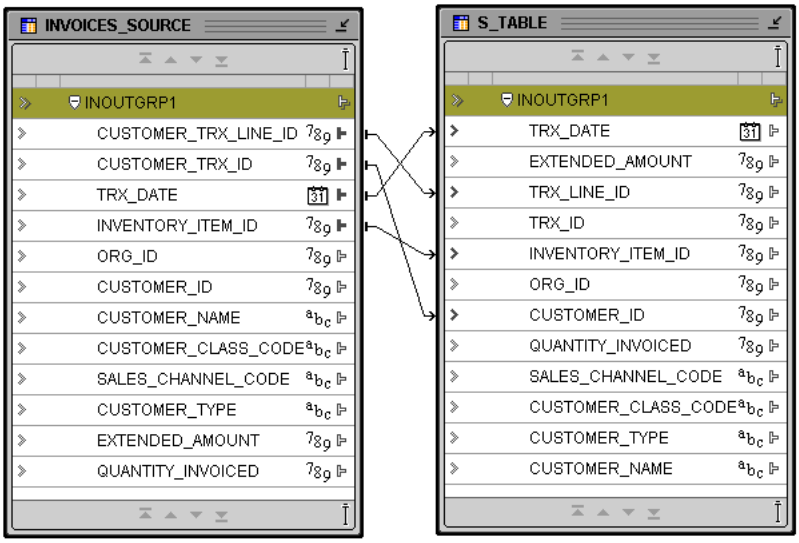
You can draw a line from a single output attribute of one operator to a single input attribute of another operator.

To connect attributes:

1. Click and hold down the mouse button while the pointer is positioned over an output attribute.
2. Drag the mouse away from the output attribute and toward the input attribute to which you want data to flow.

   As you drag the mouse, a line appears on the Mapping Editor canvas to indicate a connection, as shown in Figure 6–9.
3. Release the mouse over the input attribute.

Figure 6–9 Connected Operators in a Mapping



4. Repeat steps one through three until you create all the required data flow connections.

When connecting attributes, keep the following rules in mind:

- You cannot connect to the same input attribute twice.
- You cannot connect attributes within the same operator.
- You cannot connect out of an input only attribute nor can you connect into an output only attribute.

- You cannot connect operators in such a way as to contradict an established cardinality. Instead, use a Joiner operator.

Connecting Groups

When you connect groups, the Mapping Editor assists you by either automatically copying the attributes or prompts you for more information as described in "Using the Connect Operators Dialog" on page 6-21.

If you connect from one operator group to a target group with no existing attributes, the Mapping Editor automatically copies the attributes and connects the attributes. This is useful for designing mappings such shown in "Example: Using the Mapping Editor to Create Staging Area Tables".

Example: Using the Mapping Editor to Create Staging Area Tables

You can use the Mapping Editor with an unbound table operator to quickly create staging area tables.

The following instructions describe how to create a staging table based on an existing source table. You can also use these instructions to create views, materialized views, flat files, and transformations.

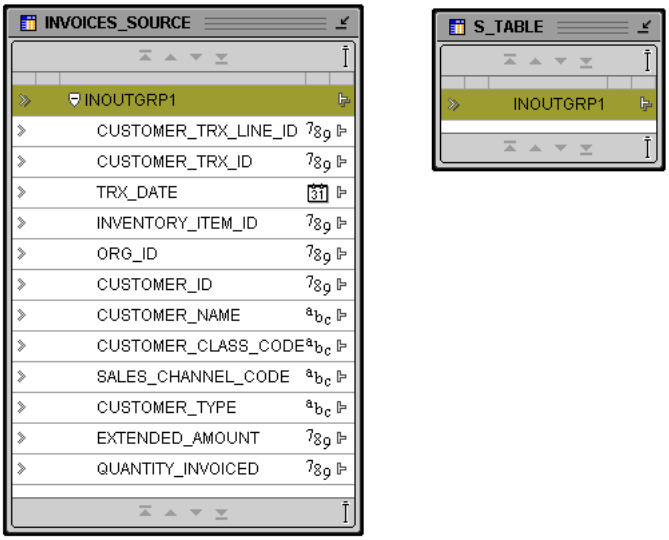
To map a source table to a staging table:

1. In the Mapping Editor, add a source table.

   From the menu bar, select **Mapping**, select **Add**, then select **Data Sources/Targets**. In the **Data Sources/Targets** menu, select **Table Operator.**

2. Use the **Add Table Operator** dialog to select and bind the source table operator in the mapping. From the Add Table Operator dialog, select **Create unbound operator with no attributes.** The mapping should now resemble Figure 6–10 with one source table and one staging area table without attributes.

Figure 6–10 Unbound Staging Table without Attributes and Source Table



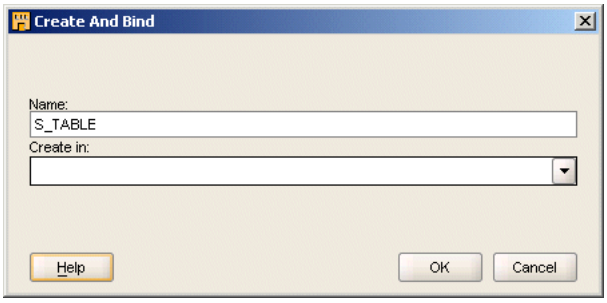
3. With the mouse button positioned over the group in the source operator, click and hold down the mouse button.

4. Drag the mouse to the staging area table group.

Warehouse Builder copies the source attributes to the staging area table and connects the two operators.

5. In the Mapping Editor, select the unbound table you added to the mapping. Right-click and select **Create and Bind.** Warehouse Builder displays the dialog shown in Figure 6–11.

Figure 6–11 Create and Bind Dialog



6. In **Create in,** specify the target module in which to create the table.

   Warehouse Builder creates the new table in the target module you specify.

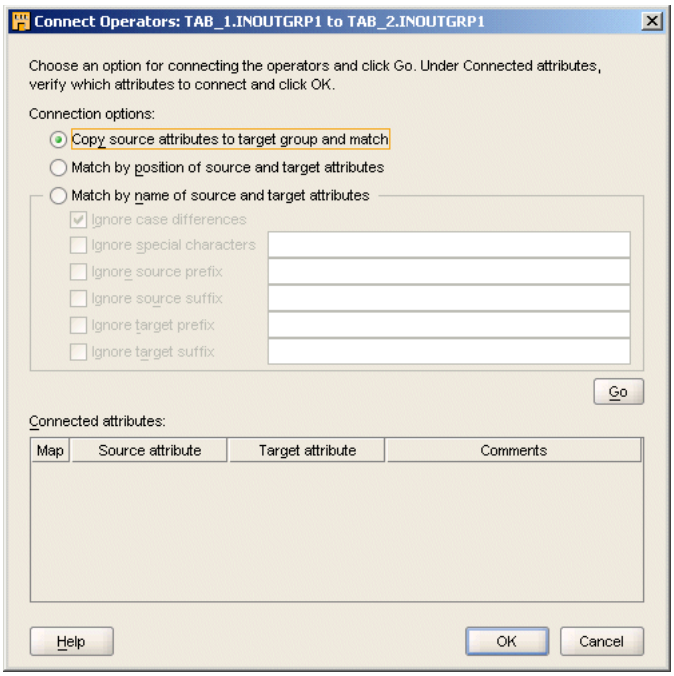
Using the Connect Operators Dialog

If you connect from one operator to a target operator with existing attributes, the Mapping Editor launches the Connect Operators dialog as shown in Figure 6–12.

Select one of the following criteria for copying and connecting attributes:

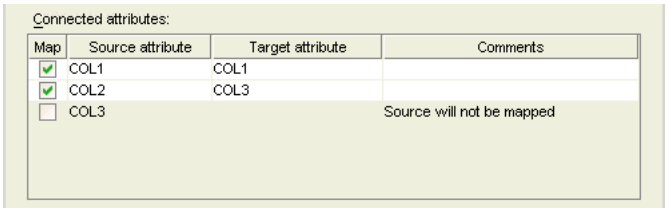
- Copy Source Attributes to Target Group and Match
- Match by Position of Source and Target Attributes
- Match by Name of Source and Target Attributes

Figure 6–12 Connect Operators Dialog Displaying Attributes to be Mapped



After you select one of the three options, select **Go.** The Connect Operators dialog displays a list of connected attributes as shown in Figure 6–13.

Figure 6–13 Connected Attributes



You can deselect attributes by clearing the **Map** check box. View the results of your selections under **Comments.**

When you select **OK,** Warehouse Builder copies the source attributes to the target group and connects the attributes.

Copy Source Attributes to Target Group and Match

Use this option to copy source attributes to a target group that already contains attributes. Warehouse Builder connects from the source attributes to the new target attributes based on the selections you make in the Connect Operators dialog. Warehouse Builder does not perform this operation on target groups that do not accept new input attributes such as dimension and cube target operators.

Match by Position of Source and Target Attributes

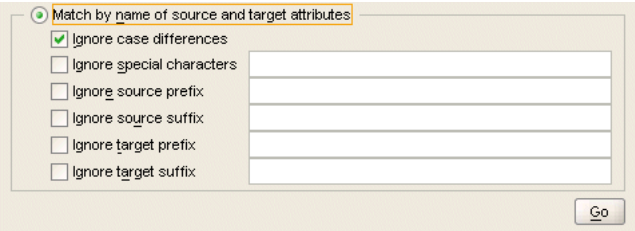
Use this option to connect existing attributes based on the position of the attributes in their respective groups. The Mapping Editor connects all attributes in order until all attributes for the target are matched. If the source operator contains more attributes than the target, then the remaining source attributes are left unconnected.

Match by Name of Source and Target Attributes

Use this option to connect attributes with matching names, as shown in Figure 6–14. By selecting from the list of options, you connect between names that do not match exactly. You can combine the following options:

- **Ignore case differences:** Considers the same character in lower-case and upper-case a match. For example, the attributes FIRST_NAME and First_Name match.
- **Ignore special characters:** Specify characters to ignore during the matching process. For example, if you specify a hyphen and underscore, the attributes FIRST_NAME, FIRST-NAME, and FIRSTNAME all match.
- **Ignore source prefix, Ignore source suffix, Ignore target prefix, Ignore target suffix:** Specify prefixes and suffixes to ignore during matching. For example, if you select Ignore source prefix and enter USER_ into the text field, then the source attribute USER_FIRST_NAME matches the target attribute FIRST_NAME.

Figure 6–14 Match by Name Matching Options



After you set the matching criteria, click **Go.**

The **Displayed Mappings** field displays the possible connections between attributes which you can verify and deselect before implementing.

Using Pluggable Mappings

You can reuse the data flow of a mapping by creating a pluggable mapping around the portion of the flow you want to reuse. A *pluggable mapping* is a reusable grouping of mapping operators that works as a single operator. It is similar to the concept of a function in a programming language and is a graphical way to define a function.

Note: The use of pluggable mappings requires the Warehouse Builder Enterprise ETL Option.

Once defined, a pluggable mapping appears as a single mapping operator, nested inside a mapping. You can reuse a pluggable mapping more than once in the same mapping, or in other mappings. You can include pluggable mappings within other pluggable mappings.

Like any operator, a pluggable mapping has a *signature* consisting of input and output attributes that enable you to connect it to other operators in various mappings. The signature is similar to the input and output requirements of a function in a programming language.

A pluggable mapping can be either *reusable* or *embedded*:

- **Reusable pluggable mapping:** A pluggable mapping is reusable if the metadata it references can exist outside of the mapping in question. You can store reusable pluggable mappings either as standalone pluggable mappings, which are private

for your use, or in folders (libraries). Users who have access to these folders can use the pluggable mappings as templates for their work.

- **Embedded pluggable mapping:** A pluggable mapping is embedded if the metadata it references is owned only by the mapping or pluggable mapping in question. An embedded pluggable mapping is not stored as either a standalone mapping or in libraries on the Global Explorer. It is stored only within the mapping or the pluggable mapping that owns it, and you can access it only by editing the object that owns it. To validate or generate the code for an embedded pluggable mapping, you must validate or generate the code for the object that owns it.

Pluggable Mapping Folders

A folder is a grouping mechanism for pluggable mappings. You can keep your pluggable mappings private, or you can place them into folders (libraries) and then publish them so that others can access them for their design work. To create a new folder to store pluggable mappings, locate Pluggable Mappings on the Project Explorer, right-click Pluggable Mapping Folders, and select **New**.

In the Create Pluggable Mapping Folder wizard, enter the name of the folder and an optional description. Click **OK** to save the folder and exit the wizard. The folder appears on the Project Explorer. You can move pluggable mappings to a folder by pasting them into the desired folder on the tree.

Creating a Pluggable Mapping

Pluggable mappings are usually predefined and used when required.You can create pluggable mappings either from within a mapping by using the mapping editor, or from the navigation tree by using the wizard. The wizard is the faster way to create a pluggable mapping because it makes some default choices and guides you through fewer choices. You can make additional choices later in the Pluggable Mapping Editor. The editor presents you with all the settings in a series of tabs.

The Pluggable Mappings node in the navigation tree contains two nodes, Standalone and Pluggable Mapping Folders. You can create pluggable mappings from either of these nodes.

Standalone Pluggable Mapping

To create a pluggable mapping from the Standalone node, right-click Standalone, and select **New**. This opens the Create Pluggable Mapping wizard, which guides you through the process of creating a new pluggable mapping. Click **Help** for information on the values to be entered on each page of the wizard.

Once you create a new pluggable mapping, Warehouse Builder opens the pluggable mapping editor and displays the name of the pluggable mapping on the title bar. The pluggable mapping editor is similar to the mapping editor, and you can add the desired operators from the palette to create a mapping.

A pluggable mapping is considered as an operator by the Warehouse Builder. You can insert it into any mapping. In the mapping editor, drag and drop Pluggable Mapping from the palette onto the canvas. This opens the Add Pluggable Mapping dialog. You can select the desired pluggable mapping and add it to your mapping.

Pluggable Mapping Folder

The Pluggable Mapping Folders node gives you the option of creating a pluggable mapping either at the time of creating a folder or after creating the folder. Right-click

Pluggable Mapping Folders and select **New**. This opens the Create Pluggable Mapping Folder dialog. Provide a name for the folder and a description (optional), and click **OK**. If you select the **Proceed to Pluggable Mapping Wizard** option, the Create Pluggable Mapping wizard opens and you can create a new pluggable mapping. If you do not select the Proceed to Pluggable Mapping Wizard option, only the pluggable mapping folder gets created. To create a pluggable mapping under this folder, right-click the folder and select **New**. This opens the Create Pluggable Mapping wizard.

Signature Groups

The signature is a combination of input and output attributes flowing to and from the pluggable mapping. Signature groups are a mechanism for grouping the input and output attributes.

A pluggable mapping must have at least one input or output signature group. Most pluggable mappings are used in the middle of a logic flow and have both input and output groups.

- To create an additional signature group, click **Add**. To overwrite the default name assigned to the group, type over its name in the **Group** column. Enter its orientation as an input or output group in the **Direction** column. Enter an optional description of the group in the **Description** column.
- To remove a signature group, select the group you want to remove and click **Remove**.

Click **Next** to continue with the wizard.

Input Signature

The input signature is the combination of input attributes that flow into the pluggable mapping. Define the input attributes for each input signature group you created.

If you defined multiple input signature groups, select the group to which you want to add attributes from the Group list box. Then click **Add** to add attributes. You can overwrite the default name given to each attribute by typing over the name in the Attribute column. You can change the data type of each attribute by clicking on its default data type and selecting a new data type from the resulting drop list. You can assign the length, precision, scale, and seconds precision by clicking the corresponding field and using the up and down arrows or typing in a number. Note that some of these fields are disabled depending on the data type you specify.

You can remove an attribute by selecting the attribute and clicking **Remove**.

Click **Next** to continue with the wizard.

Output Signature

The output signature is the combination of output attributes that flow out of the pluggable mapping. Define the output attributes for each output signature group you created.

If you defined multiple output signature groups, select the group to which you want to add attributes from the Group list box. Then click **Add** to add attributes. You can overwrite the default name given to each attribute by typing over the name in the Attribute column. You can change the data type of each attribute by clicking on its default data type and selecting a new data type from the resulting drop list. You can assign the length, precision, and scale by clicking the corresponding field and using the up and down arrows or typing in a number. Note that some of these fields are disabled depending on the data type you specify.

You can remove an attribute by selecting the attribute and clicking **Remove**.

Click **Next** to continue with the wizard.

You can also add an Input Signature or an Output Signature from the palette of the pluggable mapping editor. Note that a pluggable mapping can have only one Input Signature and Output Signature. Also, pluggable mapping Input and Output signatures can only be added within pluggable mappings. They cannot be added to normal mappings.

Pluggable Mapping Implementation

Implementation refers to the data transformation logic that the pluggable mapping performs. There are three types of pluggable mappings depending on how the metadata for the implementation is obtained by the Warehouse Builder:

- **Black-Box**: The Warehouse Builder provides and receives attributes and attribute groups to and from the transformation logic without being aware of the underlying functionality. This is identical to using a library function. The calling function uses the library function without being aware of how the function actually works.
- **White-Box**: The Warehouse Builder is aware of the underlying functionality of the transformation logic and can provide optimization while generating the code for the mapping. This is identical to using a private function in a program.
- **System-managed**: A system-managed implementation is identical to white-box implementation. However, the transformation logic is generated by the Warehouse Builder and not stored in the repository.

Pluggable Mapping Usage

Pluggable mapping usage refers to the utilization of a pluggable mapping in another mapping. The mapping which uses the pluggable mapping is known as the *utilizing context* of the pluggable mapping. The pluggable mapping is seen as a stage component in the mapping that utilizes it. This is identical to a function call, wherein a function is called and utilized in another function.

Pluggable Mapping Usage Signature

The signature of a pluggable mapping usage represents the signature of the pluggable mapping when it is utilized in another mapping. In the utilizing context, the signature is expressed as the attribute groups and the attributes of the pluggable mapping usage.

A signature is the only way by which data can be passed from a utilizing context to a pluggable mapping. The pluggable mapping cannot directly access the attributes and the attribute groups of a utilizing context.

Types of Pluggable Mapping Usage

There are two types of pluggable mapping usage depending on the implementation and how the signature of the mapping is captured:

- **Reusable Usage**: Reusable pluggable mapping usage captures only the signature information from the pluggable mapping. The implementation of the pluggable mapping is not included in the usage.
- **Embedded Usage**: Embedded pluggable mapping usage represents a reusable pluggable mapping or an embedded pluggable mapping in the utilizing context by capturing the signature of the pluggable mapping. If an embedded pluggable

mapping usage is used to represent a reusable white-box, the implementation of the reusable white-box will also be copied into the embedded pluggable mapping. The implementation of an embedded usage is owned by the utilizing context and is always available to it.

When a reusable or embedded pluggable mapping usage is used to represent a reusable pluggable mapping, information about how the pluggable mapping usage is derived from the reusable pluggable mapping is stored in the usage. This information is called *binding* and the pluggable mapping usage is bound to that reusable mapping. If such information is not available, the reusable pluggable mapping is *unbound*.

Pluggable Mapping Editor

The pluggable mapping editor is similar to the mapping editor. Use the main panel to select and edit the operators that constitute your pluggable mapping. For more information on using this editor to design pluggable mappings, consult these topics:

- Using Pluggable Mappings on page 6-23
- About the Mapping Editor on page 6-5
- Adding Operators on page 6-11
- Editing Operators on page 6-14
- Connecting Operators on page 6-18
- Setting Operator, Group, and Attribute Properties on page 6-28
- Synchronizing Operators and Repository Objects on page 6-29

Setting Mapping Properties

When you select white space on the mapping canvas, the editor displays the mapping properties in the property inspector along the left side. You can set the following property for the mapping:

- Target Load Order on page 6-27

Target Load Order

If your mapping includes only one target or is a SQL*Loader or ABAP mapping, target load ordering does not apply. Accept the default settings and continue with your mapping design.

When you design a PL/SQL mapping with multiple targets, Warehouse Builder calculates a default ordering for loading the targets. If you define foreign key relationships between targets, Warehouse Builder creates a default order that loads the parent and then the child. If you do not create foreign key relationships or if a target table has a recursive relationship, Warehouse Builder assigns a random ordering as the default.

You can override the default load ordering by setting the Target Load Order property. If you make a mistake when reordering the targets, you can restore the default ordering by selecting the Reset to Default option.

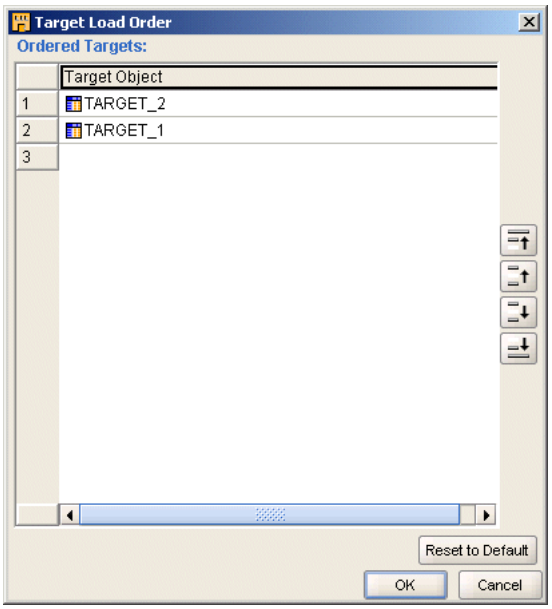
To specify the loading order for multiple targets:

1. Click on whitespace in the mapping canvas to view the mapping properties in the Property Window in the upper right corner.

2. Go to the Map Targets Load Order property and click the Ellipses button on the right side.

   Warehouse Builder displays the Map targets load order dialog as shown in Figure 6–15 which shows TARGET2 loading before TARGET1.

Figure 6–15 Target Load Order Dialog



3. To change the loading order, select a target and use the shuttle buttons on the right to move the target up or down on the list.

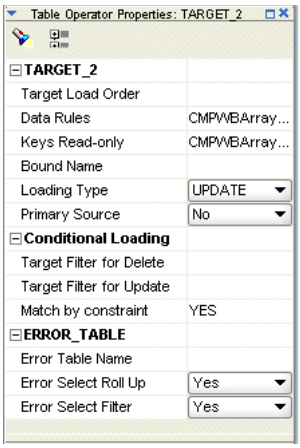
Reset to Default

Use the Reset to Default button to instruct Warehouse Builder to recalculate the target loading order. You may want to recalculate if you made an error reordering the targets or if you assigned an order and later change the mapping design such that the original order became invalid.

Setting Operator, Group, and Attribute Properties

When you select an object on the canvas, the editor displays its associated properties in the property inspector along the left side as shown in Figure 6–16.

Figure 6–16 Property Inspector for a Table Operator



You can view and set the following types of properties:

- **Operator Properties:** Properties that affect the operator as a whole. The properties you can set depend upon the operator type. For example, the steps for Using Oracle Source and Target Operators differ from the steps for Using Flat File Source and Target Operators.
- **Group Properties:** Properties that affect a group of attributes. Most operators do not have properties for their groups. Examples of operators that do have group properties include the splitter operator and the deduplicator.
- **Attribute Properties:** Properties that pertain to attributes in source and target operators. Examples of attribute properties are data type, precision, and scale.

Synchronizing Operators and Repository Objects

Many of the operators you use in a mapping have corresponding definitions in the Warehouse Builder repository. This is true of source and target operators such as table and view operators. This is also true of other operators such as sequence and transformation operators whose definitions you may want to use across multiple mappings. As you make changes to these operators, you may want to propagate those changes back to the repository object.

You have the following choices in deciding the direction in which you propagate changes:

Synchronizing From a Repository Object to an Operator: After you begin using mappings in a production environment, there may be changes to the sources or targets that impact your ETL designs. Typically, the best way to manage these changes is through the Warehouse Builder Dependency Manager described in Chapter 31, "Managing Metadata Dependencies". Use the Dependency Manager to automatically evaluate the impact of changes and to synchronize all effected mappings at one time. Alternatively, in the Mapping Editor, you can manually synchronize objects as described in "Synchronizing From a Repository Object to an Operator" on page 6-30.

Synchronizing from an Operator to a Repository Object: When you make changes to an operator in a mapping, you may want to propagate those changes to its corresponding repository definition. For example, the sources you imported and used in a mapping may have complex physical names for its attributes.

You can synchronize in the following method:

- **Synchronizing An Operator:** You can select a single operator and synchronize it with the definition of a specified repository object.

Note that synchronizing is different from refreshing. The refresh command ensures that you are up-to-date with changes made by other users in a multiuser environment. Synchronizing matches operators with their corresponding repository objects.

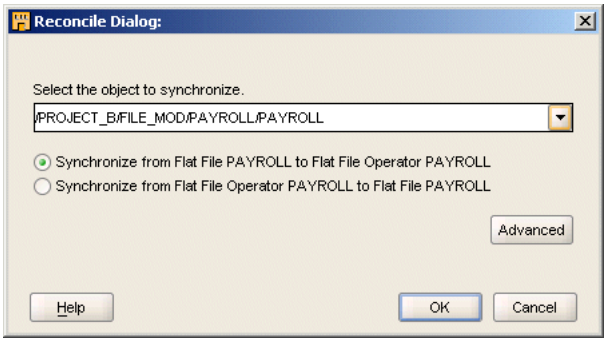
Synchronizing An Operator

To synchronize an operator, complete the following steps:

1. Select an operator on the Mapping Editor canvas.
2. From the **Edit** menu, select **Synchronize** or right-click the header of the operator and select **Synchronize.**

   The Synchronize Operator dialog displays as shown in Figure 6–17.

Figure 6–17 Synchronizing an Operator



3. By default, Warehouse Builder selects the option for you to synchronize your selected operator with its associated object in the repository. You can accept the default or select another repository object from the list box.

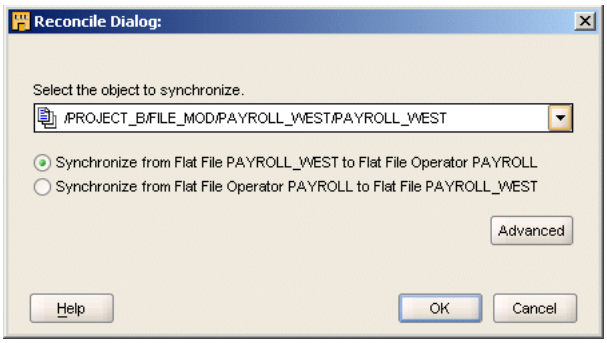
   In this step you also specify either Synchronizing From a Repository Object to an Operator or select the option for Synchronizing from an Operator to a Repository Object.
4. As an optional step, click **Advanced** to set the Matching Strategies.
5. Click **OK.**

Synchronizing From a Repository Object to an Operator

In the Mapping Editor, you can synchronize from a repository object for any of the following reasons:

- **Manually propagate changes:** Propagate changes you made in a repository object to its associated operator. Changes to the repository object can include structural changes, attribute name changes, attribute data type changes. To automatically propagate changes in a repository object across multiple mappings, see Chapter 31, "Managing Metadata Dependencies".
- **Synchronize an operator with a new repository object:** You can associate an operator with a new repository object if, for example, you migrate mappings from one version of a data warehouse to a newer version and maintain different object definitions for each version. Figure 6–18 shows an example of synchronizing a flat file operator with a new repository object.

Figure 6–18 Synchronizing from a Different Repository Object



- **Prototype mappings using tables:** When working in the design environment, you could choose to design the ETL logic using tables. However, for production, you may want to the mappings to source other repository object types such as views, materialized views, or cubes.

Synchronizing Operators based on Repository Objects

Table 6–7 lists operators and the types of repository objects from which you can synchronize.

Table 6–7 Operators Synchronized with Repository Objects

To: Operator	From: Repository Object Type
Cube Operator	Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
Dimension Operator	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
External Table Operator	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
Flat File Operator	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
Key Lookup Operator	Tables only
Materialized View Operator	Tables, External Tables, Views, Materialized Views, Files, Dimensions and Cubes
Post Mapping Process Operator	Transformations only
Pre Mapping Process Operator	Transformations only
Sequence Operator	Sequences only
Table Operator	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
Transformation Operator	Transformations only
View Operator	Tables, External Tables, Views, Materialized Views, Files, Dimensions and Cubes

Note that when you synchronize from an external table operator, Warehouse Builder updates the operator based on the repository external table only and not its associated

flat file. To update an operator such as external table based on its associated flat file, see "Synchronizing an External Table Definition with a Record in a File" on page 14-28.

Synchronizing from an Operator to a Repository Object

As you make changes to operators in a mapping, you may want to propagate those changes back to a repository object. By synchronizing, you can propagate changes from the following operators: tables, views, materialized views, transformations, and flat file operators.

Synchronize from the operator to a repository object for any of the following reasons:

- **Propagate changes:** Propagate changes you made in an operator to its associated repository object. When you rename the business name for an operator or attribute, Warehouse Builder propagates the first 30 characters of the business name as the bound name.
- **Replace repository objects:** Synchronize to replace an existing repository object.

Synchronizing from an operator has no impact on the dependent relationship between other operators and the repository object. Table 6–8 lists the operators from which you can synchronize.

Table 6–8 Outbound Synchronize Operators

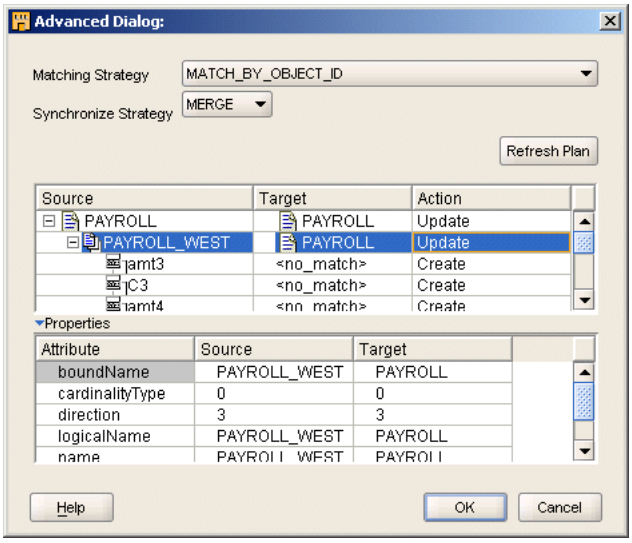
Mapping Objects	Create Repository Objects	Propagate Changes	Replace Repository Objects	Notes
External Tables	Yes	Yes	Yes	Updates the repository external table only and not the flat file associated with the external table. See "Synchronizing an External Table Definition with a Record in a File" on page 14-28.
Flat Files	Yes	Yes	No	Creates a new, comma-delimited flat file for single record type flat files only. Cannot replace an existing file.
Mapping Input Parameters	Yes	Yes	Yes	Copies input attributes and data types as input parameters.
Mapping Output Parameters	Yes	Yes	Yes	Copies output attribute and data types as return specification for the function.
Materialized Views	Yes	Yes	Yes	Copies attributes and data types as columns.
Tables	Yes	Yes	Yes	Copies attributes and data types as columns. Constraint properties are not copied.
Transformations	Yes	Yes	Yes	
Views	Yes	Yes	Yes	Copies attributes and data types as columns.

Advanced Options for Synchronizing

Use the Advanced Dialog to view and edit the details of how Warehouse Builder synchronizes your selected objects. After you select from the Matching Strategies, click **Refresh Plan** to view the actions Warehouse Builder takes, as shown in Figure 6–19.

In the context of synchronizing, *source* refers to the object from which to inherit differences and *target* refers to the object to be changed. For example, if Figure 6–19, the flat file PAYROLL_WEST is the source and the flat file operator PAYROLL is the target. Therefore, Warehouse Builder creates new attributes for the PAYROLL operator to correspond to fields in the flat file PAYROLL_WEST.

Figure 6–19 Advanced Synchronizing Options



Matching Strategies

Set the matching strategies to determine how Warehouse Builder compares an operator to a repository object. If synchronization introduces changes such as adding or deleting attributes in an operator, Warehouse Builder refreshes the Mapping Editor. If synchronizing removes an operator attribute, data flow connections to or from the attribute are also removed. If synchronizing adds an operator attribute, the Mapping Editor displays the new attributes at the end of the operator. Data flow connections between matched attributes are preserved. If you rename an attribute in the source object, it is interprets it as if the attribute were deleted and a new attribute added.

You can specify the following strategies for reconciling an object in a mapping:

- Match by Object Identifier
- Match by Bound Name
- Match by Position

Match by Object Identifier

This strategy compares the unique object identifiers of an operator attributes with those of a repository object. Match by object identifier is not available for synchronizing an operator and repository object of different types such as a view operator and a repository table.

Use this strategy if you want the target object to be consistent with changes to the source object and if you want to maintain separate business names despite changes to physical names in the target object.

Warehouse Builder removes attributes from the source object that do not correspond to attributes in the target object. This can occur when an attribute is added to the source or removed from the repository object without properly synchronizing the change.

Match by Bound Name

This strategy matches the bound names of the operator attributes to the physical names of the repository object attributes. Matching is case-sensitive.

Use this strategy if you want bound names to be consistent with physical names in the repository object. You can also use this strategy with a different repository object if there are changes in the repository object that would change the structure of the operator.

Warehouse Builder removes attributes of the operator that cannot be matched with those of the repository object. Attributes of the selected repository object that cannot be matched with those of the operator are added as new attributes to the operator. Because bound names are read-only after you have bound an operator to a repository object, you cannot manipulate the bound names to achieve a different match result.

Match by Position

This strategy matches operator attributes with columns, fields, or parameters of the selected repository object by position. The first attribute of the operator is synchronized with the first attribute of the repository object, the second with the second, and so on.

Use this strategy to synchronize an operator with a different repository object and you want to preserve the business names in the operator attributes. This strategy is most effective when the only changes to the repository object are the addition of extra columns, fields, or parameters at the end of the object.

If the target object has more attributes than the source object, then Warehouse Builder removes the excess attributes. If the source object has more attributes than target object, Warehouse Builder adds as new attributes.

Debugging a Mapping

You can use the Mapping Editor to debug complex data flows you design in mappings. Once you begin a debug session and connect to a valid target schema, the debugging functions appear on the toolbar and under Debug on the Mapping Editor main menu. You can run a debugging session using a defined set of test data and follow the flow of data as it is extracted, transformed, and loaded to ensure that the designed data flow behaves as expected. If you find problems, you can correct them and restart the debug session to ensure that the problems have been fixed before proceeding to deployment.

Before you Begin

Ensure that you are connected to a Control Center and that the Control Center is running.

Starting a Debug Session

To start a debug session, from the Mapping Editor, select **Debug** and then **Start**, or you can click Start Debug on the toolbar. The Mapping Editor switches to debug mode with the debug panels appearing in the bottom and the side of the editor. and the debugger connects to the appropriate Control Center for the project. The debug-generated code is deployed to the target schema specified by the location of the module that contains the map being debugged.

Note: When the connection cannot be made, an error message is display and you have an option to edit the connection information and retry.

After the connection has been established, a message displays to indicate that you may want to define test data. When you have previously defined test data, then you are asked if you want to continue with initialization.

To debug a mapping, each source or target operator must be bound to a database object and test data must be defined for the database object. By default, the debugger uses the same source and target data that is currently defined for the non-debug deployment of the map.

The Debug Panels of the Mapping Editor

When the Mapping Editor is opened in Debug mode it has new panels Debug Info Panel and Debug Data Panel.

Debug Info Panel

When the Mapping Editor is in Debug mode, the left middle panel is the Debug Info panel which contains the following tabs:

- **Messages:** Displays all debugger operation messages. These messages let you know the status of the debug session. This includes any error messages that occur while running the mapping in debug mode.
- **Breakpoints:** Displays a list of all breakpoints that you have set in the mapping. You can use the check boxes to activate and de-activate breakpoints. For more information, see "Setting Breakpoints" on page 6-37.
- **Test Data:** Displays a list of all data objects used in the mapping. The list also indicates which data objects have test data defined.

Debug Data Panel

When the Mapping Editor is in Debug mode, the Debug Data panel is the right bottom panel. The Debug Data Panel includes Step Data and watch point tabs that contain input and output information for the operators being debugged. The Step Data tab contains information about the current step in the debug session. Additional tabs can be added for each watch you set. These watch tabs allow you to keep track and view data that has passed or will pass through an operator regardless of the currently active operator in the debug session. Operators that have more than one input group or more than one output group display an additional drop down list that enables you to select a specific group.

If an operator has more than one input or output group then the debugger will have a drop down list in the upper right corner, above the input or output groups. Use this drop down list to select the group you are interested in. This applies both to the step data and to a watch.

Defining Test Data

Every source or target operator in the mapping is listed on the Test Data tab in the left bottom panel. It also contains the object type, the source, and a check mark that indicates that the database object has already been bound to the source or target operator.

The object type listed on the tab is determined by whether or not the column names in the data source you select (for example, a table) matches the columns in the mapping operators. There are two possible types:

- **Direct Access**. When there is an exact match then the type is listed as Direct Access.
- **Deployed as View**. When you choose a data source with columns that do not match the mapping operator columns, you can choose how you want the columns mapped. This object would then be deployed as a view when you run the mapping and the type will be listed as Deployed as View.

Click **Edit** to add or change the binding of an operator as well as the test data in the bound database objects. Before you can run the mapping in debug mode, each listed source or target operator must be bound and have a check mark. The need to have test data defined and available in the bound database object depends on what aspect of the data flow you are interested in focusing on when running the debug session. Typically, you will need test data for all source operators. Test data for target operators is usually necessary if you want to debug loading scenarios that involve updates or target constraints.

To define or edit test data:

1. From the Test Data tab in the Mapping Editor, select an operator from the list and click **Edit**. The Define Test Data dialog is displayed.
2. In the Define Test Data dialog, specify the characteristics of the test data that you want Warehouse Builder to use when it debugs. There are many characteristics that you can specify. For example, you can specify that the test data be from a new or existing database object or that you can or cannot manually edit the test data. Click Help on the Define Test Data dialog for more information.

Creating New Tables to Use as Test Data

When you create a new table using the Define Test Data dialog, the name of the table is the name of the data operator prefixed by `DBG_`. (Note that, if this name all ready exists in the target, then Warehouse Builder adds a sequence number as a suffix to guarantee a unique object name.) Warehouse Builder creates the table in the target schema that you specified when you started the debug run. The debugger does not automatically drop that table, consequently you can always reuse it for other sessions. Constraints are not carried over for the new table.

When you create a new table, Oracle Warehouse Builder creates the new table in the connected runtime schema. The new table has an automatically generated name and the value of the Debug Binding name changes to reflect the new table name. The new table has columns defined for it that exactly match the names and data types of the mapping source or target attributes. In addition, any data that is displayed in the grid at the time the table is created are copied into the newly created table.

Editing the Test Data

You can edit test data at anytime using the Define Test Data dialog. If you change the binding of the operator to another database object, you must re-initialize the debug session to implement the change before running the mapping again in debug mode.

Note: The data loaded in the target definitions will be implicitly committed. If you do not want the target objects updated, then you should create copies of target objects by clicking **Create New Table**.

Setting Breakpoints

If you are interested in how a specific operator is processing data, you can set a breakpoint on that operator which will cause a break in the debug session. This enables you to proceed quickly to a specific operator in the data flow without having to go through all the operators step by step. When the debug session gets to the breakpoint, you can run data through the operator step by step to ensure it is functioning as expected. Breakpoint settings are not stored when you close the mapping.

To set or remove a breakpoint:

1. From the Mapping Editor, click an operator and then select **Debug** and then **Set Breakpoint.** You can also click the Breakpoint button on the toolbar to toggle the breakpoint on and off for the currently highlighted operator.

   If you are setting the breakpoint, the name of the operator set as a breakpoint appears in the list on the Breakpoints tab on the left bottom panel. If you are removing the breakpoint the name is removed. Use the **Clear** button on the Breakpoint tab to remove breakpoints.

2. Uncheck or check the breakpoints on the Breakpoint tab to disable or enable them.

Setting Watches

The Step Data tab on the right bottom panel always shows the data for the current operator. If you want to keep track of data that has passed through any other operator irrespective of the active operator, you can set a watch.

Use watches to track data that has passed through an operator or in the case of sources and targets, the data that currently resides in the bound database objects. You can also set watch points on operators after the debug run has already passed the operator and look back to see how the data was processed by an operator in the data flow.

To set a watch:

From the Mapping Editor, click an operator and then select **Debug** and then **Set Watch.** You can also click the Set Watch button on the toolbar to toggle the watch on and off. The name of the operator will appear as an additional tab on the right bottom panel bottom containing the input and or output groups for the operator.

To remove a watch:

To remove a watch, again select the operator and use the watch button on the toolbar, use set watch from the debug menu or use toggle debug watch from the right mouse button menu.

Running the Mapping

After you have defined the test data connections for each of the data operators, you can initially generate the debug code by selecting **Re-initialize** from the Debug menu, or by clicking Re-initialize on the toolbar. Warehouse Builder generates the debug code and deploys the package to the target schema you specified.

You can choose to run the debug session in one of the following modes:

- Continue processing until the next breakpoint or until the debug run finishes by using the Resume button on the toolbar or the associated menu item.
- Process row by row using the Step button on the toolbar or use the associated menu item.

- Process all remaining rows for the current operator by using the Skip button on the toolbar or the associated menu item.
- Reset the debug run and go back to the beginning by using the Reset button or the associated item from the debug menu.

Selecting the First Source and Path to Debug

A mapping may have more than one source and more than one path to debug:

- When a mapping has more than one source then Warehouse Builder prompt you to designate the source with which to begin. For example, when two tables are mapped to a joiner, you will have to select the first source table you want to use when debugging.
- There may be multiple paths that the debugger can walk through after it has finished one path. For example, this is the case when you use a splitter. Having finished one path the debugger asks you whether you would like to complete the other paths as well.

The mapping finishes if all target operators have been processed or if the maximum number of errors as configured for the mapping has been reached. The debug connection and test data definitions are stored when you commit changes to the Warehouse Builder repository. Breakpoint and watch settings are not stored and must be re-set each time you open a mapping.

As the debugger runs it generates debug messages whenever applicable. You can follow the data flow through the operators. The active operator is indicated by a red dashed box surrounding the operator.

Debugging Mappings with Correlated Commit

How a mapping is debugged varies depending on whether the mapping has the Correlated Commit parameter set to ON or OFF:

- When you begin a debug session for a mapping that has the Correlated Commit parameter set to ON, the mapping is not debugged using paths. Instead, all paths are executed and all targets are loaded during the initial stepping through the mapping regardless of what path is chosen. Also, if one of the targets has a constraint violation for the step, then none of the targets are loaded for that step.
- When you begin a debug session for a mapping that has the Correlated Commit parameter set to OFF, the mapping is debugged using one path at a time. All other paths are left unexecuted and all other targets are not loaded unless the you reach the end of the original path and then chooses to go back and execute another path in the mapping.

For example: You have a mapping that has a source S1, connected to a splitter that goes to two targets T1 and T2:

- If Correlated Commit is OFF, then the mapping is debugged starting with S1. You can then choose either the path going to T1 or the path going to T2. If you choose the path to T1, the data going to T1 is processed and displayed, and the target T1 is loaded. After T1 is completely loaded, you are given the option to go back and execute the other path and load target T2.
- If Correlated Commit is ON, then the mapping is also debugged staring with S1 and you are given the option of choosing a path however in this case, the path you choose only determines the path that gets displayed in the mapping editor as you step through the data. All paths are executed simultaneously. This is also how a

mapping using Correlated Commit gets executed when the deployable code is run.

Setting a Starting Point

You can select an operator as a starting point, even if it is not a source. To set an operator as a starting point, start a debug session, then select the operator and click **Set as Starting Point** or choose the Set as Starting Point menu item.

When an operator is set as a starting point, Warehouse Builder combines all the upstream operators and sources into a single query, which is used as a source, and the operator is automatically used as the first source when stepping through the map. The operators that are upstream of the starting point operator are not steppable, and do not have displayable data, even if a watch point is set.

A good use of "set as starting point" would be if the map had three source tables that were all connected to a single joiner. Assuming that each source table contains a large number of rows, too many rows to efficiently step through in the debugger (say more than 50000 rows). In this case, it would be a good idea to set the joiner operator as a starting point, and limit the row count for the one of the source tables to a more manageable number of rows (say 500) by using the Test Data Editor. In this case it would be best to limit the row count of the source table that is effectively driving the joiner (that is, the source with which all the other sources are joined in the join condition).

Debugging Pluggable Submap Operators

You can also debug a map which contains one or more pluggable submap operators. This could include a user-defined pluggable submap operator from the pluggable folder, or a system-defined submap operator. When the debug session is started, the map will go through debug initialization and start stepping at the first executable operator, just as usual.

If during the course of stepping through the operator, the debugger reaches a pluggable operator, then that operator is highlighted as the current step operator just like any other operator. If you click **Step** at this point, then the debugger steps through all of the operators contained by the pluggable without changing the graphical context of the map to show the implementation of the pluggable map. If you click **Step Into**, then the graphical context of the map changes to the pluggable map implementation, and the current step operator is set to the first executable source operator inside the pluggable map. The first executable source operator for the pluggable is one of the operators connected from the input signature operator.

You can now step through the pluggable map just as you would any other type of map. When the pluggable operator contains targets, the debugger loads theses just as it does for a top-level map. When the final executable operator is done executing, then the next time you click **Step**, the context changes back to the top level map and begins execution at the next executable operator following the pluggable that was just executed. When the pluggable has no output connections, and it is the final executable operator in the top-level map, then stepping is done.

You can set breakpoints and watch points on operators inside of a pluggable submap. Additionally, during normal editing, you can change the graphical context as you do in normal editing, by clicking **Visit Child Graph** and **Return to Parent Graph**.

Re-Initializing a Debug Session

When you have made changes to the mapping, or have bound source or target operators to different database objects, then you must re-initialize the debug session to

continue debugging the mapping with the new changes. To re-initialize, click the re-initialize button on the toolbar or select the re-initialize menu item in the debug menu. Re-initializing both regenerates and re-deploys the debug code. After re-initialization, the mapping debug session starts from the beginning.

Scalability

Scalability when debugging a mapping applies both to the amount of data that is passed as well as to the number of columns displayed in the Step Data panel. The Define Test Data dialog provides a row limit that you can use to limit the amount of data that flows through the mapping. Also, you can define your own data set by creating your own table and manipulating the records manually.

To restrict the number of columns displayed on the step data window or on a watch tab you can use display sets. By default every operator has a display set ALL and a display set MAPPED to display only the mapped attributes. You can manually add display sets on sources by using the mapping editor directly. Select the use display set option under the right mouse button on an input or output group to select the display set.

7

Designing Process Flows

After you design mappings that define the operations for moving data from sources to targets, you can create and define process flows. Use process flows to interrelate mappings and activities external to Warehouse Builder such as email, FTP commands, and operating system executables.

You can use process flows to manage dependencies between mappings but not to schedule specifically when a given mapping should run. To schedule mappings, process flows, and other executable objects, see Chapter 28, "Scheduling ETL Objects".

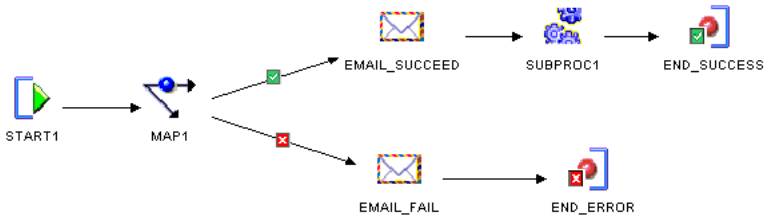
This chapter contains the following topics:

About Process Flows

A process flow describes dependencies between Warehouse Builder mappings and external activities such as email, FTP, and operating system commands.

Each process flow begins with a Start activity and concludes with an End activity for each stream in the flow. You can design a process flow to launch other process flows. Figure 7–1 shows an example of a process flow that launches a mapping MAP1. If the mapping completes successfully, Warehouse Builder sends an email notification EMAIL_SUCCEED and launches another process flow SUBPROC1. If the mapping fails, Warehouse Builder sends an email EMAIL_FAIL and ends the process flow.

Figure 7–1 Sample Process Flow



When you design a process flow in Warehouse Builder, you use an interface known as the Process Flow Editor. Alternatively, you can create and define process flows using the Warehouse Builder scripting language, OMB Scripting Language as described in the Oracle Warehouse Builder API and Scripting Reference.

About Process Flow Modules and Packages

Process flow modules allow you to group process flow packages. Process flow packages, in turn, allow you to group process flows. Together, the process flow modules and packages provide two levels by which to manage and deploy process flows. You can validate, generate, and deploy process flows either at the module or the package level.

You can interrelate process flows contained within the same module. That is, you can design a process flow that launches other process flows so long as they are in the same module. For example, Figure 7–1 depicts a process flow PROC1 that includes process flow SUBPROC1. For PROC1 to run successfully, SUBPROC1 and PROC1 can be in the same or separate packages but must be contained within the same module.

You can copy process flows from one package to another package in the same or a different module and you can copy packages to a different module. To do so, use the Copy and Paste commands available under **Edit** on the Design Center main menu.

Deploying Process Flows to Workflow Engines

Warehouse Builder process flows comply with the XML Process Definition Language (XPDL) standard set forth by the Workflow Management Coalition (WfMC). When you generate a process flow, Warehouse Builder generates an XML file in the XPDL format. You can plug the generated XML file into any workflow engine that follows the XPDL standard.

Warehouse Builder is integrated with Oracle Workflow. From the Warehouse Builder Control Center described in Chapter 29, "Deploying Target Systems", you can deploy process flow packages or modules to Oracle Workflow.

Instructions for Defining Process Flows

To define a process flow, refer to the following sections:

1. Creating Process Flow Modules on page 7-3
2. Creating Process Flow Packages on page 7-3
3. Creating Process Flows on page 7-3
4. Creating and Using Activity Templates on page 7-11
5. Adding Activities on page 7-9
6. Connecting Activities on page 7-15
7. Using Activities in Process Flows on page 22-17
8. Using parameters and variables on page 7-16
9. Configuring Process Flows on page 24-13
10. Validating and Generating Process Flows.
11. Scheduling Process Flows (optional)

    When you are satisfied that the process flow runs as expected, you can schedule the process flow to run on a single or multiple dates as described in Chapter 28, "Scheduling ETL Objects".
12. Deploying Process Flows as described in Chapter 29, "Deploying Target Systems".

Creating Process Flow Modules

Before working with process flows, create a process flow module. The module is a container by which you can validate, generate, and deploy a group of process flows. Process flow modules include process flow packages which include process flows.

To create a process flow module:

1. Right-click the **Process Flow Modules** node in the Project Explorer and select **New.**

   Warehouse Builder displays the Welcome page for the Create Module Wizard.

2. Click **Next.**

   On the Name and Description page, type a module name that is unique within the project. Enter an optional text description.

3. Click **Next.**

   The wizard displays the Connection Information page.

   You can accept the default location that the wizard creates for you based on the module name. Or, select an existing location from the drop down box. Click **Edit** to type in the connection information and test the connection.

4. Click **Next.**

   The wizard displays the Finish page. Verify the name and deployment location of the new process flow module.

   When you click **Finish,** Warehouse Builder stores the definition for the module and inserts its name in the Project Explorer, and prompts you to create a process flow package.

Creating Process Flow Packages

After you create a module for process flows, you can create the process flow package. The process flow package is an additional grouping mechanism from which you can deploy process flows.

To create a process flow package:

1. Right-click a process flow module in the Project Explorer and select **New.**

   Warehouse Builder displays the Create Process Flow Package dialog.

2. Type a name and optional description for the process flow package.

   If you intent to integrate with Oracle Workflow, please note that Oracle Workflow restricts package names to 8 bytes.

3. Select **OK.**

   Warehouse Builder prompts you to create a process flow.

Creating Process Flows

After you create a module and package for process flows, you can create the process flows.

To create a process flow:

1. Right-click a process flow package in the Project Explorer and select **New.**

   Warehouse Builder displays the Create Process Flow dialog.

2. Type a name and optional description for the process flow.

   Note for scheduling process flows: If you intend to schedule the execution of a process flow as described in Chapter 28, "Scheduling ETL Objects", there is an additional consideration. For any ETL object you want to schedule, the limit is 25 characters for physical names and 1995 characters for business names. Follow this additional restriction to enable Warehouse Builder to append to the process flow name the suffix *_job* and other internal characters required for deployment and execution.

3. Select **OK.**

   Warehouse Builder launches the Process Flow and displays the process flow with a Start activity and an End_Success activity.

4. You can now model the process flow with activities and transitions.

5. Continue with the steps listed in "Instructions for Defining Process Flows" on page 7-2.

About the Process Flow Editor

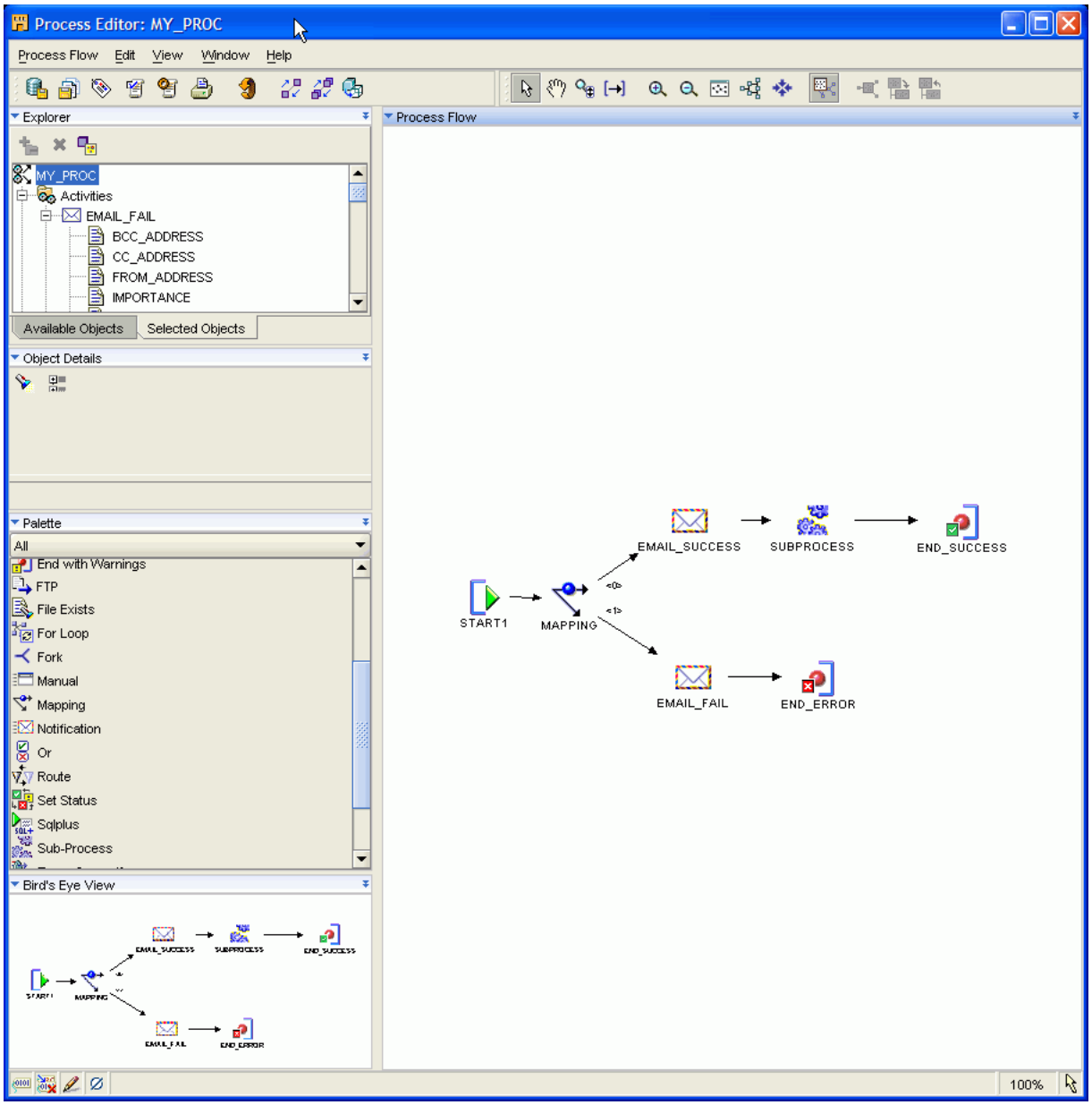
After you create a process flow module and package, use the Process Flow Editor to design and edit process flows. The Process Flow Editor includes a variety of activities that you add and then connect with transitions to design a flow.

Activities represents units of work in a process flow. These units of work can involve components internal or external to Warehouse Builder.

Transitions indicate the sequence and conditions in which to launch activities.

Figure 7–2 shows the Process Flow Editor with a sample process flow. Activities display as icons on the canvas. In this example, the activities include a start, a mapping activity, 2 email activities, a subprocess, and two end activities. Arrows between the activities represent the transitions.

Figure 7–2 Process Flow Editor



Standard Editor Components

The Process Flow Editor has the following standard components common to most editors in Warehouse Builder:

- **Title Bar:** At the top of the editor, the title bar displays the name of the process flow. In Figure 7–2, the name is MY_PROC.
- **Menu Bar:** Below the title bar, the menu bar provides access to the process flow editor commands.
- **Toolbar:** Below the menu bar, the toolbar provides icons for commonly used commands.
- **Canvas:** The canvas provides the work space where you design and modify process flows. When you first create a new process, the Process Flow panel displays with a Start activity and an End activity.

- **Palette:** When you first launch a Process Flow Editor, Warehouse Builder displays the palette along the left side and it contains activity icons that you can drag and drop onto the canvas. You can relocate the palette anywhere on the editor. You can choose to hide or display the palette by clicking on the collapse icon on the palette.
- **Indicator Bar:** In the lower panel, under the Bird's Eye View panel and Canvas panel, you can see mode icons, indicators, and descriptions as shown in Figure 7–3.

Figure 7–3 Indicator Bar on the Process Flow Editor

In the left corner are Naming Mode, Rename Mode, Read/Write, and Validation Mode.

In the right corner are the percent zoom indicator and the navigation mode. In the preceding figure, the zoom level is at 100% and the navigation mode is set to Select Mode.

Process Flow Editor Windows

You can resize windows by placing your mouse on the border of the window, pressing the mouse button when the double sided arrow appears, and dragging your mouse to indicate the desired size.

You can move windows by placing you mouse at the center of its top border, pressing the mouse button when the cross hairs appear, dragging the mouse to the desired location, releasing the mouse button after Warehouse Builder displays the desired shape and location for the window.

To show or hide windows, select **Window** from the menu bar and either activate or deactivate the check mark corresponding to the window.

Explorer

When you first launch the editor, Warehouse Builder displays an explorer for the editor in the upper left corner. The explorer provides a tree listing of all the activities on the canvas and their parameters. When you select an activity on the canvas, Warehouse Builder navigates to the activity on the explorer.

Object Details

When you first launch the editor, Warehouse Builder displays the object details on the left side. This window displays the properties for all activities and their parameters. Select an activity either from the canvas or the explorer and Warehouse Builder displays its properties. From the explorer, if you select an activity parameter, the object details window displays the properties for that parameter. You can edit properties displayed with a white background but not a grey background.

Bird's Eye View

Use the Bird's Eye View window to navigate large and complex process flows.

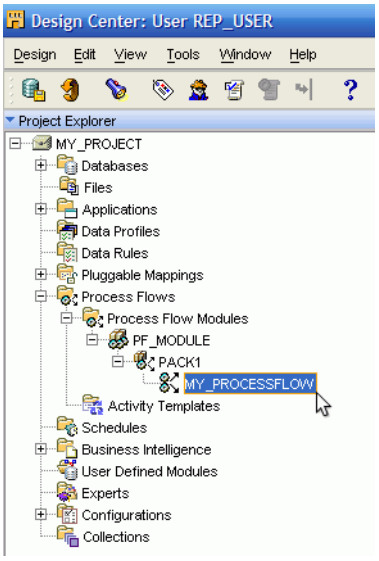
Displaying the Process Flow Editor

To display the Process Flow Editor:

1. From **Process Flows** on the Project Explorer, select a process flow module. If no process flow modules are listed, create a process flow module as described in "Creating Process Flow Modules" on page 7-3.
2. Select a process flow package from a process flow module. If no process flow packages are listed, create a process flow package as described in "Creating Process Flow Packages" on page 7-3.
3. Select a process flow from the Project Explorer as shown in Figure 7–4. If no process flows are listed in the process flow package, right-click the process flow package and select **Create Process Flow.**

   Warehouse Builder prompts you to name the process flow and then launches the editor for you.

Figure 7–4 Process Flow in the Project Explorer



4. To open an existing process flow, double-click the process flow in the Project Explorer.

   Or, select a process flow and then from the **Edit** menu, select **Open Editor.** Or, select a process flow and type **Ctrl** and the letter **O.** Or, right-click a process flow, and select **Open Editor** from the pop-up menu.

   Warehouse Builder displays the Process Flow Editor in the Select mode.

Navigating the Process Flow Editor

The Process Flow Editor includes a variety of tools to assist you in navigating, selecting, and viewing objects on the canvas. Commands you use frequently when designing Process Flows include the following:

Select mode

Use the select mode to select objects on the canvas. When you select an activity, the editor displays a blue border around the activity and you can edit, move, or delete the activity.

You can edit the activity using the object details window in conjunction with the Available Objects tab in the editor explorer window. When you select a transition, the editor changes the arrow from black to blue. Edit the transition in the object details.

To activate the Select mode, click the icon in the toolbar or select **Edit** and **Select Mode** from the menu.

Edge Navigation

Edge navigation assists you in navigating complex designs on the canvas. Select the icon from the menu bar and then select a activity on the canvas. When you release the mouse button, Warehouse Builder navigates to the next activity in the flow and moves the focus to that activity. To navigate backward through a flow, select the edge navigation icon and then select the last activity in the flow.

> **Note:** If you used the Process Flow Editor in Warehouse Builder 10*g* Release 1 and previous releases, you may recognize the icon for edge navigation in this release is the same as was used for the create transition icon. In the Process Flow Editor and the Expert Editor, the create transition icon is no longer required or available.

For navigating and displaying complex designs in the editor, you may find the following tools useful:

- Pan
- Interactive zoom
- Zoom in
- Zoom out
- Fit in Window
- Auto layout
- Center
- Expand Child Graph
- Visit Child Graph
- Return to Parent Graph

About Activities

The basic design elements for process flows include activities and transitions. Activities represent units of work for the process flow such launching a mapping or verifying the existence of a file on a drive or directory. When you design a process flow in Warehouse Builder, you select activities from the editor palette, drag them onto the canvas, and set their parameters. Warehouse Builder includes the following types of activities:

- Oracle Warehouse Builder Specific Activities: These activities enable you to launch Warehouse Builder objects such as mappings, transformations or other process flows. The process flow executes the object and issues a commit statement.
- Utility Activities: These activities enable you to perform services such as send emails and transfer files.
- Control Activities: These activities enable you to control the progress and direction of the process flow. For instance, use the Fork activity to launch multiple activities concurrently.

For the utility and control type activities, you can reuse their parameters by defining activity templates as described in "Creating and Using Activity Templates" on

page 7-11. For emails for example, use an email template to specify the SMTP server name and port number, the list of addresses, and the priority. Then you can reuse that template when you add email activities to a process flow.

For a description of each activity, see "Using Activities in Process Flows" on page 22-1.

Adding Activities

To add an activity to a process flow:

1. View the activities listed in the palette located along the left side of the editor.

   By default, the palette lists all activities. To find a particular activity, use the list box on the palette to narrow the displayed list to one of the following types of activities: Oracle Warehouse Builder Specific Activities, Utility Activities, and Control Activities.

2. Select an activity from the palette and drag it onto the canvas.

   The editor displays the activity on the canvas with the name highlighted in blue.

3. To accept the default name, press **Enter.** To change the name, type in the new name.

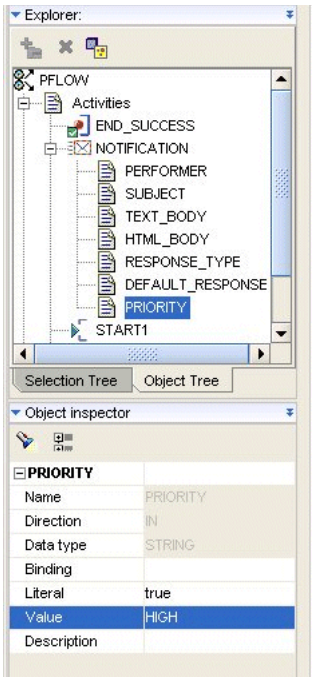
   The editor lists the activity on the explorer window located at the left side of the editor and in the object details window along the left side.

4. In **Object Details,** enter the parameters for the activity.

   These parameters vary according to the type of activity. Figure 7–5, for example, shows the parameters for a notification activity which includes performer, subject, text_body, priority, and so on.

   For each parameter, Warehouse Builder defines a read only Name, Direction, and Data Type. And for each parameter, you can specify values for Binding, Literal, Value, and Description.

Figure 7–5 The Parameters for a Notification Activity



Parameters for Activities

Each parameters has the following properties:

Name

The name of the activity parameter, such as PRIORITY in Figure 7–5. For information about a specific parameter, look up the activity by name under "Using Activities in Process Flows" on page 22-1.

Direction

The parameter direction is read-only for parameters that are not created by the user. A direction of IN indicates that the parameter is an input parameter for the activity.

Data Type

The parameter data type is read-only for parameters that are not created by the user. Warehouse Builder assigns the appropriate data type for all default parameters.

Binding

Use binding to pass in parameters from outside the process flow for parameters that are not created by the user. If you assign a parameter in **Binding,** it overrides any text you assign to **Value.**

Literal

If you type in a value for the parameter in the field **Value,** indicate whether the value is literal or an expression. The literal data types follow the PL/SQL literal value specification except for calendar data types. These data types are represented in a standard format as the Process Flow interacts with data sources from different locations. Table 7–1 provides the Literal Value Type, Format, and Example.

Table 7–1 Example of Literal Value Types

Literal Value Type	Format	Example
DATE	YYYY-MM-DD	2006-03-21
DATE	YYYY-MM-DD HH24:MI:SS	2006-03-21 15:45:00
TIMESTAMP	YYYY-MM-DD HH24:MI:SS.FF9	2006-03-21 15:45:00.000000000
TIMESTAMP_TZ	YYYY-MM-DD HH24:MI:SS.FF9 TZH:TZM	2006-03-21 15:45:00.000000000 +01:00
YMINTERVAL	[+-]YYYYYYYYY-MM	+000000001-01
DMINVERVAL	[+-]DDDDDDDDD HH24:MI.SS.FF9	+000000001 01:01:01.000000001

Value

This is the value of the parameter. For some parameters such as PRIORITY in Figure 7–5, Warehouse Builder enables you to select from a list of values. For other parameters, Warehouse Builder assigns default values which you can override by typing in a new value or using the field **Binding.** In the absence of a list of possible values or a default value, you must type in a value.

Description

You can type an optional description for each property.

Creating and Using Activity Templates

In designing process flows you may want to reuse existing activities. For example, each time a mapping fails in a process flow, you may want to send an email to the same group of administrators. You create a template for the email activity once and then use and edit the activity in many process flows.

To create an activity template:

1. In the Project Explorer, navigate to the Activity Templates node under the Process Flows node.
2. To create a folder for containing templates, right-click the Activity Templates node and select **New**.
3. Assign a name for the folder.

   Consider creating a folder for each type of template you plan to create. For instance, you could create separate folders to contain email and ftp templates.
4. The Create Activity Template Wizard will appear.

   Note: If the wizard does not appear automatically, then right-click a folder and select **New**.

   Follow the prompts in the Create Activity Template Wizard to complete the Name and Description Page, the Parameters Page, and the wizard summary page.
5. See "Using Activity Templates" on page 7-13 for instructions on how to use the template in a process flow.

Name and Description Page

The rules for naming objects in the Activity Template depend on the naming mode you select in "Naming Preferences" on page 3-8. Warehouse Builder maintains a business and a physical name for each object in the repository. The business name is its descriptive business name. The physical name is the name Warehouse Builder uses when generating code.

When you name objects while working in one naming mode, Warehouse Builder creates a default name for the other mode. Therefore, when working in the business name mode, if you assign a activity template name that includes mixed cases, special characters and spaces, Warehouse Builder creates a default physical name for you.

Assign a name and select the type of activity template you want to create. Also, write an optional description for the template.

Naming Activities

In the physical naming mode, a Activity name can be from 1 to 30 alphanumeric characters and blank spaces are not allowed. In the business naming mode, the limit is 2000 characters and blank spaces and special characters are allowed. In both naming modes, the name should be unique across the project.

Describing Activities

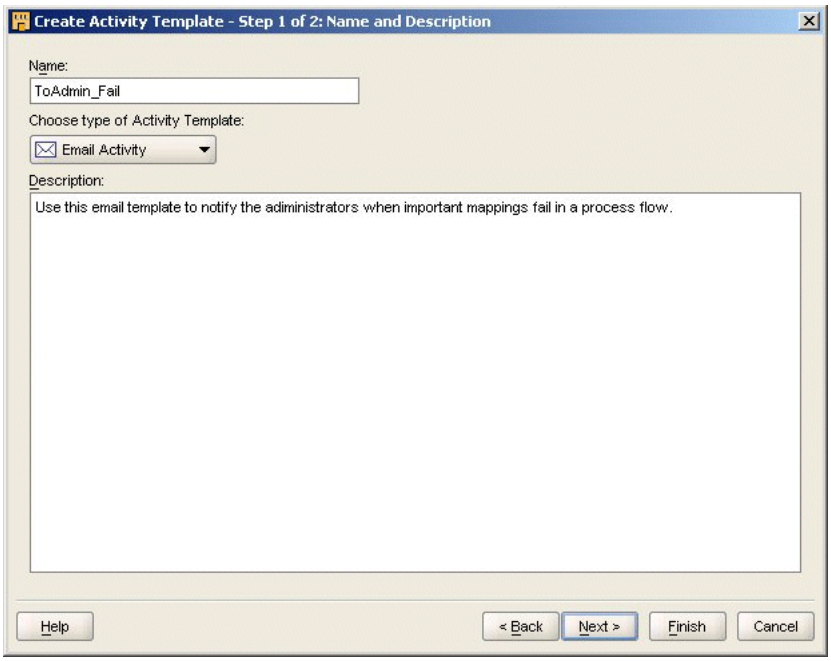
The description can be between 2 and 2,000 alphanumeric characters and can contain blank spaces. Specifying a description for a activity template is optional.

The following activity templates are available from the drop down menu.

- Assign
- Email
- FTP
- File Exists
- Manual
- Notification
- Set Status
- Sqlplus
- User Defined
- Wait

Figure 7–6 shows an example of the Name and Description page for an email activity template.

Figure 7–6 Describing an Email Activity Template



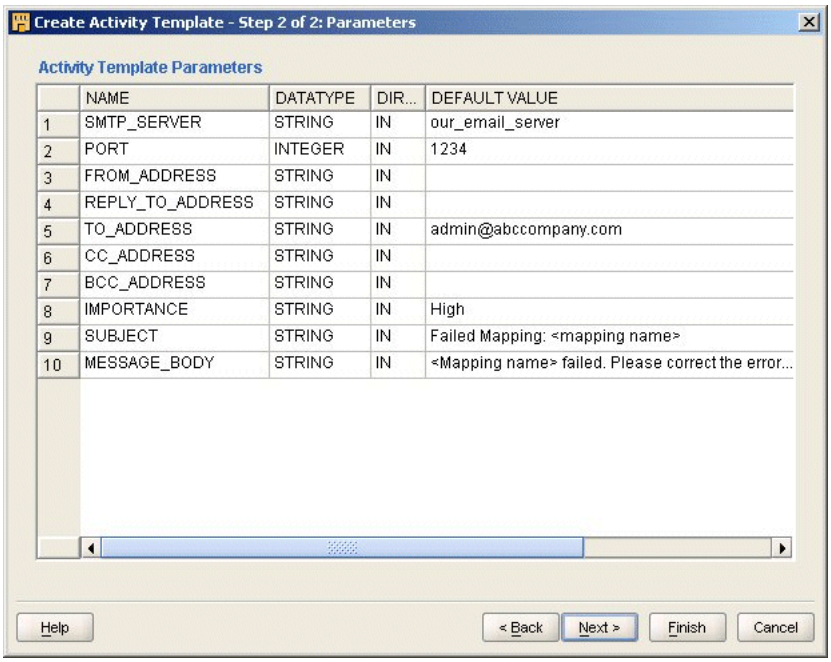
Parameters Page

The wizard displays parameters based on the type of activity you previously selected: Assign, Manual, Email, File Exists, Data Auditor, FTP, Notification, Set Status, Sqlplus, User Defined, While Loop, Wait, or Notification activity.

Enter default values for the activity. When you use the activity template in a process flow, you can accept or edit the default values. In Figure 7–7, for example, you could

edit the default values for the email subject and message body to contain the name of the mapping.

Figure 7–7 Parameters Page for Email Activity Template

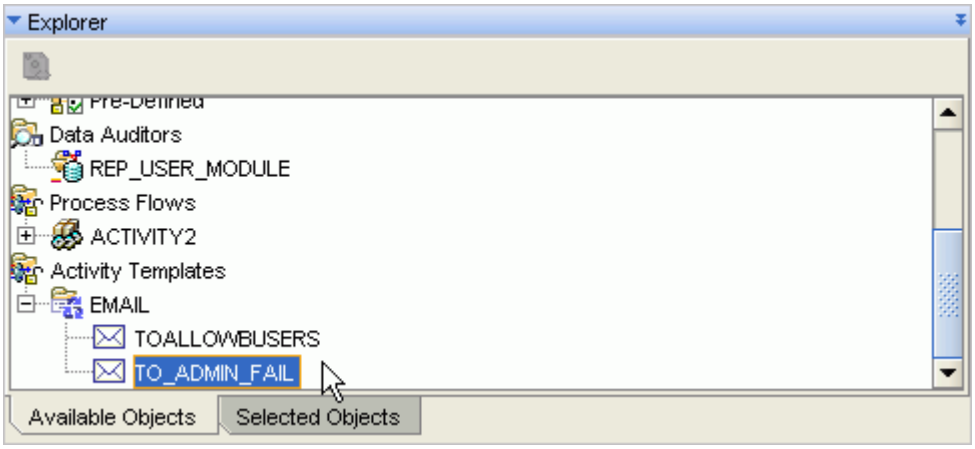


Using Activity Templates

Complete the following steps to use an activity template:

1. In the Project Explorer, navigate to the process flow module under the Process Flows node.
2. To open the **Process Editor**, right-click the Process Flow module and select Open Editor.
3. In the **Process Editor**, click the **Available Objects** tab in the explorer window and expand the **Activity Templates** as shown in Table 7–8.

Figure 7–8 Explorer Window with an Activity Template selected

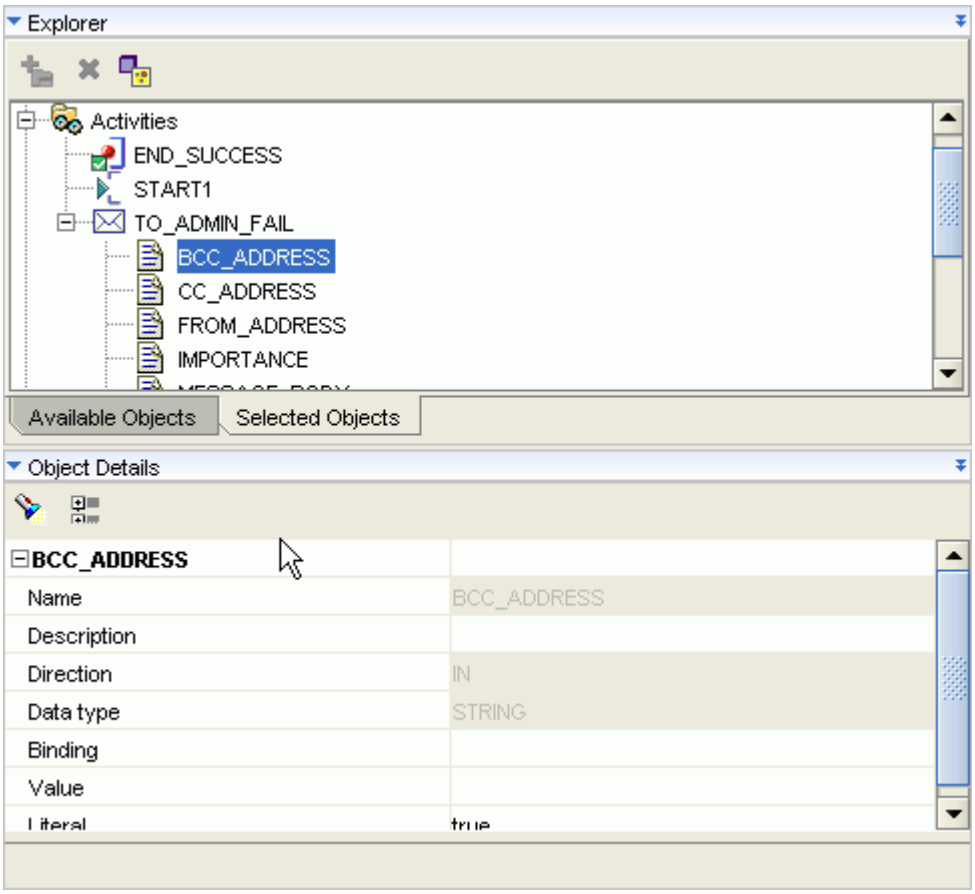


4. Drag and drop the activity template onto the canvas.

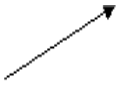
Activity templates in a process flow behave like regular activities.

5. To edit the activity, be sure to click the Selected Objects tab in the explorer window and then edit the activity in the object details window as shown in Figure 7–9.

Figure 7–9 Editing an Activity Template



About Transitions



Use transitions to indicate the sequence and conditions in which activities occur in the process flow. You can use transitions to execute an activity based on the completion state of the preceding activity.

When you add a transition to the canvas, by default, the transition has no condition applied to it. The process flow continues once the preceding activity completes, regardless of the ending state of the previous activity.

A transition with no condition applied to it has different semantics depending on the source activity type. If the activity type is FORK, then it may have multiple unconditional transitions in which each transition begins a new flow in the process flow. If the source activity type is not a FORK, then there may be only one unconditional transition and it is used when no other conditional transition is activated, like the final "ELSE" in an "IF...THEN...ELSIF...ELSE...END" PL/SQL statement.

Rules for Valid Transitions

For a transition to be valid, it must conform to the following rules:

- All activities, apart from START and END, must have at least one incoming transition.
- Only the AND and OR activities can have more than one incoming transition.
- Only a FORK activity can have more than one unconditional outgoing transition.
- A FORK activity can have only unconditional outgoing transitions
- An activity that has an enumerated set of outcomes must have either an outgoing transition for each possible outcome or an unconditional outgoing transition.
- An activity can have zero or more outgoing complex expression transitions.
- An activity with an outgoing complex expression transition must have an unconditional outgoing transition to act as an otherwise.
- An END_LOOP transition must have only one unconditional transition to its associated FOR_LOOP or WHILE_LOOP activity.
- The transition taken by the "exit" outcome of a FOR_LOOP or WHILE_LOOP must not connect to an activity that could be executed as a result of the "loop" outcome.

Connecting Activities

To create dependencies using transitions:

1. When working in the select mode, place your mouse along the right border of the activity icon at its center line.

   The editor displays the cursor as a small horizontal arrow, indicating that you can now use the mouse button to connect activities.

2. Press the left mouse button and scroll towards the next activity. As you begin to scroll, the cursor appears as an arrow with a plus sign beneath it. Continue to scroll towards the next activity until the plus sign under the cursor arrow changes to a circle. Release the mouse button to connect the two activities.

   The editor displays an arrow between the two activities, assigns a default name to the transition, and displays the transition in the explorer and object selector windows.

3. In the object selector window, view or edit the following attributes:

   Name: The editor assigns a default name which you can change.

   Description: You can type an optional description for the transition.

   Condition: Transitions you initially draw on the canvas are unconditional by default. To override the default and apply conditions, click the button in the Condition as described in "Defining Transition Conditions". If you select a condition, the editor displays the associated icon imposed onto the transition line on the canvas.

   Source: This property is read-only and indicates the first activity in the connection.

   Target: This property is read-only and indicates the second activity in the connection.

Configuring Activities

Some activities such as Sqlplus on page 22-17 require additional configuration. These configuration details for a given activity are listed in Chapter 22, "Using Activities in Process Flows".

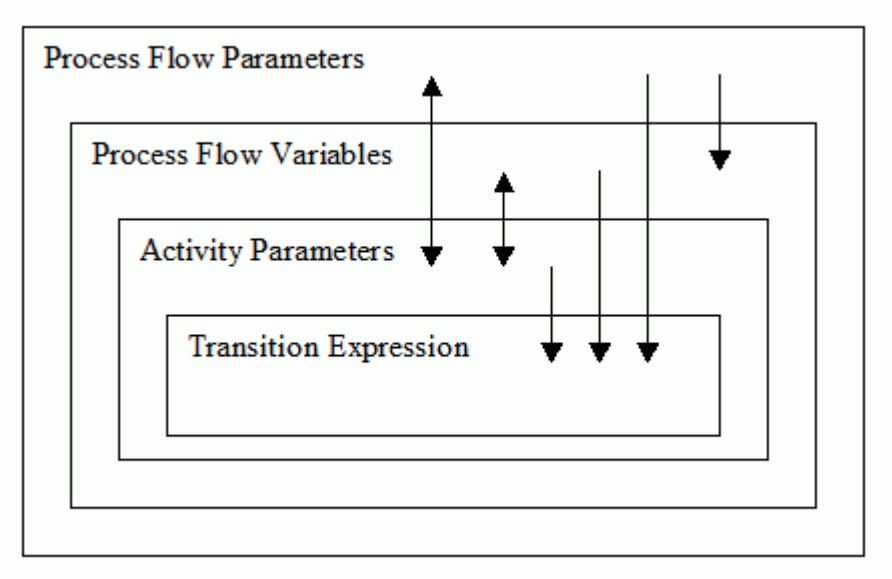
Using parameters and variables

Process Flows and activities support the PL/SQL parameter passing concept, allowing data to be passed and so encouraging reuse through parameterization. This is accomplished through data stores, which are implemented as either parameters or variables.

- Parameters allow the passing of data between a Process Flow and its activities or subprocesses.
- Variables allow the storage of transient data, which is then maintained for the lifetime of the Process Flow execution. Variables are the main mechanism by which data can be passed between activities.

Process Flows implement a Scope hierarchy which allows the data to be passed between data stores. Figure 7–10 shows the direction in which the data is passed.

Figure 7–10 Relationship between the scope and the direction in which the data is passed



Process Flows follows the following rules for a scope hierarchy:

1. Process Flow variables can be initialized from flow parameters, but the reverse is not allowed
2. Activity parameters can pass data bi-directionally between process flow variables and process flow parameters.
3. Transition expressions can be evaluated against their source activity parameters and process flow parameters and process flow variables.
4. A data store cannot be accessed from another data store within the same scope.

Using Namespace

The namespace allows a data store of an inner scope to hide the data store of an outer scope, similar to PL/SQL. By qualifying the data store name with the process flow name or activity, the hidden data store name can be referenced. For Example:

My_PROC.VAR1

The namespace does not allow referencing of data from another data store within the same scope.

Using Bindings

A data store may be bound to another data store in an outer scope, which supports the passing of data in both directions.

Process Flow bindings follow the same semantics as PL/SQL with the following rules:

1. All the data is passed within the Process Flow by value.
2. Variables can be initialized through a binding; they cannot return a value back.
3. An INOUT parameter can be bound to an IN parameter in an outer scope. The output value, which is passed by value, is audited and then discarded.

As you can see in Figure 7–10, a variable may not pass data out to a Process Flow parameter, therefore this is accomplished by the use of an Assign operator, which can be bound to the variable and the parameter.

Expressions

Oracle Warehouse Builder supports the evaluation of PL/SQL expressions for the derivation of parameters values and the evaluation of 'complex expression' transitions.

The expression must yield a correctly typed value for data store. Automatic conversion from VARCHAR is supported. Where the expression is associated with a transition then a BOOLEAN result is expected.

During evaluation, an expression will have access to the outer scopes which enclose it, therefore an expression for an activity's parameter will be able to use process flow variables and process flow parameters in its evaluation.

Global Expression Values

Warehouse Builder also makes additional data values available to the expression from the current activity and the owning Process Flow. Table 7–2 lists these values.

Table 7–2 Global Expression Values

Identifier	Type	Description
NUMBER_OF_ERRORS	NUMBER	Number of errors reported on completion of activity's execution.
NUMBER_OF_WARNINGS	NUMBER	Number of warnings reported on completion of activity's execution.
RETURN_RESULT	VARCHAR2(64)	Textual representation of result. For example, 'SUCCESS', 'WARNING', 'ERROR'.
RETURN_RESULT_NUMBER	NUMBER	Enumeration of RESULT_RESULT1 = SUCCESS2 = WARNING3 = ERROR.

Table 7–2 (Cont.) Global Expression Values

Identifier	Type	Description
RETURN_CODE	NUMBER	Integer 0-255, specific to activity, synonymous with an Operating System return code.
PARENT_AUDIT_ID	NUMBER	The audit id of the calling Process Flow.
AUDIT_ID	NUMBER	The audit id of the activity.

The following additional constants are also provided:

Identifier	Type	Description
SUCCESS	NUMBER	SUCCESS enumerated value.
WARNING	NUMBER	WARNING enumerated value.
ERROR	NUMBER	ERROR enumerated value.

Defining Transition Conditions

Use the Transition Editor to specify one of the enumerated conditions or write an expression for a complex condition. The enumerated conditions include success, warning, and error and display on the canvas as shown in Table 7–3.

Table 7–3 Types of Conditions for Transitions

Icon	Transition	Description
	Success	The process flow continues only if the preceding activity ends in success.
	Warning	The process flow continues only if the preceding activity ends with warnings.
	Error	The process flow continues only if the preceding activity ends in error.
	Warning	The process flow continues only if the preceding activity ends with warnings.
	Complex	The process flow continues only if the preceding activity returns a value that meets the criteria you specify in an expression.
	Extended	The process flow continues only if the proceeding notification activity ends with an extended result.

Extended transition is only valid for Notification activities because they are the only activity which return an extended result. The activity acquires this icon when set to an outcome of #MAIL, #NOMATCH, #TIE, #TIMEOUT. Table 7–4 list the output and the description of the Extended transition.

Table 7–4 Output and Description of the Extended Transition

Output	Description
#NOMATCH	Result of a voting notification where no candidate acquired the minimum number of votes to win
#TIE	Result of a voting notification where the result was a tie.

Table 7–4 (Cont.) Output and Description of the Extended Transition

Output	Description
#MAIL	A mail error occurred for the notification. Some recipients did not receive it and so it was cancelled.
#TIMEOUT	The notification did not receive a response within the configured amount of time.

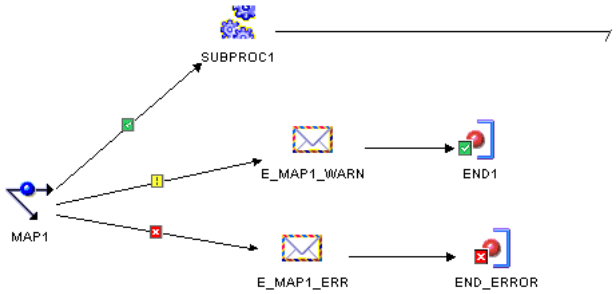
If the activity has only one outgoing activity, you can specify any of the conditions listed in Table 7–3 or leave the transition unconditional.

The rules for using multiple outgoing transitions depend on the type of activity. The general rule is that you can use an unlimited number of complex conditions in addition to one of each of the following: SUCCESS, WARNING, ERROR, and UNCONDITIONAL. The exception to this rule comes when you use control activities such as AND, FORK, and OR.

When you add multiple outgoing transitions from an activity, ensure that the conditions do not conflict. A conflict occurs when the process flow logic can evaluate to more than one outgoing transition being true.

Figure 7–11 shows a portion of a process flow in which different activities are triggered based on the three possible completion states of MAP1. Since only one of these conditions can be satisfied at a time, there is no conflict. If you attempt to add an unconditional transition or another conditional transition, two transition conditions would be true and the process flow would be invalid.

Figure 7–11 Outgoing Transition Conditions



Evaluation Security Context

The PL/SQL expression is executed in the context of the Control Center user that requested the execution of the activity. However in the case where the Oracle Workflow schema is hosted in a remote database instance then the effective user of the generate database link with be used instead. A different Control Center user may be selected but configuring the Process Flow and specifying an 'Evaluation Location'. Therefore the expression may reference any PL/SQL function that is accessible to the Control Center user.

8

Understanding Performance and Advanced ETL Concepts

Use this chapter as a guide for creating ETL logic that meets your performance expectations.

This chapter includes the following topics:

Best Practices for Designing PL/SQL Mappings

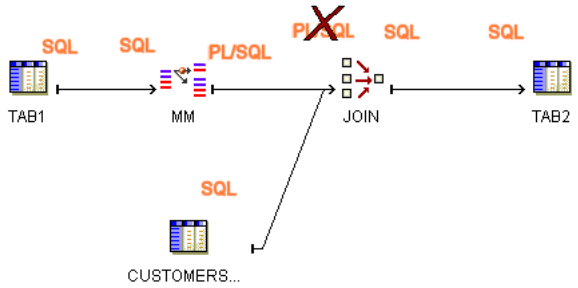
This section addresses PL/SQL mapping design and includes:

Warehouse Builder generates code for PL/SQL mappings that meet the following criteria:

- The output code of each operator satisfies the input code requirement of its next downstream operator.
- If the mapping contains an operator that generates only PL/SQL output, all downstream dataflow operators must also be implemented by PL/SQL. You can use SQL operators in such a mapping only after loading the PL/SQL output to a target.

As you design a mapping, you can evaluate its validity by taking note of the input and output code types for each operator in the mapping. For example, you can see that the mapping in Figure 8–1 is invalid because the Match-Merge operator MM generates PL/SQL output but the subsequent Join operator accepts SQL input only.

Figure 8–1 Mapping Violates Input Requirement for Join Operator



To achieve the desired results for the mapping, consider joining the source tables before performing the Match-Merge, as shown in Figure 8–2, or loading the results from the Match-Merge to a staging table before performing the join, as shown in Figure 8–3.

Figure 8–2 Valid Mapping Design with Sources Joined Before Match-Merge

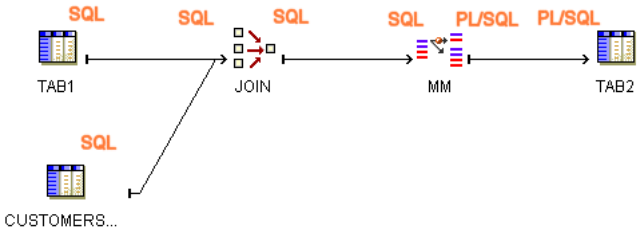


Figure 8–3 Valid Mapping Design with Staging Table

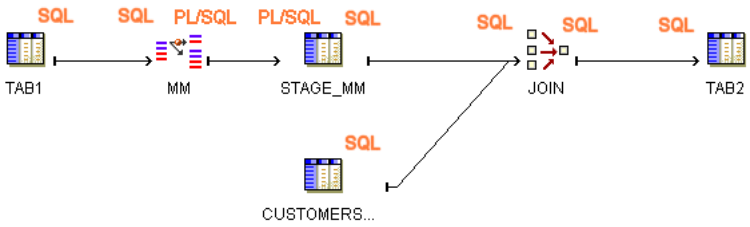


Table 8–1 and Table 8–2 list the implementation types for each Warehouse Builder operator. These tables also indicate whether or not PL/SQL code includes the operation associated with the operator in the cursor. This information is relevant in determining which operating modes are valid for a given mapping design. It also determines what auditing details are available during error handling.

Table 8–1 Source-Target Operators Implementation in PL/SQL Mappings

Operator	Implementation Types	Valid in Set Based Mode	Valid in Row Based Mode	Valid in Row Based (Target Only)
Source Operators: Tables, Cubes, Views, External Tables.	SQL	Yes.	Yes.	Yes. Part of cursor.
Target Operators: Tables, Cubes, Views,	SQL PL/SQL	Yes, except when loading = DELETE or loading= UPDATE and database is not 10*g* or higher.	Yes.	Yes. Not part of cursor.
Flat File as source	For PL/SQL, create External Table.	Yes.	Yes.	Yes. Part of the cursor.
Flat File as target	SQL	Yes, except when loading = DELETE or loading= UPDATE and database is not 10*g* or higher.	Yes.	Yes. Not part of cursor.
Advanced Queue as source	SQL	Yes.	Yes.	Yes, part of cursor.
Advanced Queue as target	SQL	Yes, except when loading = DELETE or loading= UPDATE and database is not 10*g* or higher.	Yes.	Yes. Not part of cursor
Sequence as source	SQL	Yes.	Yes.	Yes, part of cursor.

Table 8–2 Data Flow Operator Implementation in PL/SQL Mappings

Operator Name	Implementation Types	Valid in Set Based Mode	Valid in Row Based Mode	Valid in Row Based (Target Only) Mode
Aggregators	SQL	Yes.	Yes, only if part of the cursor.	Yes, only if part of the cursor.
Constant Operator	PL/SQL SQL	Yes.	Yes.	Yes.
Data Generator	SQL*Loader Only	N/A	N/A	N/A
Deduplicator	SQL	Yes.	Yes, only if part of the cursor.	Yes, only if part of the cursor.
Expression	SQL PL/SQL	Yes.	Yes.	Yes.
Filter	SQL PL/SQL	Yes.	Yes.	Yes.
Joiner	SQL	Yes.	Yes, only if part of the cursor.	Yes, only if part of the cursor.
Key Lookup	SQL	Yes.	Yes, only if part of the cursor.	Yes, only if part of the cursor.
Mapping Input Parameter	SQL PL/SQL	Yes.	Yes.	Yes.
Mapping Output Parameter	SQL PL/SQL	Yes.	Yes.	Yes.

Table 8–2 (Cont.) Data Flow Operator Implementation in PL/SQL Mappings

Operator Name	Implementation Types	Valid in Set Based Mode	Valid in Row Based Mode	Valid in Row Based (Target Only) Mode
Match-Merge	SQL input PL/SQL output (PL/SQL input from XREF group only)	No.	Yes.	Yes. Not part of cursor.
Name-Address	PL/SQL	No.	Yes.	Yes. Not part of cursor.
Pivot	SQL PL/SQL	Yes.	Yes.	Yes.
Post-Mapping Process	Irrelevant	Yes, independent of dataflow.	Yes.	Yes.
Pre-Mapping Process	Irrelevant	Yes, independent of dataflow.	Yes.	Yes.
Set	SQL	Yes.	Yes, only if part of the cursor.	Yes, only if part of the cursor.
Sorter	SQL	Yes.	Yes, only if part of the cursor.	Yes, as part of the cursor.
Splitter	SQL PL/SQL	Yes.	Yes.	Yes.
Table Function	SQL or PL/SQL input SQL output only	Yes.	Yes.	Yes.
Transformation as a procedure	PL/SQL	No.	Yes.	Yes. Not part of cursor.
Transformation as a function that does not perform DML	SQL PL/SQL	Yes.	Yes.	Yes, included in the cursor.

Set Based Versus Row Based Operating Modes

For mappings with a PL/SQL implementation, select one of the following operating modes:

- Set based
- Row based
- Row based (Target Only)
- Set based fail over to row based
- Set based fail over to row based (target only)

The default operating mode you select depends upon the performance you expect, the amount of auditing data you require, and how you design the mapping. Mappings have at least one and as many as three valid operating modes, excluding the options for failing over to row based modes. During code generation, Warehouse Builder generates code for the specified default operating mode as well as the deselected modes. Therefore, at runtime, you can select to run in the default operating mode or any one of the other valid operating modes.

The types of operators in the mapping may limit the operating modes you can select. As a general rule, mappings run in set based mode can include any of the operators except for Match-Merge, Name-Address, and Transformations used as procedures. Although you can include any of the operators in row based and row based (target only) modes, there are important restrictions on how you use SQL based operators such as Aggregators, Joins, and Key Lookups. To use SQL based operators in either of the row based modes, ensure that the operation associated with the operator can be included in the cursor.

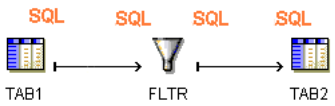
These general rules are explained in the following sections.

Set based

In set based mode, Warehouse Builder generates a single SQL statement that processes all data and performs all operations. Although processing data as a set improves performance, the auditing information available is limited. Runtime Auditing is limited to reporting to the execution error only. In set based mode, you cannot view details on which rows contain errors.

Figure 8–4 shows a simple mapping and the associated logic Warehouse Builder uses to generate code for the mapping when run in set based operating mode. TAB1, FLTR, and TAB2 are processed as a set using SQL.

Figure 8–4 Simple Mapping Run in Set Based Mode



To correctly design a mapping for the set based mode, avoid operators that require row by row processing such as Match-Merge and Name-Address operators. If you include an operator in the dataflow that cannot be performed in SQL, Warehouse Builder does not generate set based code and issues an error when you execute the package in set based mode.

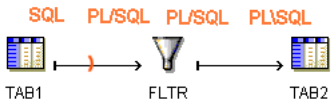
For target operators in a mapping, the loading types INSERT/UPDATE and UPDATE/INSERT are always valid for set based mode. Warehouse Builder support s UPDATE loading in set based mode only when the Oracle Database is 10*g* or higher. Warehouse Builder does not support DELETE loading in set based mode. For a complete listing of how Warehouse Builder handles operators in set based mappings, see Table 8–2 on page 8-3.

Row based

In row based mode, Warehouse Builder generates statements that process data row by row. The select statement is in a SQL cursor. All subsequent statements are PL/SQL. You can access full runtime auditing information for all operators performed in PL/SQL and only limited information for operations performed in the cursor.

Table 8–5 shows a simple mapping and the associated logic Warehouse Builder uses to generate code for the mapping when run in row based operating mode. TAB1 is included in the cursor and processed as a set using SQL. FLTR and TAB2 are processed row by row using PL/SQL.

Figure 8–5 Simple Mapping Run in Row Based Mode



If the mapping includes any SQL based operators, which cannot be performed in PL/SQL, Warehouse Builder attempts to generate code with those operations in the cursor. To generate valid row based code, design your mapping such that, if you include any of the following SQL based operators, Warehouse Builder can include the operations in the cursor:

- Aggregation
- Deduplicator
- Join
- Key Lookup
- Sequence
- Set
- Sorter

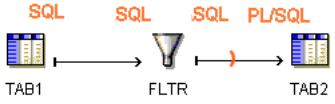
For the preceding operators to be included in the cursor, do not directly precede it by an operator that generates PL/SQL code. In other words, you cannot run the mapping in row-based mode if it contains a Transformation implemented as procedure, a Flat File used as a source, a Match-Merge, or Name-Address operator directly followed by any of the seven SQL based operators. For the design to be valid, include a staging table between the PL/SQL generating operator and the SQL based operator.

Row based (Target Only)

In row based (Target Only) mode, Warehouse Builder generates a cursor select statement and attempts to include as many operations as possible in the cursor. For each target, Warehouse Builder inserts each row into the target separately. You can access full runtime auditing information for all operators performed in PL/SQL and only limited information for operations performed in the cursor. Use this mode when you expect fast set based operations to extract and transform the data but need extended auditing for the loading the data, which is where errors are likely to occur.

Table 8–6 shows a simple mapping and the associated logic Warehouse Builder uses to generate code for the mapping when run in row based (target only) operating mode. TAB1 and FLTR are included in the cursor and processed as a set using SQL. TAB2 is processed row by row.

Figure 8–6 Simple Mapping Run in Row Based (Target Only) Mode



Row based (target only) places the same restrictions on SQL based operators as the row based operating mode. Additionally, for mappings with multiple targets, Warehouse Builder generates code with a cursor for each target.

About Committing Data in Warehouse Builder

There are two major approaches to committing data in Warehouse Builder. You can commit or rollback data based on the mapping design. To do this, use one of the commit control methods described in "Committing Data Based on Mapping Design" on page 8-7.

Alternatively, for PL/SQL mappings, you can commit or rollback data independently of the mapping design. Use a process flow to commit the data or establish your own method as described in "Committing Data Independently of Mapping Design" on page 8-10.

Committing Data Based on Mapping Design

By default, Warehouse Builder loads and then automatically commits data based on the mapping design. For PL/SQL mappings you can override the default setting and control when and how Warehouse Builder commits data. You have the following options for committing data in mappings:

Automatic: This is the default setting and is valid for all mapping types. Warehouse Builder loads and then automatically commits data based on the mapping design. If the mapping has multiple targets, Warehouse Builder commits and rolls back each target separately and independently of other targets. Use the automatic commit when the consequences of multiple targets being loaded unequally are not great or are irrelevant.

Automatic Correlated: Automatic correlated commit is a specialized type of automatic commit that applies to PL/SQL mappings with multiple targets only. Warehouse Builder considers all targets collectively and commits or rolls back data uniformly across all targets. Use the correlated commit when it is important to ensure that every row in the source impacts all affected targets uniformly. For more information about correlated commit, see "Committing Data from a Single Source to Multiple Targets" on page 8-7.

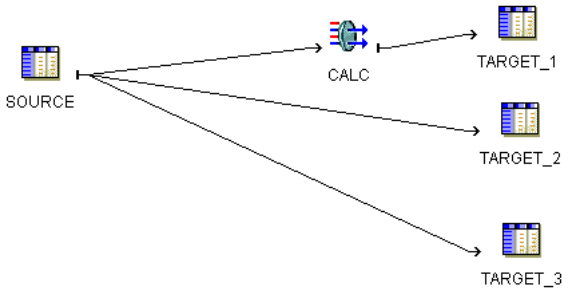
Manual: Select manual commit control for PL/SQL mappings that you want to interject complex business logic, perform validations, or run other mappings before committing data. For examples, see "Embedding Commit Logic into the Mapping" on page 8-9 and "Committing Data Independently of Mapping Design" on page 8-10.

Committing Data from a Single Source to Multiple Targets

If you want to populate multiple targets based on a common source, you may also want to ensure that every row from the source impacts all affected targets uniformly.

Figure 8–7 shows a PL/SQL mapping that illustrates this case. The target tables all depend upon the source table. If a row from SOURCE causes changes in multiple targets, for instance TARGET_1 and TARGET_2, then Warehouse Builder should commit the appropriate data to both effected targets at the same time. If this relationship is not maintained when you run the mapping again, the data can becomes inaccurate and possibly unusable.

Figure 8–7 Mapping with Multiple Targets Dependent on One Source



If the number of rows from SOURCE is relatively small, maintaining the three targets may not be difficult. Manually maintaining targets dependent on a common source, however, becomes more tedious as you increase the number of rows from the source, or as you design more complex mappings with more targets and transformations.

To ensure that every row in the source properly impacts every target, configure the mapping to use the correlated commit strategy.

Using the Automatic Correlated Commit Strategy

In set based mode, correlated commit may impact the size of your rollback segments. Space for rollback segments may be a concern when you merge data (insert/update or updated/insert).

Correlated commit operates transparently with PL/SQL bulk processing code.

The correlated commit strategy is not available for mappings run in any mode that are configured for Partition Exchange Loading or include an Advanced Queue, Match-Merge, or Table Function operator.

Automatic Commit versus Automatic Correlated Commit

The combination of the commit strategy and operating mode determines mapping behavior. Table 8–3 shows the valid combinations you can select.

Table 8–3 Valid Commit Strategies for Operating Modes

Operating Mode	Automatic Correlated Commit	Automatic Commit
Set based	Valid	Valid
Row based	Valid	Valid
Row based (target only)	Not Applicable	Valid

Correlated commit is not applicable for row based (target only). By definition, this operating mode places the cursor as close to the target as possible. In most cases, this results in only one target for each select statement and negates the purpose of committing data to multiple targets. If you design a mapping with the row based (target only) and correlated commit combination, Warehouse Builder runs the mapping but does not perform the correlated commit.

To understand the effects each operating mode and commit strategy combination has on a mapping, consider the mapping from Figure 8–7 on page 8-8. Assume the data from source table equates to 1,000 new rows. When the mapping runs successfully, Warehouse Builder loads 1,000 rows to each of the targets. If the mapping fails to load the 100th new row to Target_2, you can expect the following results, ignoring the

influence from other configuration settings such as Commit Frequency and Number of Maximum Errors:

- **Set based/ Correlated Commit:** A single error anywhere in the mapping triggers the rollback of all data. When Warehouse Builder encounters the error inserting into Target_2, it reports an error for the table and does not load the row. Warehouse Builder rolls back all the rows inserted into Target_1 and does not attempt to load rows to Target_3. No rows are added to any of the target tables. For error details, Warehouse Builder reports only that it encountered an error loading to Target_2.
- **Row based/ Correlated Commit:** Beginning with the first row, Warehouse Builder evaluates each row separately and loads it to all three targets. Loading continues in this way until Warehouse Builder encounters an error loading row 100 to Target_2. Warehouse Builder reports the error and does not load the row. It rolls back the row 100 previously inserted into Target_1 and does not attempt to load row 100 to Target_3. Next, Warehouse Builder continues loading the remaining rows, resuming with loading row 101 to Target_1. Assuming Warehouse Builder encounters no other errors, the mapping completes with 999 new rows inserted into each target. The source rows are accurately represented in the targets.
- **Set based/ Automatic Commit:** When Warehouse Builder encounters the error inserting into Target_2, it does not load any rows and reports an error for the table. It does, however, continue to insert rows into Target_3 and does not roll back the rows from Target_1. Assuming Warehouse Builder encounters no other errors, the mapping completes with one error message for Target_2, no rows inserted into Target_2, and 1,000 rows inserted into Target_1 and Target_3. The source rows are not accurately represented in the targets.
- **Row based/Automatic Commit:** Beginning with the first row, Warehouse Builder evaluates each row separately for loading into the targets. Loading continues in this way until Warehouse Builder encounters an error loading row 100 to Target_2 and reports the error. Warehouse Builder does not roll back row 100 from Target_1, does insert it into Target_3, and continues to load the remaining rows. Assuming Warehouse Builder encounters no other errors, the mapping completes with 999 rows inserted into Target_2 and 1,000 rows inserted into each of the other targets. The source rows are not accurately represented in the targets.

Embedding Commit Logic into the Mapping

For PL/SQL mappings only, you can embed commit logic into the mapping design by adding a pre or post mapping operator with SQL statements to commit and rollback data. When you run the mapping, Warehouse Builder commits or rollback data based solely on the SQL statements you provide in the pre or post mapping operator.

Use these instructions to implement a business rule that is tedious or impossible to design given existing Warehouse Builder mapping operators. For example, you may want to verify the existence of a single row in a target. Write the required logic in SQL and introduce that logic to the mapping through a pre or post mapping operator.

To include commit logic in the mapping design:

1. Design the mapping to include a pre or post mapping operator. Use one of these operators to introduce commit and rollback SQL statements.
2. Configure the mapping with **Commit Control** set to Manual.

   In the Project Explorer, right-click the mapping and select **Configure.** Under **Code Generation Options,** select **Commit Control** to Manual.

To understand the implications of selecting to commit data manually, refer to "About Manual Commit Control" on page 8-10.

3. Deploy the mapping.
4. Run the mapping.

   Warehouse Builder executes the mapping but does not commit data until processing the commit logic you wrote in the pre or post mapping operator.

Committing Data Independently of Mapping Design

You may want to commit data independently of the mapping design for any of the following reasons:

- **Running Multiple Mappings Before Committing Data**: You may want to run multiple mappings without committing data until successfully running and validating all mappings. This can be the case when you have separate mappings for loading dimensions and cubes.
- **Maintaining targets more efficiently:** If bad data is loaded and committed to a very large target, it can be difficult and time consuming to repair the damage. To avoid this, first check the data and then decide whether to issue a commit or rollback command.

The first step to achieve these goals is to configure the mapping with commit control set to Manual.

About Manual Commit Control

Manual commit control enables you to specify when Warehouse Builder commits data regardless of the mapping design. Manual commit control does not affect auditing statistics. This means that you can view the number of rows inserted and other auditing information before issuing the commit or rollback command.

When using manual commit, be aware that this option may have performance implications. Mappings that you intend to run in parallel maybe be executed serially if the design requires a target to be read after being loaded. This occurs when moving data from a remote source or loading to two targets bound to the same table.

When you enable manual commit control, Warehouse Builder runs the mapping with PEL switched off.

Running Multiple Mappings Before Committing Data

This section provides two sets of instructions for committing data independent of the mapping design. The first set describes how to run mappings and then commit data in a SQL Plus session. Use these instructions to test and debug your strategy of running multiple mappings and then committing the data. Then, use the second set of instructions to automate the strategy.

Both sets of instructions rely upon the use of the main procedure generated for each PL/SQL mapping.

Main Procedure

The main procedure is a procedure that exposes the logic for launching mappings in Warehouse Builder. You can employ this procedure in PL/SQL scripts or use it in interactive SQL Plus sessions.

When you use the main procedure, you must specify one required parameter, *p_status*. And you can optionally specify other parameters relevant to the execution of the

mapping as described in Table 8–4. Warehouse Builder uses the default setting for any optional parameters that you do not specify.

Table 8–4 Parameter for the Main Procedure

Parameter Name	Description
p_status	Use this required parameter to write the status of the mapping upon completion. It operates in conjunction with the predefined variable called *status*. The status variable is defined such that OK indicates the mapping completed without errors. OK_WITH_WARNINGS indicates the mapping completed with user errors. FAILURE indicates the mapping encountered a fatal error.
p_operating_mode	Use this optional parameter to pass in the Default Operating Mode such as SET_BASED.
p_bulk_size	Use this optional parameter to pass in the Bulk Size.
p_audit_level	Use this optional parameter to pass in the Default Audit Level such as COMPLETE.
p_max_no_of_errors	Use this optional parameter to pass in the permitted Maximum Number of Errors.
p_commit_frequency	Use this optional parameter to pass in the Commit Frequency.

Committing Data at Runtime

For PL/SQL mappings only, you can run mappings and issue commit and rollback commands from the SQL Plus session. Based on your knowledge of SQL Plus and the Main Procedure, you can manually run and validate multiple mappings before committing data.

To commit data manually at runtime:

1. Design the PL/SQL mappings. For instance, create one mapping to load dimensions and create a separate mapping to load cubes.

   These instructions are not valid for SQL*Loader and ABAP mappings.

2. Configure both mappings with **Commit Control** set to Manual.

   In the Project Explorer, right-click the mapping and select **Configure.** Under **Code Generation Options,** set **Commit Control** to Manual.

3. Generate each mapping.

4. From a SQL Plus session, issue the following command to execute the first mapping called *map1* in this example:

   ```
   var status varchar2(30);
   ```

   execute `map1.main(:status);`

   The first line declares the predefined status variable described in Table 8–4. In the second line, *p_status* is set to the status variable. When *map1* completes, SQL Plus displays the mapping status such as *OK*.

5. Execute the second mapping, in this example, the cubes mappings called *map2*.

   You can run the second in the same way you ran the previous map. Or, you can supply additional parameters listed in Table 8–4 to dictate how to run the *map2* in this example:

   ```
   map2.main (p_status => :status,            \
   ```

```
p_operating_mode => 'SET_BASED',  \
p_audit_level => 'COMPLETE')  ;
```

6. Verify the results from the execution of the two mappings and send either the commit or rollback command.
7. Automate your commit strategy as described in "Committing Mappings Using the Process Flow Editor" on page 8-12.

Committing Mappings Using the Process Flow Editor

For PL/SQL mappings only, you can commit or rollback mappings together. Based on your knowledge of the Sqlplus activity, the Main Procedure, and writing PL/SQL scripts, you can use process flows to automate logic that commits data after all mappings complete successfully or rollback the data if any mapping fails.

To commit multiple mappings using a process flow:

1. Design the PL/SQL mappings.

   These instructions are not valid for SQL*Loader and ABAP mappings.
2. Ensure each mapping is deployed to the same schema.

   All mappings must have their locations pointing to the same schema. You can achieve this by designing the mappings under the same target module. Or, for multiple target modules, ensure that the locations point to the same schema.
3. Configure each mapping with **Commit Control** set to Manual.

   In the Project Explorer, right-click the mapping and select **Configure.** Under **Code Generation Options,** set **Commit Control** to Manual.
4. Design a process flow using a sqlplus activity instead of multiple mapping activities.

   In typical process flows, you add a mapping activity for each mapping and the process flow executes an implicit commit after each mapping activity. However, in this design, do not add mapping activities. Instead, add a single sqlplus activity.
5. Write a PL/SQL script that uses the main procedure to execute each mapping. The following script demonstrates how to run the next mapping only if the initial mapping succeeds.

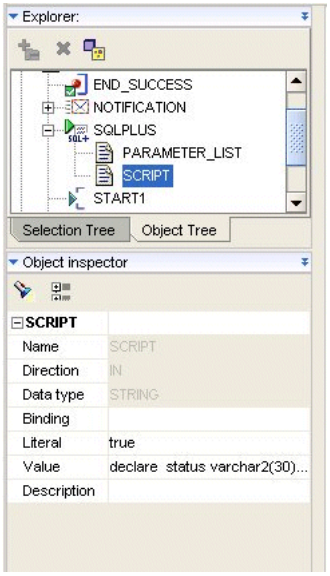
```
declare
  status varchar2(30);
begin
  map1.main(status);
  if status!='OK' then
    rollback
  else
    map2.main(status);
      if status!='OK' then
        rollback;
      else
        commit;
```

```
        end if;
      end if;
    end if;
```

6. Paste your PL/SQL script into the sqlplus activity.

   In the editor explorer, select **SCRIPT** under the sqlplus activity and then double-click **Value** in the object inspector shown in Figure 8–8.

Figure 8–8 Specifying a Script in the Sqlplus Activity



7. Optionally apply a schedule to the process flow as described in "Process for Defining and Using Schedules" on page 11-4.

8. Deploy the mappings, process flow, and schedule if you defined one.

Ensuring Referential Integrity in PL/SQL Mappings

When you design mappings with multiple targets, you may want to ensure that Warehouse Builder loads the targets in a specific order. This is the case when a column in one target derives its data from another target.

To ensure referential integrity in PL/SQL mappings:

1. Design a PL/SQL mapping with multiple targets.

2. Optional step: Define a parent/child relationship between two of the targets by specifying a foreign key.

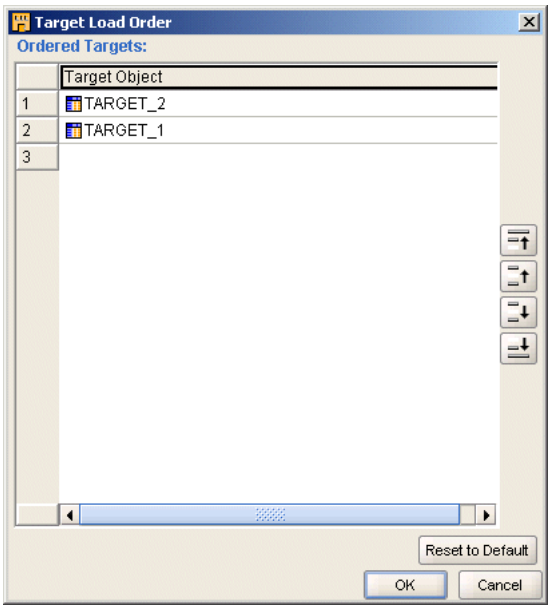
   A foreign key in the child table must refer to a primary key in the parent table. If the parent does not have a column defined as a primary key, you must add a column and define it as the primary key. For an example of how to do this, see "Using Conventional Loading to Ensure Referential Integrity in SQL*Loader Mappings" on page 8-14.

3. In the mapping properties, view the Target Load Order property.

   If you defined a foreign key relationship in the previous step, Warehouse Builder calculates a default loading order that loads parent targets before children. If you

did not define a foreign key, use the Target Load Order dialog shown in to define the loading order.

Figure 8–9 Target Load Order Dialog



For more information, see "Target Load Order" on page 6-27.

4. Ensure that the Use Target Load Ordering configuration property described on page 24-6 is set to its default value of true.

Best Practices for Designing SQL*Loader Mappings

This section includes the following topics:

- Using Conventional Loading to Ensure Referential Integrity in SQL*Loader Mappings on page 8-14
- Using Direct Path Loading to Ensure Referential Integrity in SQL*Loader Mappings on page 8-21

Using Conventional Loading to Ensure Referential Integrity in SQL*Loader Mappings

If you are extracting data from a multiple-record-type file with a master-detail structure and mapping to tables, add a Mapping Sequence operator to the mapping to retain the relationship between the master and detail records through a surrogate primary key or foreign key relationship. A master-detail file structure is one where a master record is followed by its detail records. In Example 8–1, records beginning with "E" are master records with Employee information and records beginning with "P" are detail records with Payroll information for the corresponding employee.

Example 8–1 A Multiple-Record-Type Flat File with a Master-Detail Structure

```
E 003715 4 153 09061987 014000000 "IRENE HIRSH" 1 08500
P 01152000 01162000 00101 000500000 000700000
P 02152000 02162000 00102 000300000 000800000
E 003941 2 165 03111959 016700000 "ANNE FAHEY" 1 09900
P 03152000 03162000 00107 000300000 001000000
```

```
E 001939 2 265 09281988 021300000 "EMILY WELLMET" 1 07700
P 01152000 01162000 00108 000300000 001000000
P 02152000 02162000 00109 000300000 001000000
```

In Example 8–1, the relationship between the master and detail records is inherent only in the physical record order: payroll records correspond to the employee record they follow. However, if this is the only means of relating detail records to their masters, this relationship is lost when Warehouse Builder loads each record into its target table.

Maintaining Relationships Between Master and Detail Records

You can maintain the relationship between master and detail records if both types of records share a common field. If Example 8–1 contains a field Employee ID in both Employee and Payroll records, you can use it as the primary key for the Employee table and as the foreign key in the Payroll table, thus associating Payroll records to the correct Employee record.

However, if your file does not have a common field that can be used to join master and detail records, you must add a sequence column to both the master and detail targets (see Table 8–5 and Table 8–6) to maintain the relationship between the master and detail records. Use the Mapping Sequence operator to generate this additional value.

Table 8–5 represents the target table containing the master records from the file in Example 8–1 on page 8-14. The target table for the master records in this case contains employee information. Columns E1-E10 contain data extracted from the flat file. Column E11 is the additional column added to store the master sequence number. Notice that the number increments by one for each employee.

Table 8–5 Target Table Containing Master Records

E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11
E	003715	4	153	09061987	014000000	"IRENE	HIRSH"	1	08500	1
E	003941	2	165	03111959	016700000	"ANNE	FAHEY"	1	09900	2
E	001939	2	265	09281988	021300000	"EMILY	WELSH"	1	07700	3

Table 8–6 represents the target table containing the detail records from the file in Example 8–1 on page 8-14. The target table for the detail records in this case contains payroll information, with one or more payroll records for each employee. Columns P1-P6 contain data extracted from the flat file. Column P7 is the additional column added to store the detail sequence number. Notice that the number for each payroll record matches the corresponding employee record in Table 8–5.

Table 8–6 Target Table Containing Detail Records

P1	P2	P3	P4	P5	P6	P7
P	01152000	01162000	00101	000500000	000700000	1
P	02152000	02162000	00102	000300000	000800000	1
P	03152000	03162000	00107	000300000	001000000	2
P	01152000	01162000	00108	000300000	001000000	3
P	02152000	02162000	00109	000300000	001000000	3

Extracting and Loading Master-Detail Records

This section contains instructions on creating a mapping that extracts records from a master-detail flat file and loads those records into two different tables. One target table stores master records and the other target table stores detail records from the flat file. The Mapping Sequence is used to maintain the master-detail relationship between the two tables.

> **Note: These instructions are for conventional path loading. For instructions on using direct path loading for master-detail records, see "Using Direct Path Loading to Ensure Referential Integrity in SQL*Loader Mappings" on page 8-21.**

This procedure outlines general steps for building such a mapping. Additional detailed instructions are available:

- Using the Import Metadata Wizard on page 16-1
- Flat File Operator on page 25-29
- Adding Operators that Bind to Repository Objects on page 6-12
- Sequence Operator on page 25-24
- Configuring Mappings Reference on page 24-1

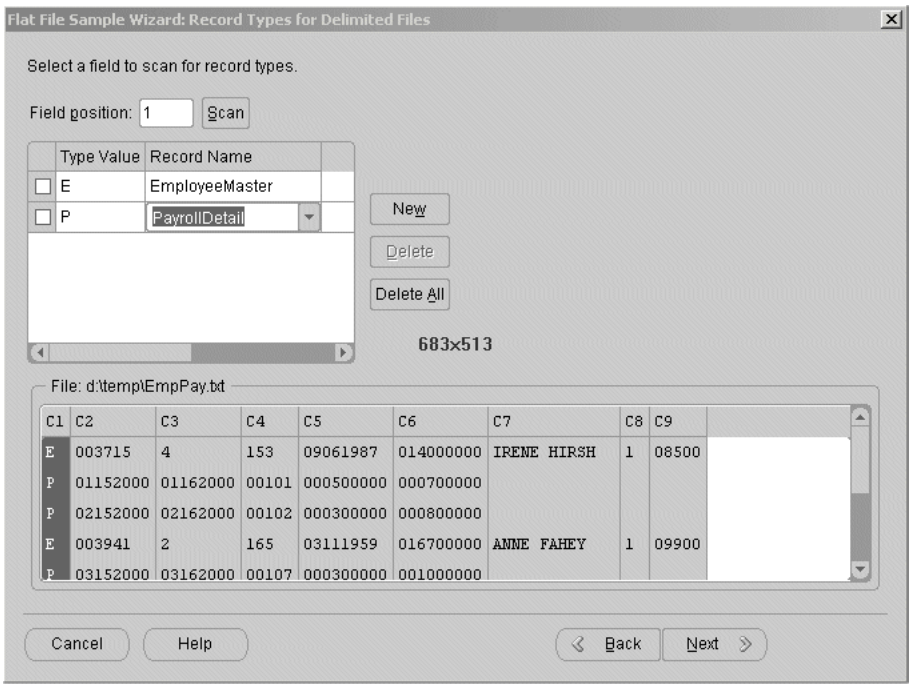
To extract from a master-detail flat file and maintain master-detail relationships:

1. Import and sample a flat file source that consists of master and detail records.

   When naming the record types as you sample the file, assign descriptive names to the master and detail records, as shown in Figure 8–10. This makes it easier to identify those records in the future.

   Figure 8–10 shows the Flat File Sample Wizard for a multiple-record-type flat file containing department and employee information. The master record type (for employee records) is called EmployeeMaster, while the detail record type (for payroll information) is called PayrollDetail.

Figure 8–10 Naming Flat File Master and Detail Record Types

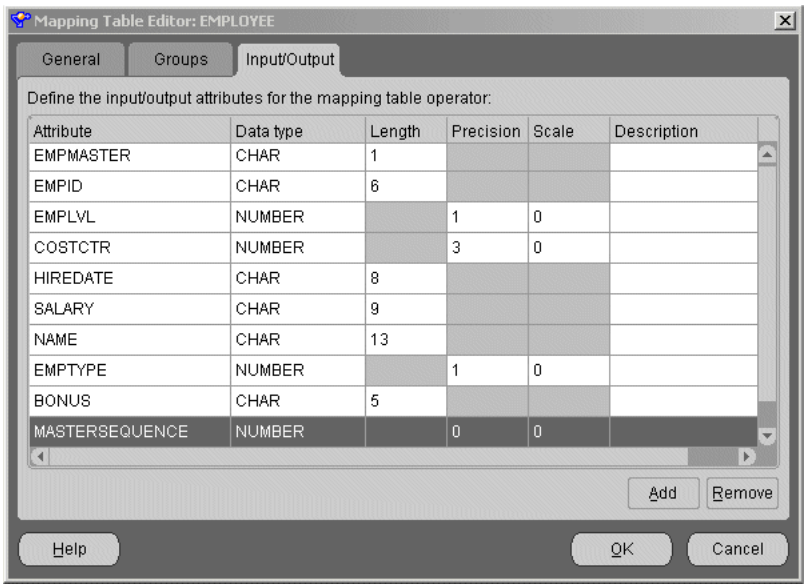


1. Drop a Mapping Flat File operator onto the mapping editor canvas and specify the master-detail file from which you want to extract data.
2. Drop a Mapping Sequence operator onto the mapping canvas.
3. Drop a Table Operator operator for the master records onto the mapping canvas.

   You can either select an existing repository table that you created earlier or create a new unbound table operator with no attributes. You can then map or copy all required fields from the master record of the file operator to the master table operator (creating columns) and perform an outbound reconciliation to define the table later.

   The table must contain all the columns required for the master fields you want to load plus an additional numeric column for loading sequence values, as shown in Figure 8–11.

Figure 8–11 Adding a Sequence Column to the Master and Detail Target Tables



1. Drop a Table Operator operator for the detail records onto the mapping canvas.

   You can either select an existing repository table that you created earlier or create a new unbound table operator with no attributes. You can then map or copy all required fields from the master record of the file operator to the master table operator (creating columns) and perform an outbound synchronize to define the table later.

   The table must contain all the columns required for the detail fields you want to load plus an additional numeric column for loading sequence values.

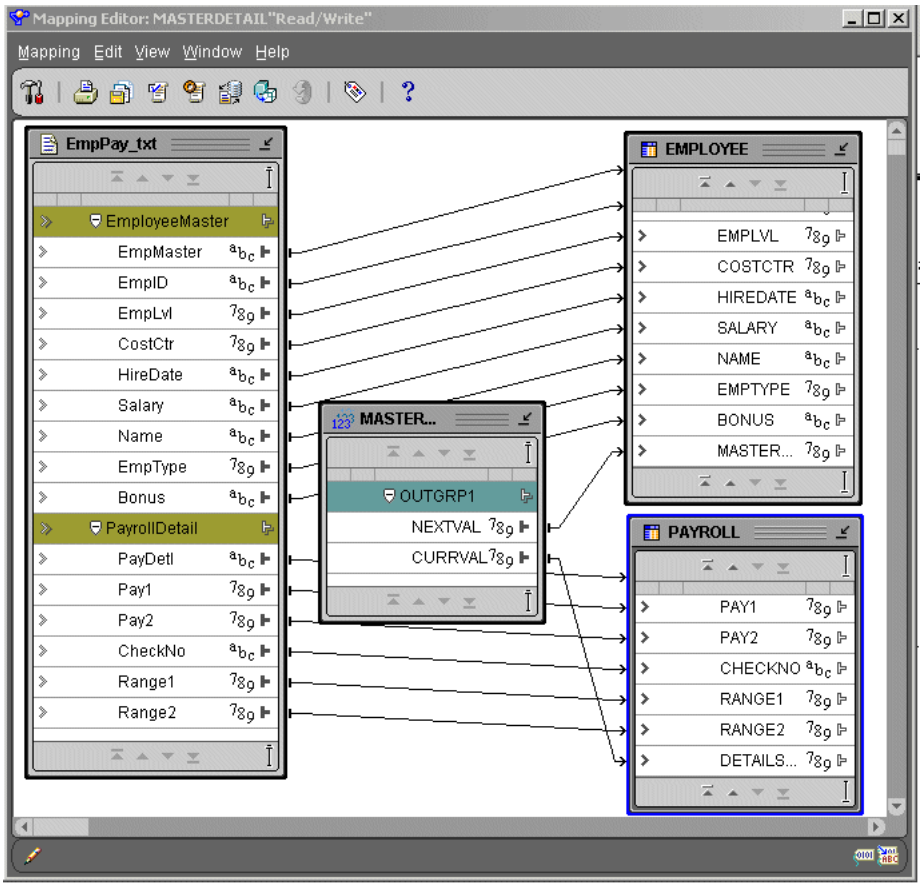
2. Map all of the necessary flat file master fields to the master table and detail fields to the detail table, as shown in Figure 8–12.

3. Map the Mapping Sequence `NEXTVAL` attribute to the additional sequence column in the master table, as shown in Figure 8–12.

4. Map the Mapping Sequence `CURRVAL` attribute to the additional sequence column in the detail table, as shown in Figure 8–12.

   Figure 8–12 shows a completed mapping with the flat file master fields mapped to the master target table, the detail fields mapped to the detail target table, and the `NEXTVAL` and `CURRVAL` attributes from the Mapping Sequence mapped to the master and detail target tables, respectively.

Figure 8–12 Completed Mapping from Master-Detail Flat File to Two Target Tables



1. Configure the mapping with the following parameters:

   Direct Mode: False

   Errors Allowed: 0

   Row: 1

   Trailing Nullcols: True (for all tables)

Error Handling Suggestions

This section contains error handling recommendations for files with varying numbers of errors.

If your data file almost never contains errors:

1. Create a mapping with a Sequence operator (see "Sequence Operator" on page 25-24).
2. Configure a mapping with the following parameters:

   Direct Mode= false

   ROW=1

   ERROR ALLOWED = 0
3. Generate the code and run an SQL Loader script.

   If the data file has errors, the loading stops when the first error happens.

4. Fix the data file and run the control file again with the following configuration values:

   CONTINUE_LOAD=TRUE

   SKIP=*number of records already loaded*

If your data file is likely to contain a moderate number of errors:

1. Create a primary key (PK) for the master record based on the `seq_nextval` column.
2. Create a foreign key (FK) for the detail record based on the `seq_currval` column which references the master table PK.

   In this case, master records with errors will be rejected with all their detail records. You can recover these records by following these steps.
3. Delete all failed detail records that have no master records.
4. Fix the errors in the bad file and reload only those records.
5. If there are very few errors, you may choose to load the remaining records and manually update the table with correct sequence numbers.
6. In the log file, you can identify records that failed with errors because those errors violate the integrity constraint. The following is an example of a log file record with errors:

```
Record 9: Rejected - Error on table "MASTER_T", column "C3".
ORA-01722: invalid number
Record 10: Rejected - Error on table "DETAIL1_T".
ORA-02291: integrity constraint (SCOTT.FK_SEQ) violated - parent key not found
Record 11: Rejected - Error on table "DETAIL1_T".
ORA-02291: integrity constraint (SCOTT.FK_SEQ) violated - parent key not found
Record 21: Rejected - Error on table "DETAIL2_T".
ORA-02291: invalid number
```

If your data file always contains many errors:

1. Load all records without using the Mapping Sequence operator.

   Load the records into independent tables. You can load the data in Direct Mode, with the following parameters that increase loading speed:

   ROW>1

   ERRORS ALLOWED=MAX
2. Correct all rejected records.
3. Reload the file again with a Sequence operator (see "Sequence Operator" on page 25-24).

Subsequent Operations

After the initial loading of the master and detail tables, you can use the loaded sequence values to further transform, update, or merge master table data with detail table data. For example, if your master records have a column that acts as a unique identifier (such as an Employee ID), and you want to use it as the key to join master and detail rows (instead of the sequence field you added for that purpose), you can update the detail table(s) to use this unique column. You can then drop the sequence column you created for the purpose of the initial load. Operators such as the Aggregator, Filter, or Match and Merge operator can help you with these subsequent transformations.

Using Direct Path Loading to Ensure Referential Integrity in SQL*Loader Mappings

If you are using a master-detail flat file where the master record has a unique field (or if the concatenation of several fields can result in a unique identifier), you can use Direct Path Load as an option for faster loading.

For direct path loading, the record number (`RECNUM`) of each record is stored in the master and detail tables. A post-load procedure uses the `RECNUM` to update each detail row with the unique identifier of the corresponding master row.

This procedure outlines general steps for building such a mapping. Additional detailed instructions are available:

- For additional information on importing flat file sources, see "Using the Import Metadata Wizard" on page 16-1.
- For additional information on using the Mapping Flat File as a source, see "Flat File Operator" on page 25-29.
- For additional information on using Table Operators, see "Adding Operators that Bind to Repository Objects" on page 6-12.
- For additional information on using the Data Generator operator, see "Data Generator Operator" on page 25-12.
- For additional information on using the Constant operator, see "Constant Operator" on page 25-8.
- For additional information on configuring mappings, see "Configuring Mappings Reference" on page 24-1.

To extract from a master-detail flat file using direct path load to maintain master-detail relationships:

1. Import and sample a flat file source that consists of master and detail records.

   When naming the record types as you sample the file, assign descriptive names to the master and detail records, as shown in Figure 8–10 on page 8-17. This will make it easier to identify those records in the future.

2. Drop a Mapping Flat File operator onto the mapping canvas and specify the master-detail file from which you want to extract data.

3. Drop a Data Generator and a Constant operator onto the mapping canvas.

4. Drop a Table Operator operator for the master records onto the mapping canvas.

   You can either select an existing repository table that you created earlier, or create a new unbound table operator with no attributes and perform an outbound synchronize to define the table later.

   The table must contain all the columns required for the master fields you plan to load plus an additional numeric column for loading the `RECNUM` value.

5. Drop a Table Operator for the detail records onto the mapping canvas.

   You can either select an existing repository table that you created earlier, or create a new unbound table operator with no attributes and perform an outbound synchronize to define the table later.

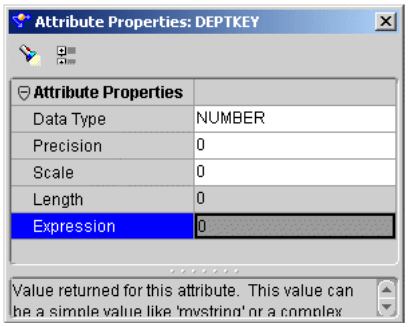
   The table must contain all the columns required for the detail fields you plan to load plus an additional numeric column for loading a `RECNUM` value, and a column that will be updated with the unique identifier of the corresponding master table row.

6. Map all of the necessary flat file master fields to the master table and detail fields to the detail table, as shown in Figure 8–14 on page 8-23.
7. Map the Data Generator operator's RECNUM attribute to the RECNUM columns in the master and detail tables, as shown in Figure 8–14 on page 8-23.
8. Add a constant attribute in the Constant operator.

   If the master row unique identifier column is of a CHAR data type, make the constant attribute a CHAR type with the expression '*'.

   If the master row unique identifier column is a number, make the constant attribute a NUMBER with the expression '0'. Figure 8–13 shows the expression property of the constant attribute set to '0'. This constant marks all data rows as "just loaded."

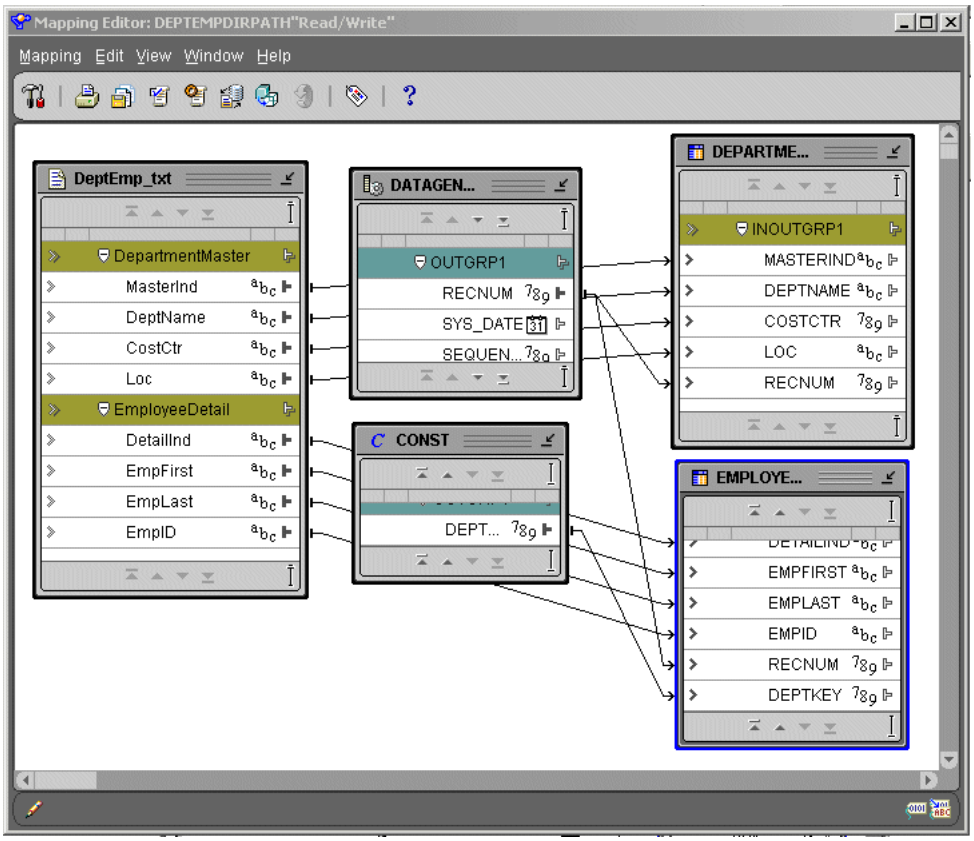
Figure 8–13 Constant Operator Properties



1. Map the constant attribute from the Constant operator to the detail table column that will later store the unique identifier for the corresponding master table record.

   Figure 8–14 shows a completed mapping with the flat file's master fields mapped to the master target table, the detail fields mapped to the detail target table, the RECNUM attributes from the Data Generator operator mapped to the master and detail target tables, respectively, and the constant attribute mapped to the detail target table.

Figure 8–14 Completed Mapping from Master-Detail Flat File with a Direct Path Load



1. Configure the mapping with the following parameters:

   Direct Mode: True

   Errors Allowed: 0

   Trailing Nullcols: True (for each table)

2. After you validate the mapping and generate the SQL*Loader script, create a post-update PL/SQL procedure and add it to the Warehouse Builder library.
3. Run the SQL*Loader script.
4. Execute an UPDATE SQL statement by running a PL/SQL post-update procedure or manually executing a script.

The following is an example of the generated SQL*Loader control file script:

```
OPTIONS ( DIRECT=TRUE,PARALLEL=FALSE, ERRORS=0, BINDSIZE=50000, ROWS=200,
READSIZE=65536)
LOAD DATA
CHARACTERSET WE8MSWIN1252
  INFILE 'g:\FFAS\DMR2.dat'
  READBUFFERS 4
  INTO TABLE "MATER_TABLE"
  APPEND
  REENABLE DISABLED_CONSTRAINTS
       WHEN
      "REC_TYPE"='P'
  FIELDS
    TERMINATED BY ','
    OPTIONALLY ENCLOSED BY '"'
```

```
    TRAILING NULLCOLS

  (
  "REC_TYPE" POSITION (1) CHAR ,
  "EMP_ID" CHAR ,
  "ENAME" CHAR ,
  "REC_NUM" RECNUM
  )

INTO TABLE "DETAIL_TABLE"
  APPEND
  REENABLE DISABLED_CONSTRAINTS
       WHEN
      "REC_TYPE"='E'
  FIELDS
    TERMINATED BY ','
    OPTIONALLY ENCLOSED BY '"'
    TRAILING NULLCOLS
     (
  "REC_TYPE" POSITION (1) CHAR ,
  "C1" CHAR ,
  "C2" CHAR ,
  "C3" CHAR ,
  "EMP_ID" CONSTANT '*',
 "REC_NUM" RECNUM
```

The following is an example of the post-update PL/SQL procedure:

```
  create or replace procedure wb_md_post_update(
     master_table varchar2
    ,master_recnum_column varchar2
    ,master_unique_column varchar2
    ,detail_table varchar2
    ,detail_recnum_column varchar2
    ,detail_masterunique_column varchar2
    ,detail_just_load_condition varchar2)
  IS
     v_SqlStmt VARCHAR2(1000);
  BEGIN
     v_SqlStmt := 'UPDATE '||detail_table||' l '||
                  ' SET l.'||detail_masterunique_column||' = (select i.'||master_
unique_column||
                  ' from '||master_table||' i '||
                  ' WHERE i.'||master_recnum_column||' IN '||
                  ' (select max(ii.'||master_recnum_column||') '||
                  ' from '||master_table||' ii '||
                  ' WHERE ii.'||master_recnum_column||' < l.'||detail_recnum_
column||') '||
                  ' ) '||
                  ' WHERE l.'||detail_masterunique_column||' = '||''''||detail_
just_load_condition||'''';
     dbms_output.put_line(v_sqlStmt);
     EXECUTE IMMEDIATE  v_SqlStmt;
  END;
  /
```

Improved Performance Through Partition Exchange Loading

Data partitioning can improve performance when loading or purging data in a target system. This practice is known as Partition Exchange Loading (PEL).

PEL is recommended when loading a relatively small amount of data into a target containing a much larger volume of historical data. The target can be a table, a dimension, or a cube in a data warehouse.

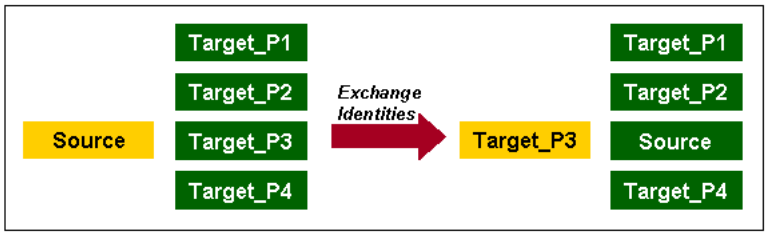
This section includes the following topics:

About Partition Exchange Loading

By manipulating partitions in your target system, you can use Partition Exchange Loading (PEL) to instantly add or delete data. When a table is exchanged with an empty partition, new data is added.

You can use PEL to load new data by exchanging it into a target table as a partition. For example, a table that holds the new data assumes the identity of a partition from the target table and this partition assumes the identity of the source table. This exchange process is a DDL operation with no actual data movement. Figure 8–15 illustrates this example.

Figure 8–15 Overview of Partition Exchange Loading



In Figure 8–15, data from a source table *Source* is inserted into a target table consisting of four partitions (`Target_P1`, `Target_P2`, `Target_P3`, and `Target_P4`). If the new data needs to be loaded into `Target_P3`, the partition exchange operation only exchanges the names on the data objects without moving the actual data. After the exchange, the formerly labeled `Source` is renamed to `Target_P3`, and the former `Target_P3` is now labeled as `Source`. The target table still contains four partitions: `Target_P1`, `Target_P2`, `Target_P3`, and `Target_P4`. The partition exchange operation available in Oracle9*i* completes the loading process without data movement.

Configuring a Mapping for PEL

To configure a mapping for partition exchange loading, complete the following steps:

1. In the Project Explorer, right-click a mapping and select **Configure.**

   Warehouse Builder displays the Configuration Properties dialog.

2. By default, PEL is disabled for all mappings. Select **PEL Enabled** to use Partition Exchange Loading.
3. Use **Data Collection Frequency** to specify the amount of new data to be collected for each run of the mapping. Set this parameter to specify if you want the data collected by Year, Quarter, Month, Day, Hour, or Minute. This determines the number of partitions.
4. Select **Direct** if you want to create a temporary table to stage the collected data before performing the partition exchange. If you do not select this parameter, Warehouse Builder directly swaps the source table into the target table as a partition without creating a temporary table. For more information, see "Direct and Indirect PEL" on page 8-26.
5. If you select **Replace Data**, Warehouse Builder replaces the existing data in the target partition with the newly collected data. If you do not select it, Warehouse Builder preserves the existing data in the target partition. The new data is inserted into a non-empty partition. This parameter affects the local partition and can be used to remove or swap a partition out of a target table. At the table level, you can set Truncate/Insert properties.

Direct and Indirect PEL

When you use Warehouse Builder to load a target by exchanging partitions, you can load the target indirectly or directly.

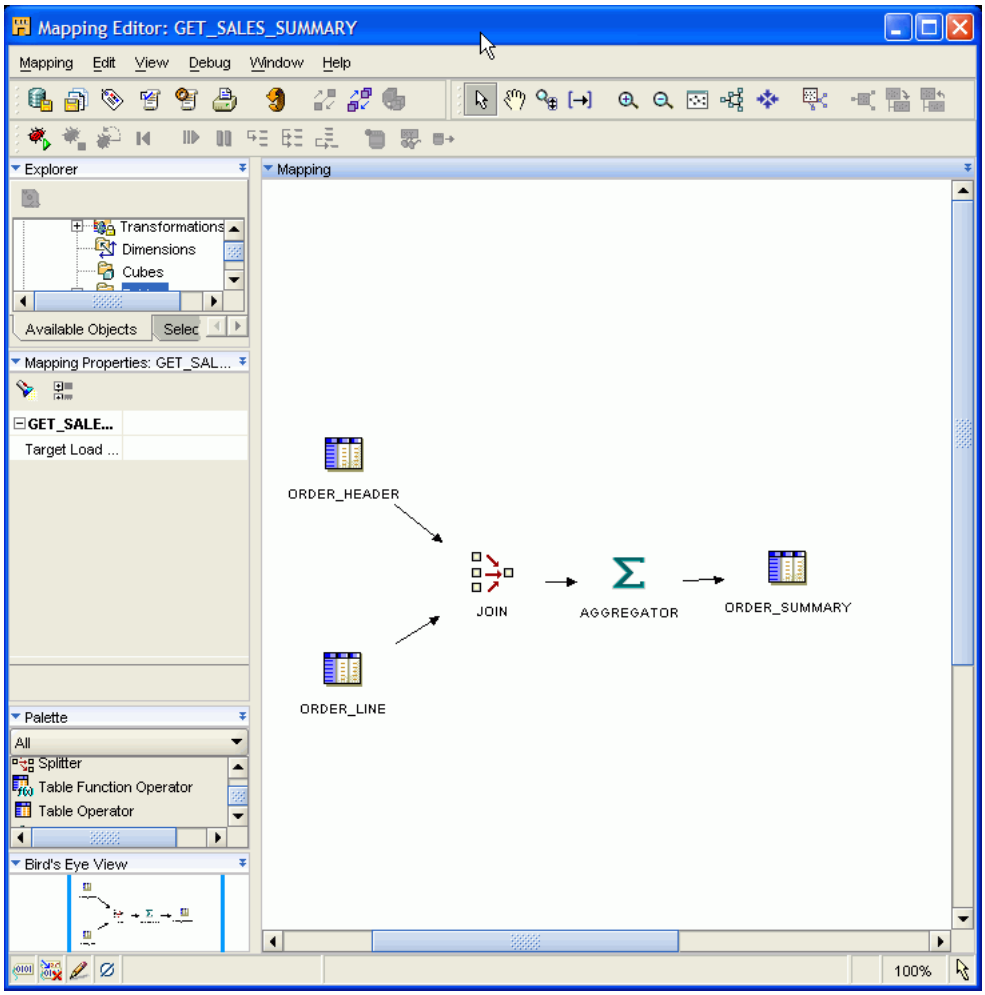
- **Indirect PEL:** By default, Warehouse Builder creates and maintains a temporary table that stages the source data before initiating the partition exchange process. For example, use Indirect PEL when the mapping includes a remote source or a join of multiple sources.
- **Direct PEL:** You design the source for the mapping to match the target structure. For example, use Direct PEL in a mapping to instantaneously publish fact tables that you loaded in a previously executed mapping.

Using Indirect PEL

If you design a mapping using PEL and it includes remote sources or a join of multiple sources, Warehouse Builder must perform source processing and stage the data before partition exchange can proceed. Therefore, configure such mappings with Direct PEL set to False. Warehouse Builder transparently creates and maintains a temporary table that stores the results from source processing. After performing the PEL, Warehouse Builder drops the table.

Figure 8–17 shows a mapping that joins two sources and performs an aggregation. If all new data loaded into the ORDER_SUMMARY table is always loaded into same partition, then you can use Indirect PEL on this mapping to improve load performance. In this case, Warehouse Builder transparently creates a temporary table after the Aggregator and before ORDER_SUMMARY.

Figure 8–16 Mapping with Multiple Sources



Warehouse Builder creates the temporary table using the same structure as the target table with the same columns, indexes, and constraints. For the fastest performance, Warehouse Builder loads the temporary table using parallel direct-path loading INSERT. After the INSERT, Warehouse Builder indexes and constrains the temporary table in parallel.

Example: Using Direct PEL to Publish Fact Tables

Use Direct PEL when the source table is local and the data is of good quality. You must design the mapping such that the source and target are in the same database and have exactly the same structure. The source and target must have the same indexes and constraints, the same number of columns, and the same column types and lengths.

For example, assume that you have the same mapping from Figure 8–17 but would like greater control on when data is loaded into the target. Depending on the amount of data, it could take hours to load and you would not know precisely when the target table would be updated.

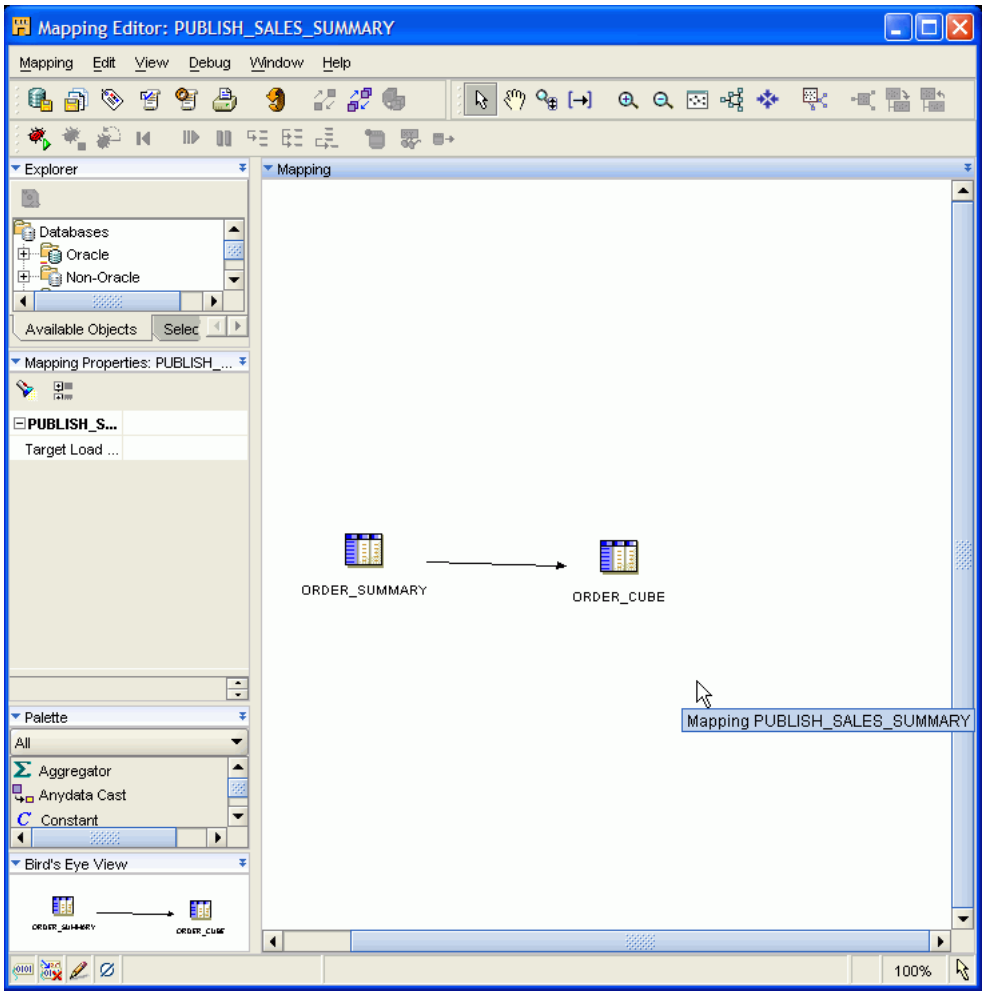
To instantly load data to a target using Direct PEL:

1. Design one mapping to join source data, if necessary, transform data, ensure data validity, and load it to a staging table. Do not configure this mapping to use PEL.

Design the staging table to exactly match the structure of the final target that you will load in a separate mapping. For example, the staging table in Figure 8–17 is ORDER_SUMMARY and should be of the same structure as the final target, ORDER_CUBE in Figure 8–18.

2. Create a second mapping that loads data from the staging table to the final target such as shown in Figure 8–18. Configure this mapping to use Direct PEL.

Figure 8–17 Publish_Sales_Summary Mapping



3. Use either the Warehouse Builder Process Flow Editor or Oracle Workflow to launch the second mapping after the completion of the first.

Using PEL Effectively

You can use PEL effectively for scalable loading performance if the following conditions are true:

- **Table partitioning and tablespace:** The target table must be Range partitioned by one DATE column. All partitions must be created in the same tablespace. All tables are created in the same tablespace.
- **Existing historical data:** The target table must contain a huge amount of historical data. An example use for PEL is for a click stream application where the target

collects data every day from an OLTP database or Web log files. New data is transformed and loaded into the target that already contains historical data.

- **New data:** All new data must to be loaded into the same partition in a target table. For example, if the target table is partitioned by day, then the daily data should be loaded into one partition.
- **Loading Frequency:** The loading frequency should be equal to or less than the data collection frequency.
- **No global indexes:** There must be no global indexes on the target table.

Configuring Targets in a Mapping

To configure targets in a mapping for PEL:

- **Step 1: Create All Partitions**
- **Step 2: Create All Indexes Using the LOCAL Option**
- **Step 3: Primary/Unique Keys Use "USING INDEX" Option**

Step 1: Create All Partitions

Warehouse Builder does not automatically create partitions during runtime. Before you can use PEL, you must create all partitions as described in "Using Partitions" on page 12-4.

For example, if you select Month as the frequency of new data collection, you need to create all the required partitions for each month of new data. Use the Data Object Editor to create partitions for a table, dimension, or cube. Figure 8–18 shows the property inspector window for table ORDER_SUMMARY. This figure shows six partitions that have been added for this table.

To use PEL, all partition names must follow a naming convention. For example, for a partition that will hold data for May 2002, the partition name must be in the format Y2002_Q2_M05.

For PEL to recognize a partition, its name must fit one of the following formats.

Ydddd

Ydddd_**Q**d

Ydddd_**Q**d_**M**dd

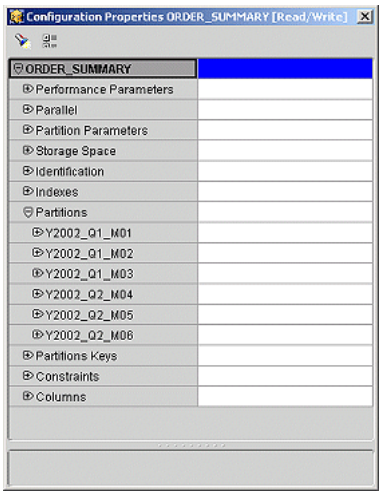
Ydddd_**Q**d_**M**dd_**D**dd

Ydddd_**Q**d_**M**dd_**D**dd_**H**dd

Ydddd_**Q**d_**M**dd_**D**dd_**H**dd_**M**dd

Where d represents a decimal digit. All the letters must be in upper case. Lower case is not recognized.

Figure 8–18 Configuration Properties for Table ORDER_SUMMARY

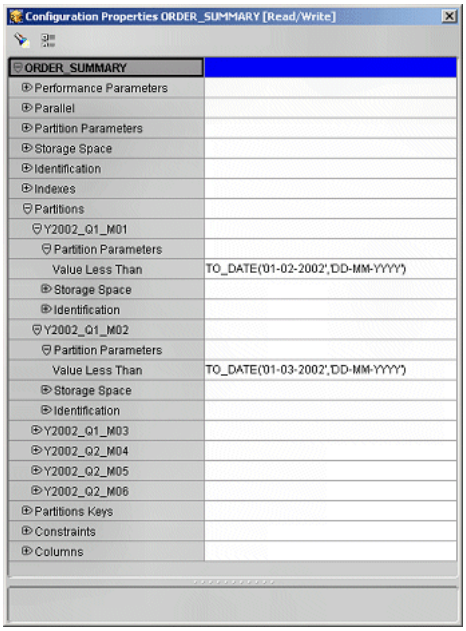


If you correctly name each partition, Warehouse Builder automatically computes the Value Less Than property for each partition. Otherwise, you must manually configure Value Less Than for each partition for Warehouse Builder to generate a DDL statement. The following is an example of a DDL statement generated by Warehouse Builder:

```
. . .
PARTITION A_PARTITION_NAME
      VALUES LESS THAN (TO_DATE('01-06-2002','DD-MM-YYYY')),
. . .
```

Figure 8–19 shows automatically generated configuration values for the Value Less Than parameter.

Figure 8–19 Automatically Generated "Value Less Than" Setting



Step 2: Create All Indexes Using the LOCAL Option

Add an index (ORDER_SUMMARY_PK_IDX) to the ORDER_SUMMARY table. This index has two columns, ORDER_DATE and ITEM_ID. Configure the following:

- Set the Index Type parameter to UNIQUE.
- Set the Local Index parameter to True.

Now Warehouse Builder can generate a DDL statement for a unique local index on table ORDER_SUMMARY.

Using local indexes provides the most important PEL performance benefit. Local indexes require all indexes to be partitioned in the same way as the table. When the temporary table is swapped into the target table using PEL, so are the identities of the index segments.

If an index is created as a local index, the Oracle server requires that the partition key column must be the leading column of the index. In the preceding example, the partition key is ORDER_DATE and it is the leading column in the index ORDER_SUMMARY_PK_IDX.

Step 3: Primary/Unique Keys Use "USING INDEX" Option

In this step you must specify that all primary key and unique key constraints are created with the USING INDEX option.

With the USING INDEX option, a constraint will not trigger automatic index creation when it is added to the table. The server will search existing indexes for an index with same column list as that of the constraint. Thus, each primary or unique key constraint must be backed by a user-defined unique local index. The index required by the constraint ORDER_SUMMARY_PK is ORDER_SUMMARY_PK_IDX which was created in "Step 2: Create All Indexes Using the LOCAL Option" on page 8-31.

Restrictions for Using PEL in Warehouse Builder

These are the restrictions for using PEL in Warehouse Builder:

- **Only One Date Partition Key:** Only one partition key column of DATE data type is allowed. Numeric partition keys are not supported in Warehouse Builder.
- **Only Natural Calendar System:** The current PEL method supports only the natural calendar system adopted worldwide. Specific business calendar systems with user-defined fiscal and quarter endings are currently not supported.
- **All Data Partitions Must Be In The Same Tablespace:** All partitions of a target (table, dimension, or cube) must be created in the same tablespace.
- **All Index Partitions Must Be In The Same Tablespace:** All indexes of a target (table, dimension, or cube) must be created in the same tablespace. However, the index tablespace can be different from the data tablespace.

High Performance Data Extraction from Remote Sources

Although you can design mappings to access remote sources through database links, performance is likely to be slow when you move large volumes of data. For mappings that move large volumes of data between sources and targets of the same Oracle Database version, you have an option for dramatically improving performance through the use of transportable modules. For instructions on using transportable modules, see Chapter 23, "Moving Large Volumes of Data".

9

Using Oracle Warehouse Builder Transformations

One of the main functions of an Extract, Transformation, and Loading (ETL) tool is to transform data. Oracle Warehouse Builder provides several methods of transforming data. This chapter discusses transformations and describes how to create custom transformation using Warehouse Builder. It also describes how to import transformation definitions.

This chapter contains the following topics:

About Transformations

Transformations are PL/SQL functions, procedures, and packages that enable you to transform data. You use transformations when designing mappings and process flows that define ETL processes.

Transformations are stored in the Warehouse Builder repository. Depending on where the transformation is defined, transformations can be shared either across the project in which they are defined or across all the projects.

Transformation packages are deployed at the package level but executed at the transformation level.

Types of Transformations

Transformations in Warehouse Builder can be categorized as follows:

- Predefined Transformations
- Custom Transformations

The following sections provide more details about these types of transformations.

Predefined Transformations

Warehouse Builder provides a set of predefined transformations that enable you to perform common transformation operations. These predefined transformations are

part of the Oracle Library that consists of built-in and seeded functions and procedures. You can directly use these predefined transformations to transform your data. For more information on the Oracle Library, see "Types of Transformation Libraries" on page 9-4.

Predefined transformations are organized into the following categories:

- Administration
- Character
- Control Center
- Conversion
- Date
- Numeric
- OLAP
- Other
- Streams
- XML

For more information about the transformations that belong to each category, see Chapter 27, "Transformations".

Custom Transformations

A custom transformation is created by the user. Custom transformations can use predefined transformations as part of their definition.

Custom transformations contains the following categories:

- **Functions:** The functions category contains standalone functions. It is available under the Public Transformations node of the Global Explorer. It is also created automatically under the Transformations node of every Oracle module in the Project Explorer. Functions can be defined by the user or imported from a database. A function transformation takes 0-n input parameters and produces a result value.
- **Procedures:** The procedures category contains any standalone procedures used as transformations. It is available under the Public Transformations node of the Global Explorer. The Procedures category is also automatically created under the Transformations node of each Oracle module in the Global Explorer. Procedures can be defined by the user or imported from a database. A procedure transformation takes 0-n input parameters and produces 0-n output parameters.
- **Packages:** PL/SQL packages can be created or imported in Warehouse Builder. The package body may be modified. The package header, which is the signature for the function or procedure, cannot be modified. The package can be viewed in the transformation library property sheet.
- **PL/SQL Types:** PL/SQL types include PL/SQL record types, ref cursor types, and nested table types. The PL/SQL types category contains any standalone PL/SQL types. It is automatically created under the Package node of every transformation, both in the Project Explorer and Global Explorer.

For more information about creating custom transformations, see "Defining Custom Transformations" on page 9-5.

In addition to the categories listed in this section, you can also import PL/SQL packages. Although you can modify the package body of an imported package, you cannot modify the package header, which is the signature for the function or procedure. For more information on importing PL/SQL packages, see "Importing PL/SQL" on page 9-14.

Transforming Data Using Warehouse Builder

Warehouse Builder provides an intuitive user interface that enables you to define transformations. You can either use the predefined transformations or define custom transformations that suit your requirements. Transformations are stored in the repository. Custom transformation can be deployed to the Oracle Database just like any other data object that you define in an Oracle module.

The Mapping Editor includes a set of prebuilt transformation operators that enable you to define common transformations when you define how data will move from source to target. Transformation operators are prebuilt PL/SQL functions, procedures, package functions, and package procedures. They take input data, perform operations on it and produce output. For more information on these operators, see Chapter 26, "Data Flow Operators".

Benefits of Using Warehouse Builder for Transforming Data

Warehouse Builder enables you to reuse PL/SQL as well as to write your own PL/SQL transformations. To enable faster development of warehousing solutions, Warehouse Builder provides custom procedures and functions written in PL/SQL.

Because SQL and PL/SQL are versatile and proven languages widely used by many information professionals, the time and expense of developing an alternative transformation language is eliminated by using Warehouse Builder. With Warehouse Builder, you can create solutions using existing knowledge and a proven, open, and standard technology.

All major relational database management systems support SQL and all programs written in SQL can be moved from one database to another with very little modification.

This means that all the SQL knowledge in your organization is fully portable to Warehouse Builder. Warehouse Builder enables you to import and maintain any existing complex custom code. You can later use these custom transformations in Warehouse Builder mappings.

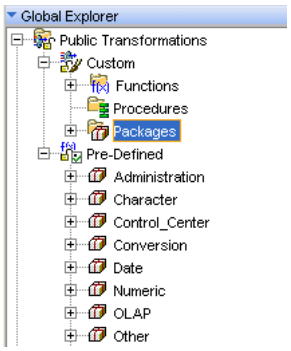
About Transformation Libraries

A transformation library consists of a set of reusable transformations. Each time you create a repository, Warehouse Builder creates a Transformation Library containing transformation operations for that project. This library contains the standard Oracle Library and an additional library for each Oracle module defined within the project.

Types of Transformation Libraries

Transformation libraries are available under the Public Transformations node of the Global Explorer in the Design Center. Figure 9–1 displays the Global Explorer with the Public Transformations node expanded.

Figure 9–1 Global Explorer with the Oracle Library



Transformation libraries can be categorized as follows:

- **Oracle Library**

  This is a collection of predefined functions from which you can define procedures for your Global Shared Library. The Oracle Library is contained in the Global Explorer. Expand the Pre-Defined node under the Public Transformation node. Each category of predefined transformations is represented by a separate node as shown in Figure 9–1. Expand the node for a category to view the predefined transformations in that category. For example, expand the Character node to view the predefined character transformations contained in the Oracle library.

- **Global Shared Library**

  This is a collection of reusable transformations created by the user. These transformations are categorized as functions, procedures, and packages defined within your repository.

  The transformations in the Global Shared Library are available under the Custom node of the Public Transformations node as shown in Figure 9–1. Any transformation that you create under this node is available across all projects in the repository. For information on creating transformations in the global shared library, see "Defining Custom Transformations" on page 9-5.

  When you deploy a transformation defined in the Global Shared Library, the transformation is deployed to the location that is associated with the default control center.

Accessing Transformation Libraries

Since transformations can be used at different points in the ETL process, Warehouse Builder enables you to access transformation libraries from different points in the Design Center.

You can access the Transformation Libraries using the following:

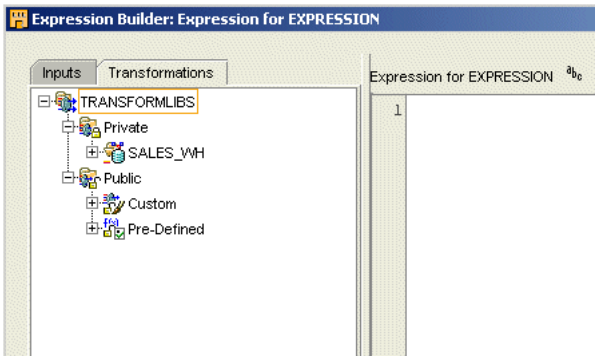
- Expression Builder

  While creating mappings, you may need to create expressions to transform your source data. The Expression Builder interface enables you to create the expressions required to transform data. Since these expressions can include transformations, Warehouse Builder enables you to access transformation libraries from the Expression Builder.

  Transformation libraries are available under the Transformations tab of the Expression Builder as shown in Figure 9–2. The Private node under TRANSFORMLIBS contains transformations that are available only in the current

project. These transformations are created under the Transformation node of the Oracle module. The Public node contains the custom transformations from the Global shared Library and the predefined transformations from the Oracle Library.

Figure 9–2 Transformation Libraries in the Expression Builder



- Add Transformation Operator dialog

  The Transformation operator in the Mapping Editor enables you to add transformations, both from the Oracle library and the Global Shared Library, to a mapping. You can use this operator to transform data as part of the mapping.

- Create Transformation Wizard

  The Implementation page of the Create Transformation Wizard enables you to specify the PL/SQL code that is part of the function or procedure body. You can use transformations in the PL/SQL code.

Defining Custom Transformations

Custom transformations include procedures, functions, and packages. Warehouse Builder provides wizards to create each type of custom transformation. Custom transformations can belong to the Global Shared Library or to a particular project.

Custom Transformations in the Global Shared Library

Custom transformations that are part of the Global Shared Library can be used across all projects of the repository in which they are defined. For example, you create a function called ADD_EMPL in the Global Shared Library of the repository REP_OWNER. This procedure can be used across all the projects in REP_OWNER.

You use the Custom node of the Public Transformations node in the Global Explorer to define custom transformations that can be used across all projects in the repository. Figure 9–1 displays the Global Explorer that you use to create such transformations.

To create a custom transformation in the Global Shared Library:

1. From the Global Explorer, expand the Public Transformations node and then the Custom node.

   Warehouse Builder displays the type of transformations that you can create. This includes functions, procedures, and packages. Note that PL/SQL types can be created only as part of a package.

2. Right-click the type of transformation you want to define and select **New**.

For example, to create a function, right-click **Functions** and select **New**. To create PL/SQL types, expand the package in which you want to create the PL/SQL type, right-click **PL/SQL Types** and select **New**.

3. For functions and procedures, Warehouse Builder displays the Welcome page of the Create Function Wizard or the Create Procedure wizard respectively. For PL/SQL types, Warehouse Builder displays the Welcome page of the Create PL/SQL Type Wizard.

   Click **Next** to proceed. See "Defining Functions and Procedures" on page 9-7 for more information about the other pages in the wizard.

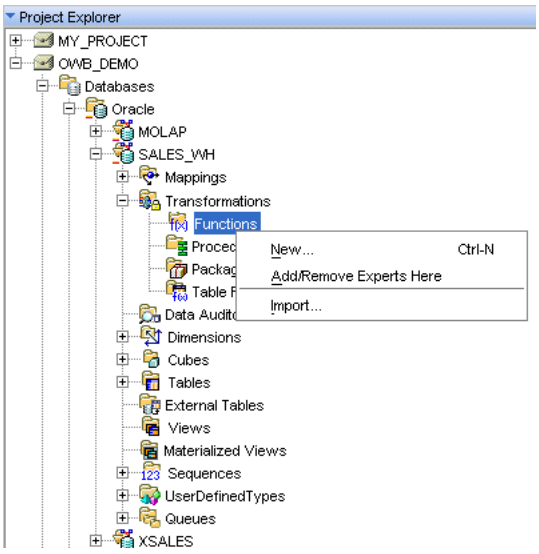
   For packages, Warehouse Builder displays the Create Transformation Library dialog. Provide a name and an optional description for the package and click **OK**. The new package is added to the Packages node. You can subsequently create procedures, functions, or PL/SQL types that belong to this package. For more information about creating PL/SQL types, see "Defining PL/SQL Types" on page 9-8.

Custom Transformations in a Project

Sometimes, you may need to define custom transformations that are required only in the current module or project. In this case, you can define custom transformations in an Oracle module of a project. Such custom transformations are accessible from all the projects in the current repository. For example, consider the repository owner called REP_OWNER that contains two projects PROJECT1 and PROJECT2. In the Oracle module called SALES of PROJECT1, you define a procedure called CALC_SAL. This procedure can be used in all modules belonging to PROJECT1, but is not accessible in PROJECT2.

Figure 9–3 displays the Project Explorer from which you can create custom transformations that are accessible within the project in which they are defined. Expand the Oracle module of the project in which you want to create a custom transformation. Expand the Transformations node under the module. There is a node for each type of custom transformation. Use these nodes to create your transformation.

Figure 9–3 Creating Custom Transformations in an Oracle Module



To define a custom transformation in an Oracle Module:

1. From the Project Explorer, expand the Oracle warehouse module node and then the Transformations node.
2. Right-click the type of transformation you want to create and select **New**.

   For example, to create a package, right-click **Packages** and select **New**. To create PL/SQL types, expand the package node under which you want to create the type, right-click **PL/SQL Types** and select **New**.

   For functions or procedures, Warehouse Builder displays the Welcome page of the Create Function Wizard or the Create Procedure Wizard respectively. For PL/SQL Types, the Welcome page of the Create PL/SQL Type Wizard is displayed. Click **Next** to proceed.

   See "Defining Functions and Procedures" on page 9-7 for information about the remaining wizard pages. For packages, Warehouse Builder opens the Create Transformation Library dialog. Provide a name and an optional description for the package and click **OK**. The package gets added under the Packages node. You can subsequently create procedures, functions, or PL/SQL types that belong to this package. For more information about creating PL/SQL types, see "Defining PL/SQL Types" on page 9-8.

Defining Functions and Procedures

Use the following pages of the Create Function Wizard or Create Procedure Wizard to define a function or procedure.

Name and Description Page

You use the Name and Description page to describe the custom transformation. Specify the following details on this page:

- **Name:** Represents the name of the custom transformation. For more information about naming conventions, see "Naming Conventions for Data Objects" on page 4-4.
- **Description:** Represents the description of the custom transformation. This is an optional field.
- **Return Type:** Represents the data type of the value returned by the function. You select a return type from the available options in the drop-down list. This field is applicable only for functions.

Parameters Page

Use the Parameters page to define the parameters, both input and output, of the transformation. Specify the following details for each parameter:

- **Name:** Enter the name of the parameter.
- **Type:** Select the data type of the parameter from the drop-down list.
- **I/O:** Select the type of parameter. The options available are Input, Output, and Input/Output.

- **Required:** Select Yes to indicate that a parameter is mandatory and No to indicate that it is not mandatory.
- **Default Value:** Enter the default value for the parameter. The default value is used when you do not specify a value for the parameter at the time of executing the function or procedure.

Implementation Page

Use the Implementation page to specify the implementation details, such as the code, of the transformation. To specify the code used to implement the function or procedure, click **Code Editor**. Warehouse Builder displays the Code Editor window of the New Transformation wizard. This editor contains two panels. The upper panel displays the code and the lower panel displays the function signature and messages.

When you create a function, the following additional options are displayed:

- **Function is deterministic:** This hint helps to avoid redundant function calls. If a stored function was called previously with the same arguments, the previous result can be used. The function result should not depend on the state of session variables or schema objects. Otherwise, results might vary across calls. Only DETERMINISTIC functions can be called from a function-based index or a materialized view that has query-rewrite enabled.
- **Enable function for parallel execution:** This option declares that a stored function can be used safely in the child sessions of parallel DML evaluations. The state of a main (logon) session is never shared with child sessions. Each child session has its own state, which is initialized when the session begins. The function result should not depend on the state of session (static) variables. Otherwise, results might vary across sessions.

Summary Page

The Summary page provides a summary of the options that you chose on the previous pages of the wizard. Click **Finish** to complete defining the function or procedure. Warehouse Builder creates the function or procedure and displays it under the corresponding folder under the Public Transformations and Custom nodes in the Global Explorer.

Defining PL/SQL Types

PL/SQL types must be defined within a package. They cannot exist independently.

About PL/SQL Types

PL/SQL types enable you to create collection types, record types, and REF cursor types in Warehouse Builder. You use PL/SQL types as parameters in subprograms or as return types for functions. Using PL/SQL types as parameters to subprograms enables you to process arbitrary number of elements. Use collection types to move data into and out of database tables using bulk SQL. For more information about PL/SQL types, refer to the *Oracle 10g Database PL/SQL Users Guide and Reference*.

Warehouse Builder enables you to create the following PL/SQL types:

- PL/SQL Record types

  Record types enable you to define records in a package. A record is a composite data structure that contains multiple fields. Each field can have different datatypes. Use records to hold related items and pass them to subprograms using a single parameter.

For example, an EMPLOYEE record can contain details related to an employee such as ID, first name, last name, address, date of birth, date of joining, and salary. You can create a record type based on the EMPLOYEE record and use this record type to pass employee data between subprograms.

- Ref Cursor types

  REF cursor types enable you to define REF cursors within a package. REF cursors are not bound to a single query and can point to different result sets. Use REF cursors when you want to perform a query in one subprogram and process the results in another subprogram. REF cursors also enable you to pass query result sets between PL/SQL stored subprograms and various clients such as an OCI client or an Oracle Forms application.

  REF cursors are available to all PL/SQL clients. For example, you can declare a REF cursor in a PL/SQL host environment such as an OCI or Pro*C program, then pass it as an input host variable (bind variable) to PL/SQL. Application development tools such as Oracle Forms, which have a PL/SQL engine, can use cursor variables entirely on the client side. Or, you can pass cursor variables back and forth between a client and the database server through remote procedure calls.

- Nested Table types

  Use nested table types to define nested tables within a package. A nested table is an unordered set of elements, all of the same datatype. You can model multidimensional arrays by defining nested tables whose elements are also nested tables.

  For example, you can create a nested table type that can hold an arbitrary number of employee IDs. This nested table type can then be passed as a parameter to a subprogram that processes only the employee records contained in the nested table type.

Usage Scenario for PL/SQL Types

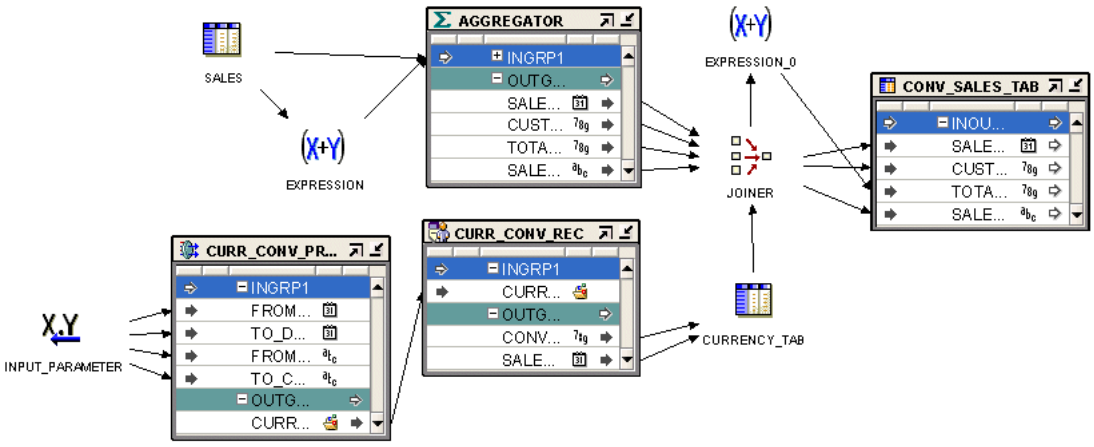
The SALES table stores the daily sales details of an organization that has offices across the world. This table contains the sale date, customer ID, product ID, amount sold, quantity sold, and currency in which the sale was made. The management wants to analyze global sales for a specified time period using a single currency, for example the US Dollar. Thus all sales values need to be converted to US Dollar. Since the currency exchange rates can change every day, the sales amounts need to be computed using the exchange rate of the sale currency on the sale date.

Solution Using PL/SQL Record Types

You can obtain the sales amount in the required currency by using a mapping that contains PL/SQL record types. A procedure is used to obtain the currency exchange rates on each day from a Web site. You then join this data with the data in the SALES table to obtain the sales in the required currency.

Figure 9–4 displays the mapping that you use to obtain a solution to the preceding scenario.

Figure 9–4 PL/SQL Record Type in a Mapping



In the mapping, the procedure CURR_CONV_PROC retrieves the currency exchange rates for a specified time period. It takes as input the sales currency, the currency to which the sales value needs to be converted, and the time period for which the currency conversion is required. The currency exchange values on each date are obtained from a Web site that contains this information. This data is stored in the record type CURR_CONV_REC, which is the output parameter of the function. This record type contains two attributes, the date and the currency conversion value on that date.

The currency exchange rates are stored in a table called CURRENCY_TAB. You then use a Joiner operator to join the SALES table and CURRECNY_TAB tables using the sale date as the Join Condition. In the Joiner operator, if the SALES table is represented as INGRP1 and the CURRENCY_TAB table as INGRP2, the Join Condition will be INGRP1.SALE_DATE = INGRP2.CONV_DATE.

Use the Expression operator to multiply the sales amount with the currency exchange rate to get the total sales in the required currency. The CONV_SALES_TAB stores the converted sales data.

Creating PL/SQL Types

You can create PL/SQL Types in the Project Explorer or the Global Explorer. For more details, see "Defining Custom Transformations" on page 9-5.

To create a PL/SQL Type, use the following steps:

1. From the Project Explorer, expand the Transformations node.

   To create a PL/SQL type in the Global Shared Library, from the Global Explorer, expand the Public Transformations node and then the Custom node.

2. Expand the package node under which you want to create the PL/SQL type.

3. Right-click **PL/SQL Types** and select **New**.

   The Welcome page of the Create PL/SQL Type Wizard is displayed. Click **Next** to proceed. The wizard guides you through the following pages:

   - Name and Description Page
   - Attributes Page
   - Return Type Page

- Summary Page

Name and Description Page

Use the Name and Description page to provide the name and an optional description for the PL/SQL type. Also use this page to select the type of PL/SQL type you want to create.

You can create any of the following PL/SQL types:

- PL/SQL Record Type

  Select this option to create a PL/SQL record type. A record type is a composite data structure whose attributes can have different data types. You can use a record type to hold related items and pass them to subprograms as a single parameter. For example, you can create an employee record whose attributes include employee ID, first name, last name, date of joining, and department ID.

- Ref Cursor Type

  Select this option to create a ref cursor. Ref cursors are like pointers to result sets. The advantage with a ref cursor is that it is not tied to any particular query.

- Nested Table Type

  Select this option to create a nested table. Nested tables represent sets of values. They are similar to one-dimensional arrays with no declared number of elements. Nested tables enable you to model multidimensional arrays by creating a nested table whose elements are also tables.

After specifying the name and selecting the type of PL/SQL type object to create, click **Next**.

Attributes Page

Use the Attributes page to define the attributes of the PL/SQL record type. You specify attributes only for PL/SQL record types. A PL/SQL record must have at least one attribute.

For each attribute, define the following:

- **Name:** The name of the attribute. The name should be unique within the record type.
- **Type:** The data type of the attribute. Select the data type from the drop-down list.
- **Length:** The length of the data type, for character data types.
- **Precision:** The total number of digits allowed for the attribute, for numeric data types.
- **Scale:** The total number of digits to the right of the decimal point, for numeric data types.
- **Seconds Precision:** The number of digits in the fractional part of the datetime field. It can be a number between 0 and 9. The Seconds Precision is used only for TIMESTAMP data types.

Click **Next** to proceed to the next step.

Return Type Page

Use the Return Type page to select the return type of the PL/SQL type. You must specify a return type when you create ref cursors and nested tables.

To define ref cursors:

The return type for a ref cursor can only be a PL/SQL record type. If you know the name of the PL/SQL record type, you can search for it by typing the name in the **Search For** field and clicking **Go**.

The area below the Search For field displays the available PL/SQL types. These PL/SQL types are grouped under the two nodes: Public and Private. Expand the Public node to view the PL/SQL types that are part of the Oracle shared library. The types are grouped by package name. The Private node contains PL/SQL types that are created as part of a package in an Oracle module. Only PL/SQL types that belong to the current project are displayed. Each Oracle module is represented by a node. Within the module, the PL/SQL types are grouped by the package to which they belong.

To define nested tables:

For nested tables, the return type can be a scalar data type or a PL/SQL record type. Select one of the following options on this page based on what the PL/SQL type returns:

- Select a scalar type as return type

  This option enables you to create a PL/SQL type that returns a scalar type. Use the drop-down list to select the data type.

- Select a PL/SQL record as return type

  This option enables you to create a PL/SQL type that returns a PL/SQL record type. If you know the name of the PL/SQL record type that is returned, type the name in the **Search For** field and click **Go**. The results of the search are displayed in the area below the option.

  You can also select the return type from the list of available types displayed. The area below this option contains two nodes: Public and Private. The Public node contains PL/SQL record types that are part of the Oracle Shared Library. The PL/SQL record types are grouped by the package to which they belong. The Private node contains the PL/SQL record types created as transformations in each Oracle module in the current project. These are grouped by module. Select the PL/SQL record type that the PL/SQL type returns.

Click **Next** to proceed with the creation of the PL/SQL type.

Summary Page

The Summary page displays the options that you have chosen on the wizard pages. Review the options. Click **Back** to modify any options. Click **Finish** to create the PL/SQL type.

Editing Transformation Properties

You can edit the definition of a transformation using the editors. Make sure you edit properties consistently. For example, if you change the name of a parameter, then you must also change its name in the implementation code.

Editing Function or Procedure Definitions

The Edit Function dialog enables you to edit function definitions. To edit a procedure definition, use the Edit Procedure dialog.

Use the following steps to edit functions or procedures:

1. From the Project Explorer, expand the Oracle module in which the transformation is created. Then expand the Transformations node.

   To edit a transformation that is part of the Global Shared Library, from the Global Explorer, expand the Public Transformations node, and then the Custom node.

2. Right-click the name of the function, procedure, or package you want to edit and select **Open Editor.**

   The Edit Function or Edit Procedure dialog is displayed. Use the following tabs to edit the function or procedure definition:

   - Name Tab on page 9-13
   - Parameters Tab on page 9-13
   - Implementation Tab on page 9-13

To edit a package, Warehouse Builder displays the Edit Transformation Library dialog. You can only edit the name and description of the package. You can edit the functions and procedures contained within the package using the steps used to edit functions or packages.

Name Tab

Use the Name tab to edit the name and description of the function or procedure. For functions, you can also edit the return data type.

Parameters Tab

Use the Parameters tab to edit, add, or delete new parameters for a function or procedure. You can also edit and define the attributes of the parameters. The contents of the Parameters tab are the same as that of the Parameters page of the Create Transformation Wizard. For more information about the contents of this page, see "Parameters Page" on page 9-7.

Implementation Tab

Use the Implementation tab to review the PL/SQL code for the function or procedure. Click **Code Editor** to edit the code. The contents of the Implementation tab are the same as that of the Implementation page of the Create Transformation Wizard. For more information on the contents of the Implementation page, see "Implementation Page" on page 9-8.

Editing PL/SQL Types

The Edit PL/SQL Type dialog enables you to edit the definition of a PL/SQL type. Use the following steps to edit a PL/SQL type:

1. From the Project Explorer, expand the Oracle module that contains the PL/SQL type. Then expand the Transformations node.

   To edit a PL/SQL type stored in the Global Shared Library, expand the Public Transformations node in the Global Explorer, and then the Custom node.

2. Expand the package that contains the PL/SQL type and then the PL/SQL Types node.

3. Right-click the name of the PL/SQL type that you want to edit and select **Open Editor**.

   The Edit PL/SLQ Type dialog is displayed. Use the following tabs to edit the PL/SQL type:

- Name Tab
- Attributes Tab
- Return Type Tab

Name Tab

The Name tab displays the name and the description of the PL/SQL type. Use this tab to edit the name or the description of the PL/SQL type.

To rename a PL/SQL type, select the name and enter the new name.

Attributes Tab

The Attributes tab displays details about the existing attributes of the PL/SQL record type. This tab is displayed for PL/SQL record types only. You can modify existing attributes, add new attributes, or delete attributes.

To add a new attribute, click the **Name** column of a blank row specify the details for the attribute. To delete an attribute, right-click the gray cell to the left the row that represents the attribute and select **Delete**.

Return Type Tab

Use the Return Type tab to modify the details of the return type of the PL/SQL type. For a ref cursor, the return type must be a PL/SQL record. For a nested table, the return type can be a PL/SQL record type or a scalar data type.

Importing PL/SQL

Use the Import Wizard to import PL/SQL functions, procedures, and packages into a Warehouse Builder project.

The following steps describe how to import PL/SQL packages from other sources into Warehouse Builder.

To import a PL/SQL function, procedure, or package:

1. From the Project Explorer, expand the project node and then Databases node.
2. Right-click an Oracle module node and select **Import**.

   Warehouse Builder displays the Import Metadata Wizard Welcome page.
3. Click **Next.**
4. Select **PL/SQL Transformation** in the Object Type field of the Filter Information page.
5. Click **Next.**

   The Import Metadata Wizard displays the Object Selection page.
6. Select a function, procedure, or package from the Available Objects list. Move the objects to the Selected Objects list by clicking the single arrow button to move a single object or the double arrow button to move multiple objects.
7. Click **Next.**

   The Import Metadata Wizard displays the Summary and Import page.
8. Verify the import information. Click **Back** to revise your selections.
9. Click **Finish** to import the selected PL/SQL transformations.

Warehouse Builder displays the Import Results page.

10. Click **OK** proceed with the import. Click **Undo** to cancel the import process.

    The imported PL/SQL information appears under the Transformations node of the Oracle node into which you imported the data.

When you use imported PL/SQL:

- You can edit, save, and deploy the imported PL/SQL functions and procedures.
- You cannot edit imported PL/SQL packages.
- Wrapped PL/SQL objects are not readable.
- Imported packages can be viewed and modified in the category property sheet.
- You can edit the imported package body but not the imported package specification.

10

Understanding Data Quality Management

Today, more than ever, organizations realize the importance of data quality. By ensuring that quality data is stored in your data warehouse or business intelligence application, you also ensure the quality of information for dependent applications and analytics.

Oracle Warehouse Builder offers a set of features that assist you in creating data systems that provide high quality information to your business users. You can implement a quality process that assesses, designs, transforms, and monitors quality. Within these phases, you will use specific functionality from Warehouse Builder to create improved quality information.

This chapter contains the following topics:

About the Data Quality Management Process

Quality data is crucial to decision-making and planning. The aim of building a data warehouse is to have an integrated, single source of data that can be used to make business decisions. Since the data is usually sourced from a number of disparate systems, it is important to ensure that the data is standardized and cleansed before loading into the data warehouse.

Warehouse Builder provides functionality that enables you to effectively manage data quality by assessing, transforming, and monitoring your data. The benefits of using Warehouse Builder for data management are as follows:

- Provides an end-to-end data quality solution
- Enables you to include data quality and data profiling as an integral part of your data integration process.
- Stores metadata regarding the quality of your data alongside your data. Both the data and the metadata are stored in the design repository.
- Automatically generates the mappings that you can use to correct data. These mappings are based on the business rules that you choose to apply to your data. and decision you make on how to correct data.

Phases in the Data Quality Life Cycle

Ensuring data quality involves the following phases:

- Quality Assessment
- Quality Design
- Quality Transformation
- Quality Monitoring

Figure 10–1 shows the phases involved in providing high quality information to your business users.

Figure 10–1 Phases Involved in Providing Quality Information

Quality Assessment

In the quality assessment phase, you determine the quality of the source data. The first step in this phase is to load the source data, which could be stored in different sources, into Warehouse Builder. You can import metadata and data from both Oracle and non-Oracle sources.

After you load the source data, you use data profiling to assess its quality. Data profiling is the process of uncovering data anomalies, inconsistencies, and redundancies by analyzing the content, structure, and relationships within the data. The analysis and data discovery techniques form the basis for data monitoring. For more information about data profiling, see "About Data Profiling" on page 10-3.

Quality Design

The quality design phase consists designing your quality processes. You can specify the legal data within a data object or legal relationships between data objects using data rules. For more information about data rules, see "About Data Rules" on page 10-16.

You also correct and augment your data. You can use data quality operators to correct and augment data. For more information, see "About Data Quality" on page 10-11.

As part of the quality design phase, you also design the transformations that ensure data quality. These transformations could be mappings that are generated by Warehouse Builder as a result of data profiling or mappings you create.

Quality Transformation

The quality transformation phase consists of running the correction mappings that are used to correct the source data.

Quality Monitoring

Data monitoring is the process of examining your data over time and alerting you when the data violates any business rules that are set. For more information about data monitoring, see "About Quality Monitoring" on page 10-17.

About Data Profiling

Data profiling is the first step for any organization to improve information quality and provide better decisions. It is a robust data analysis method available in Warehouse Builder that you can use to discover and measure defects in your data before you start working with it. Because of its integration with the ETL features in Warehouse Builder and other data quality features, such as data rules and built-in cleansing algorithms, you can also generate data cleansing and schema correction. This enables you to automatically correct any inconsistencies, redundancies, and inaccuracies in both the data and metadata.

Data profiling enables you to discover many important things about your data. Some common findings include the following:

- A domain of valid product codes
- A range of product discounts
- Columns that hold the pattern of an e-mail address
- A one-to-many relationship between columns
- Anomalies and outliers within columns
- Relations between tables even if they are not documented in the database

This section contains the following topics:

Uses of Data Profiling

Using the data profiling functionality in Warehouse Builder enables you to:

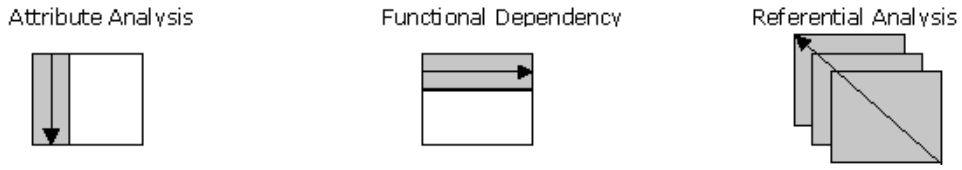
- Profile data from any source or combination of sources that Warehouse Builder can access.
- Explore data profiling results in tabular or graphical format.
- Drill down into the actual data related to any profiling result.
- Derive data rules, either manually or automatically, based on the data profiling results.

- Attach any data rule to a target object and select an action to perform if the rule fails.
- Create a data auditor from a data rule to continue monitoring the quality of data being loaded into an object.
- Derive quality indices such as six-sigma valuations.
- Profile or test any data rules you want to verify before putting in place.

Types of Data Profiling

Following the selection of data objects, determine the aspects of your data that you want to profile and analyze. As shown in Figure 10–2, data profiling offers three main types of analysis: attribute analysis, functional dependency, and referential analysis. You can also create custom profiling processes using data rules, allowing you to validate custom rules against the actual data and get a score of their accuracy.

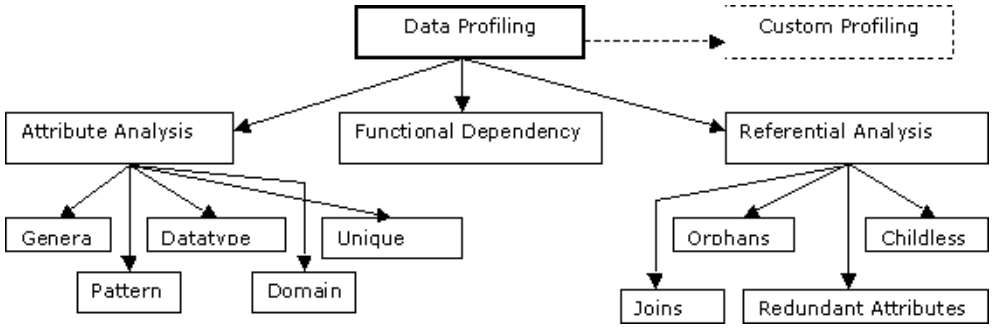
Figure 10–2 Three Types of Data Profiling



Attribute Analysis

Attribute analysis seeks to discover both general and detailed information about the structure and content of data stored within a given column or attribute. Attribute analysis looks for information about patterns, domains, data types, and unique values.

Figure 10–3 Data Profiling in Warehouse Builder



Pattern analysis attempts to discover patterns and common types of records by analyzing the string of data stored in the attribute. It identifies the percentages of your data that comply with a certain regular expression format pattern found in the attribute. Using these pattern results, you can create data rules and constraints to help clean up current data problems. Some commonly identified patterns include dates, e-mail addresses, phone numbers, and social security numbers.

Table 10–1 shows a sample attribute, Job Code, that could be used for pattern analysis.

Table 10–1 Sample Columns Used for Pattern Analysis

Job ID	Job Code
7	337-A-55
9	740-B-74
10	732-C-04
20	43-D-4

Table 10–2 shows the possible results from pattern analysis, where D represents a digit and X represents a character. After looking at the results and knowing that it is company policy for all job codes be in the format of DDD-X-DD, you can derive a data rule that requires all values in this attribute to conform to this pattern.

Table 10–2 Pattern Analysis Results

Job Code	% Occurred
DDD-X-DD	75%
DD-X-D	25%

Domain analysis identifies a domain or set of commonly used values within the attribute by capturing the most frequently occurring values. For example, the Status column in the Customers table is profiled and the results reveal that 90% of the values are among the following: "MARRIED", "SINGLE", "DIVORCED". Further analysis and drilling down into the data reveal that the other 10% contains misspelled versions of these words with few exceptions. Configuration of the profiling determines when something is qualified as a domain, so review the configuration before accepting domain values. You can then let Warehouse Builder derive a rule that requires the data stored in this attribute to be one of the three values that were qualified as a domain.

Data type analysis enables you to discover information about the data types found in the attribute. This type of analysis reveals metrics such as minimum and maximum character length values as well as scale and precision ranges. In some cases, the database column is of data type VARCHAR2, but the values in this column are all numbers. Then you may want to ensure that you only load numbers. Using data type analysis, you can have Warehouse Builder derive a rule that requires all data stored within an attribute to be of the same data type.

Unique key analysis provides information to assist you in determining whether or not an attribute is a unique key. It does this by looking at the percentages of distinct values that occur in the attribute. You might determine that attributes with a minimum of 70% distinct values should be flagged for unique key analysis. For example, using unique key analysis you could discover that 95% of the values in the EMP_ID column are unique. Further analysis of the other 5% reveals that most of these values are either duplicates or nulls. You could then derive a rule that requires that all entries into the EMP_ID column be unique and not null.

Functional Dependency

Functional dependency analysis reveals information about column relationships. This enables you to search for things such as one attribute determining another attribute within an object.

Table 10–3 shows the contents of the Employees table in which the attribute Dept Location is dependent on the attribute Dept Number. Note that the attribute Dept Number is not dependent on the attribute Dept Location.

Table 10–3 Employees Table

ID	Name	Salary	Dept Number	Dept Location
10	Alison	1000	10	SF
20	Rochnik	1000	11	London
30	Meijer	300	12	LA
40	John	500	13	London
50	George	200	13	London
60	Paul	600	13	London
70	Ringo	100	13	London
80	Yoko	600	13	London
90	Jones	1200	10	SF

Referential Analysis

Referential analysis attempts to detect aspects of your data objects that refer to other objects. The purpose behind this type of analysis is to provide insight into how the object you are profiling is related or connected to other objects. Because you are comparing two objects in this type of analysis, one is often referred to as the parent object and the other as the child object. Some of the common things detected include orphans, childless objects, redundant objects, and joins. Orphans are values that are found in the child object, but not found in the parent object. Childless objects are values that are found in the parent object, but not found in the child object. Redundant attributes are values that exist in both the parent and child objects.

Table 10–4 and Table 10–5 show the contents of two tables that are candidates for referential analysis. Table 10–4 is the child object and Table 10–5 is the parent object.

Table 10–4 Employees Table (Child)

ID	Name	Dept. Number	City
10	Alison	17	NY
20	Rochnik	23	SF
30	Meijer	23	SF
40	Jones	15	SD

Table 10–5 Department Table (Parent)

Dept. Number	Location
17	NY
18	London
20	SF
23	SF
55	HK

Referential analysis of these two objects would reveal that Dept. Number 15 from the Employees table is an orphan and Dept. Numbers 18, 20, and 55 from the Department table are childless. It would also reveal a join on the Dept. Number column.

Based on these results, you could derive referential rules that determine the cardinality between the two tables.

Data Rule Profiling

In addition to attribute analysis, functional dependency, and referential analysis, Warehouse Builder offers data rule profiling. Data rule profiling enables you to create rules to search for profile parameters within or between objects.

This is very powerful as it enables you to validate rules that apparently exist and are defined by the business users. By creating a data rule, and then profiling with this rule you can verify if the data actually complies with the rule, and whether or not the rule needs amending or the data needs cleansing.

For example, you could create a rule that Income = Salary + Bonus for the Employee table shown in Table 10–6. You can then catch errors such as the one for employee Alison.

Table 10–6 Sample Employee Table

ID	Name	Salary	Bonus	Income	
10	Alison	1000	50	1075	X
20	Rochnik	1000	75	1075	
30	Meijer	300	35	335	
40	Jones	1200	500	1700	

How to Perform Data Profiling

Data profiling is, by definition, a resource-intensive process that requires forethought and planning. It analyzes data and columns and performs many iterations to detect defects and anomalies in your data. So it warrants at least some forethought and planning in order to be as effective as possible.

Before beginning data profiling, you should first identify the data objects that you want to target. Instead of profiling everything, choose objects that are deemed crucial. You should not select an entire source system for profiling at the same time. Not only is it a waste of resources, but it is also often unnecessary. Select areas of your data where quality is essential and has the largest fiscal impact.

For example, you have a data source that contains five tables: Customers, Regions, Orders, Products, and Promotions. You decide that the two most important tables with respect to data quality are Customers and Orders. The Customers table is known to contain many duplicate and erroneous entries that cost your company money on wasted marketing efforts. The Orders table is known to contain data about orders in an incorrect format. In this case, you would select only these two tables for data profiling.

After you have chosen the object you want to profile, use the following steps to guide you through the profiling process:

1. Import or Select the Metadata
2. Create a Data Profile
3. Profile the Data
4. View Profile Results and Derive Data Rules
5. Generating Corrections
6. Define and Edit Data Rules Manually

7. Generate, Deploy, and Execute

The data profiling process ends at step 4. Steps 5 to 7 are optional and can be performed if you want to perform data correction after the data profiling.

Import or Select the Metadata

Data profiling requires the profiled objects to be present in the project in which you are performing data profiling. Ensure that these objects are either imported into this project or created in it. Also ensure that the data is loaded into the objects. Having the data loaded is essential to data profiling.

Also, because data profiling uses mappings to run the profiling, you must ensure that all locations that you are using are registered. Data profiling attempts to register your locations. If, for some reason, data profiling cannot register your locations, you will need to explicitly register the locations before you begin profiling.

Note: You can only profile data in the default configuration.

Create a Data Profile

After your system is set up, you can create a data profile. A data profile is a metadata object in the Warehouse Builder repository and you create in the navigation tree. It contains the definitions and settings necessary for profiling objects. It includes the set of data objects you want profiled, the settings controlling the profiling operations, the results returned after you profile the data, and correction information (if you decide to use these corrections). For more information about creating data profiles, see "Using Data Profiles" on page 20-2.

Profile the Data

After you have created a data profile, you can open it in the Data Profile Editor to profile the data or review profile results from a previous run. Data profiling is achieved by performing deep scans of the selected objects. This can be a time-consuming process, depending on the number of objects and type of profiling you are running. However, profiling is run as an asynchronous job, and the client can be closed during this process. You will see the job running in the job monitor and Warehouse Builder prompts you when the job is complete.

The results are generated and can be viewed from the Data Profile Editor as soon as they are available. For more information about how to profile data, see "Profiling the Data" on page 20-13.

You can, and should, configure the profile before running it if there are specific types of analysis you do, or do not, want to run. Configuration of the profile and its objects is possible at the following levels:

- the entire profile (all the objects it contains)
- an individual object (for example, a table)
- a single column in a a table

For example, if you know you only have one problematic column in a table and you already know that most of the records should conform to values within a certain domain, then you can focus your profiling resources on domain discovery and analysis. By narrowing down the type of profiling necessary, you use less resources and obtain the results faster. For more information about configuring data profiles, see "Property Inspector" on page 20-7.

View Profile Results and Derive Data Rules

The profiling results contain a variety of analytical and statistical information about the data profiled. You can immediately drill down into anomalies and view the data that caused them. You can then determine what data must be corrected. For more information about viewing profiling results, see "Viewing the Results" on page 20-15.

Based on your decisions, you can derive data rules. Data rules are used to ensure that only values compliant with the data rules are allowed within a data object. Data rules will form the basis for correcting or removing data if you decide to cleanse the data. You can also use data rules to report on non-compliant data. For more information about deriving data rules, see "Deriving Data Rules" on page 20-27.

Generating Corrections

After you have derived data rules from the profiling results, you can create the schema and mapping corrections. The schema correction creates scripts that can be used to create a corrected set of source data objects with the derived data rules applied.

The mapping correction creates new correction mappings to take your data from the source objects and load them into new objects. For more information about creating schema and mapping corrections, see "Correcting Schemas and Cleansing Data" on page 20-29.

Define and Edit Data Rules Manually

Data rules can be derived or manually created. Before and after you have created the corrections, you can define additional data rules manually. For more information about defining and editing data rules manually, see "Creating Data Rule Folders" on page 20-37.

Generate, Deploy, and Execute

Finally, you can generate, deploy, and execute the correction mappings and data rules. After you run the correction mappings with the data rules, your data is corrected. The derived data rules remain attached to the objects in the corrected schema for optional use in data monitors.

About Six Sigma

Warehouse Builder provides Six Sigma results embedded within the other data profiling results to provide a standardized approach to data quality.

What is Six Sigma?

Six Sigma is a methodology that attempts to standardize the concept of quality in business processes. It achieves this goal by statistically analyzing the performance of business processes. The goal of Six Sigma is to improve the performance of these processes by identifying the defects, understanding them, and eliminating the variables that cause these defects.

Six Sigma metrics give a quantitative number for the number of defects in each 1,000,000 opportunities. The term "opportunities" can be interpreted as the number of records. The perfect score is 6.0. The score of 6.0 is achieved when there are only 3.4 defects in each 1,000,000 opportunities. The score is calculated using the following formula:

- Defects Per Million Opportunities (DPMO) = (Total Defects / Total Opportunities) * 1,000,000

- Defects (%) = (Total Defects / Total Opportunities)* 100%
- Yield (%) = 100 - %Defects
- Process Sigma = NORMSINV(1-((Total Defects) / (Total Opportunities))) + 1.5

  where NORMSINV is the inverse of the standard normal cumulative distribution.

Six Sigma Metrics for Data Profiling

Six Sigma metrics are also provided for data profiling in Warehouse Builder. When you perform data profiling, the number of defects and anomalies discovered are shown as Six Sigma metrics. For example, if data profiling finds that a table has a row relationship with a second table, the number of records in the first table that do not adhere to this row-relationship can be described using the Six Sigma metric.

Six Sigma metrics are calculated for the following measures in the Data Profile Editor:

- **Aggregation:** For each column, the number of null values (defects) to the total number of rows in the table (opportunities).
- **Data Types:** For each column, the number of values that do not comply with the documented data type (defects) to the total number of rows in the table (opportunities).
- **Data Types:** For each column, the number of values that do not comply with the documented length (defects) to the total number of rows in the table (opportunities).
- **Data Types:** For each column, the number of values that do not comply with the documented scale (defects) to the total number of rows in the table (opportunities).
- **Data Types:** For each column, the number of values that do not comply with the documented precision (defects) to the total number of rows in the table (opportunities).
- **Patterns:** For each column, the number of values that do not comply with the common format (defects) to the total number of rows in the table (opportunities).
- **Domains:** For each column, the number of values that do not comply with the documented domain (defects) to the total number of rows in the table (opportunities).
- **Referential:** For each relationship, the number of values that do not comply with the documented foreign key (defects) to the total number of rows in the table (opportunities).
- **Referential:** For each column, the number of values that are redundant (defects) to the total number of rows in the table (opportunities).
- **Unique Key:** For each unique key, the number of values that do not comply with the documented unique key (defects) to the total number of rows in the table (opportunities).
- **Unique Key:** For each foreign key, the number of rows that are childless (defects) to the total number of rows in the table (opportunities).
- **Data Rule:** For each data rule applied to the data profile, the number of rows that fail the data rule to the number of rows in the table.

About Data Quality

Warehouse Builder enables you to automatically create correction mappings based on the results of data profiling. On top of these automated corrections that make use of the underlying Warehouse Builder architecture for data quality, you can create your own data quality mappings.

Warehouse Builder provides functionality that enables you to ensure data quality. During transformation of the source data, you can use the following operators to ensure data quality:

- Match-Merge Operator

  See "About the Match-Merge Operator" on page 10-11

- Name and Address Operator

  See "About the Name and Address Operator" on page 10-11

About the Match-Merge Operator

Match-Merge is a data quality operator that identifies matching records and merges them into a single record. Master data management working on various systems will make use of this operator to ensure that records are created and matched with a master record.

You define the business rules that the Match-Merge operator uses to identify records that refer to the same data. After the operator identifies matching records, it merges them into a single, consolidated record.

You have the following options for using a match-merge operator:

- **Define a new match-merge operator:** Drag the Match-Merge operator from the Palette onto the mapping. The Mapping Editor launches the MATCHMERGE Wizard.
- **Edit an existing match-merge operator:** Right-click the operator and select **Open Details**. The Mapping Editor opens the MATCHMERGE Editor.

About the Name and Address Operator

Warehouse Builder enables you to perform name and address cleansing on data using the Name and Address operator.

The Name and Address operator identifies and corrects errors and inconsistencies in name and address source data by comparing input data to the data libraries supplied by third-party name and address cleansing software vendors. You can purchase the data libraries directly from these vendors.

> **Note:** The Name and Address operator requires separate licensing and installation of third-party name and address cleansing software. Refer to the *Oracle Warehouse Builder Installation and Administration Guide* for more information.

You have the following options for using a Name and Address operator:

- **Define a new Name and Address operator:** Drag the operator from the Toolbox onto the mapping. The Mapping Editor launches the Name and Address wizard.

- **Edit an existing Name and Address operator:** Right-click the operator and select **Open Details**. The Mapping Editor opens the Name and Address Editor.

The errors and inconsistencies corrected by the Name and Address operator include variations in address formats, use of abbreviations, misspellings, outdated information, inconsistent data, and transposed names. The operator fixes these errors and inconsistencies by:

- Parsing the name and address input data into individual elements.
- Standardizing name and address data, using standardized versions of nicknames and business names and standard abbreviations of address components, as approved by the postal service of the appropriate country. Standardized versions of names and addresses facilitate matching and householding, and ultimately help you obtain a single view of your customer.
- Correcting address information such as street names and city names. Filtering out incorrect or undeliverable addresses can lead to savings on marketing campaigns.
- Augmenting names and addresses with additional data such as gender, ZIP+4, country code, apartment identification, or business and consumer identification. You can use this and other augmented address information, such as census geocoding, for marketing campaigns that are based on geographical location.

  Augmenting addresses with geographic information facilitates geography-specific marketing initiatives, such as marketing only to customers in large metropolitan areas (for example, within an *n*-mile radius from large cities); marketing only to customers served by a company's stores (within an *x*-mile radius from these stores). Oracle Spatial, an option with Oracle Database, and Oracle Locator, packaged with Oracle Database, are two products that you can use with this feature.

The Name and Address operator also enables you to generate postal reports for countries that support address correction and postal matching. Postal reports often qualify you for mailing discounts. For more information, see "About Postal Reporting" on page 10-15.

Example: Correcting Address Information

This example follows a record through a mapping using the Name and Address operator. This mapping also uses a Splitter operator to demonstrate a highly recommended data quality error handling technique.

Example Input

In this example, the source data contains a Customer table with the row of data shown in Table 10–7.

Table 10–7 Sample Input to Name and Address Operator

Address Column	Address Component
Name	Joe Smith
Street Address	8500 Normandale Lake Suite 710
City	Bloomington
ZIP Code	55437

The data contains a nickname, a last name, and part of a mailing address, but it lacks the customer's full name, complete street address, and the state in which he lives. The

data also lacks geographic information such as latitude and longitude, which can be used to calculate distances for truckload shipping.

Example Steps

This example uses a mapping with a Name and Address operator to cleanse name and address records, followed by a Splitter operator to load the records into separate targets depending on whether they were successfully parsed. This section explains the general steps required to design such a mapping.

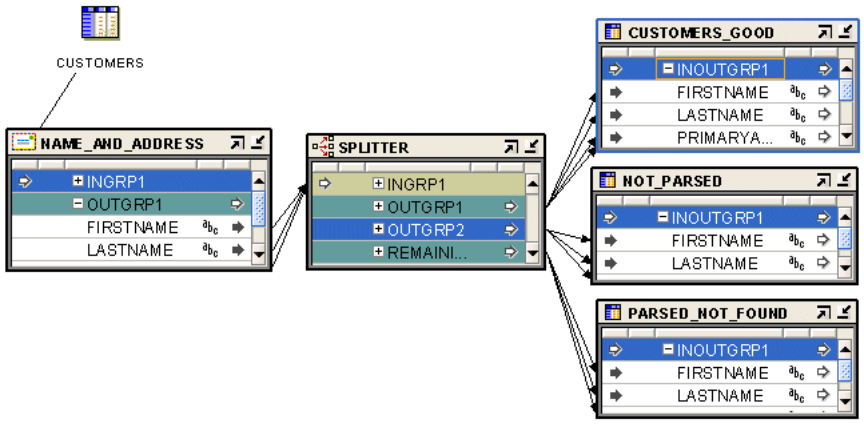
To make the listed changes to the sample record:

1. In the Mapping Editor, begin by adding the following operators to the canvas:
   - A CUSTOMERS table from which you extract the records. This is the data source. It contains the data in "Example Input" on page 10-12.
   - A Name and Address operator. This action launches the Name and Address Wizard. Follow the steps of the wizard.
   - A Splitter operator. For information on using this operator, see "Splitter Operator" on page 26-32.
   - Three target operators into which you load the successfully parsed records, the records with parsing errors, and the records whose addresses are parsed but not found in the postal matching software.
2. Map the attributes from the CUSTOMERS table to the Name and Address operator ingroup. Map the attributes from the Name and Address operator outgroup to the Splitter operator ingroup.

   You are not required to use the Splitter operator, but it provides an important function in separating good records from problematic records.
3. Define the split conditions for each of the outgroups in the Splitter operator and map the outgroups to the targets.

Figure 10–4 shows a mapping designed for this example. The data is mapped from the source table to the Name and Address operator, and then to the Splitter operator. The Splitter operator separates the successfully parsed records from those that have errors. The output from OUTGRP1 is mapped to the CUSTOMERS_GOOD target. The split condition for OUTGRP2 is set such that records whose `Is Parsed` flag is `False` are loaded to the NOT_PARSED target. That is, the Split Condition for OUTGRP2 is set as INGRP1.ISPARSED='F'. The Records in the REMAINING_RECORDS group are successfully parsed, but their addresses are not found by the postal matching software. These records are loaded to the PARSED_NOT_FOUND target.

Figure 10–4 Name and Address Operator Used with a Splitter Operator in a Mapping



Example Output

If you run the mapping designed in this example, the Name and Address operator standardizes, corrects, and completes the address data from the source table. In this example, the target table contains the address data as shown in Table 10–8. Compare it with the input record from Table 10–7 on page 10-12.

Table 10–8 Sample Output from Name and Address Operator

Address Column	Address Component
First Name Standardized	JOSEPH
Last Name	SMITH
Primary Address	8500 NORMANDALE LAKE BLVD
Secondary Address	STE 710
City	BLOOMINGTON
State	MN
Postal Code	55437-3813
Latitude	44.849194
Longitude	-093.356352
Is Parsed	True or False. Indicates whether a record can be separated into individual elements.
Is Good Name	True or False. Indicates whether the name was found in a postal database.
Is Good Address	True or False. Indicates whether the address was found in a postal database or was parsed successfully.
Is Found	True or False. Indicates whether the address was found in a postal database.
Name Warning	True or False. Indicates whether problems occurred in parsing the name.
Street Warning	True or False. Indicates whether problems occurred in parsing the address.
City Warning	True or False. Indicates whether problems occurred in parsing the city name.

In this example, the following changes were made to the input data:

- Joe Smith was separated into separate columns for `First_Name_Standardized` and `Last_Name`.
- Joe was standardized into JOSEPH and Suite was standardized into STE.
- Normandale Lake was corrected to NORMANDALE LAKE BLVD.
- The first portion of the postal code, 55437, was augmented with the ZIP+4 code to read 55437-3813.
- Latitude and longitude locations were added.
- The records were tested in various ways, and the good records are directed to a different target from the ones that have problems.

Handling Errors in Name and Address Data

Name and Address parsing, like any other type of parsing, depends on identification of keywords and patterns containing those keywords. Free-form name and address data difficult to parse because the keyword set is large and it is never 100% complete. Keyword sets are built by analyzing millions of records, but each new data set is likely to contain some undefined keywords.

Because most free-form name and address records contain common patterns of numbers, single letters, and alphanumeric strings, parsing can often be performed based on just the alphanumeric patterns. However, alphanumeric patterns may be ambiguous or a particular pattern may not be found. Name and Address parsing errors set parsing status codes that you can use to control data mapping.

Since the criteria for quality vary among applications, numerous flags are available to help you determine the quality of a particular record. For countries with postal matching support, use the `Is Good Group` flag, because it verifies that an address is a valid entry in a postal database. Also use the `Is Good Group` flag for U.S. Coding Accuracy Support System (CASS) and Canadian Software Evaluation and Recognition Program (SERP) certified mailings.

Unless you specify postal reporting, an address does not have to be found in a postal database to be acceptable. For example, street intersection addresses or building names may not be in a postal database, but they may still be deliverable. If the `Is Good Group` flag indicates failure, additional error flags can help determine the parsing status.

The `Is Parsed` flag indicates success or failure of the parsing process. If `Is Parsed` indicates parsing success, you may still wish to check the parser warning flags, which indicate unusual data. You may want to check those records manually.

If `Is Parsed` indicates parsing failure, you must preserve the original data to prevent data loss.

Use the Splitter operator to map successful records to one target and failed records to another target.

About Postal Reporting

All address lists used to produce mailings for discounted automation postal rates must be matched by postal report-certified software. Certifications depend on the third-party vendors of name and address software and data. The certifications may include the following:

- **United States Postal Service:** Coding Accuracy Support System (CASS)

- **Canada Post:** Software Evaluation and Recognition Program (SERP)
- **Australia Post:** Address Matching Approval System (AMAS)

United States Postal Service CASS Certification

The Coding Accuracy Support System (CASS) was developed by the United States Postal Service (USPS) in cooperation with the mailing industry. The system provides mailers a common platform to measure the quality of address-matching software, focusing on the accuracy of five-digit ZIP Codes, ZIP+4 Codes, delivery point codes, and carrier route codes applied to all mail. All address lists used to produce mailings for automation rates must be matched by CASS-certified software.

To meet USPS requirements, the mailer must submit a CASS report in its original form to the USPS.

Canada Post SERP Certification

Canada Post developed a testing program called Software Evaluation and Recognition Program (SERP), which evaluates software packages for their ability to validate, or validate and correct, mailing lists to Canada Post requirements. Postal programs that meet SERP requirements are listed on the Canada Post Web site.

Canadian postal customers who use Incentive Lettermail, Addressed Admail, and Publications Mail must meet the Address Accuracy Program requirements. Customers can obtain a Statement of Accuracy by comparing their databases to Canada Post's address data.

Australia Post AMAS Certification

The Address Matching Approval System (AMAS) was developed by Australia Post to improve the quality of addressing. It provides a standard by which to test and measure the ability of address-matching software to:

- Correct and match addresses against the Postal Address File (PAF)
- Append a unique Delivery Point Identifier (DPID) to each address record, which is a step toward barcoding mail.

AMAS allows companies to develop address matching software which:

- Prepares addresses for barcode creation
- Ensures quality addressing
- Enables qualification for discounts on PreSort letter lodgements

PreSort Letter Service prices are conditional upon customers using AMAS Approved Software with Delivery Point Identifiers (DPIDs) being current against the latest version of the PAF.

A declaration that the mail was prepared appropriately must be made when using the Presort Lodgement Document, available from post offices.

About Data Rules

Data rules are definitions for valid data values and relationships that can be created in Warehouse Builder. They determine legal data within a table or legal relationships between tables. Data rules help ensure data quality. They can be applied to tables, views, dimensions, cubes, materialized views, and external tables. Data rules are used in many situations including data profiling, data and schema cleansing, and data auditing.

The metadata for a data rule is stored in the repository. To use a data rule, you apply the data rule to a data object. For example, you create a data rule called `gender_rule` that specifies that valid values are 'M' and 'F'. You can apply this data rule to the `emp_gender` column of the `Employees` table. Applying the data rule ensures that the values stored for the `emp_gender` column are either 'M' or 'F'. You can view the details of the data rule bindings on the Data Rule tab of the Data Object Editor for the `Employees` table.

There are two ways to create a data rule. A data rule can be derived from the results of data profiling, or it can be created using the Data Rule Wizard. For more information about data rules, see "Using Data Rules" on page 20-35.

About Quality Monitoring

Quality monitoring builds on your initial data profiling and data quality initiatives. It enables you to monitor the quality of your data over time. You can define the business rules to which your data should adhere.

To monitor data using Warehouse Builder you need to create data auditors. Data auditors ensure that your data complies with the business rules you defined. You can define the business rules that your data should adhere to using a feature called data rules.

About Data Auditors

Data auditors are processes that validate data against a set of data rules to determine which records comply and which do not. Data auditors gather statistical metrics on how well the data in a system complies with a rule, and they can off-load defective data into auditing and error tables.

Data auditors have thresholds that allow you to create logic based on the fact that too many non-compliant records can divert the process flow into an error or notification stream. Based on this threshold, the process can choose actions. Also the audit result can be captured and stores for analysis purposes.

Data auditors can be deployed and executed ad-hoc, but they are typically run to monitor the quality of the data in an operational environment like a data warehouse or ERP system and, therefore, can be added to a process flow and scheduled.

When executed, the data auditor sets several output values. One of these output values is called the audit result. If the audit result is 0, then there were no errors. If the audit result is 1, at least one error occurred. If the audit result is 2, then at least one data rule failed to meet the specified error threshold. Data auditors also set the actual measured values such as Error Percent and Six Sigma values. For more information about using data auditors, see "Using Data Auditors" on page 20-41.

Data auditors are a very important tool in ensuring data quality levels are up to the standards set by the users of the system. It also helps determine spikes in bad data allowing events to the tied to these spikes.

You can use the Metadata Loader (MDL) utilities to import or export metadata related to data auditors. For more information about using the MDL utilities, see Chapter 33, "Importing and Exporting with the Metadata Loader (MDL)".

11

Deploying to Target Schemas and Executing ETL Logic

This chapter contains the following topics:

About Deployment

Deployment is the process of creating physical objects in a target location from the logical objects in a Warehouse Builder repository.

Deploying a mapping or a process flow includes these steps:

- Generate the PL/SQL, SQL*Loader, or ABAP script, if necessary.
- Copy the script from the Design Center to the Control Center. Also copy SQL*Loader control files to the Control Center.
- Deploy any new connectors; that is, create the database links and database directories between locations.

After deploying a mapping or a process flow, you must explicitly start the scripts, as described in "Starting the ETL Process" on page 29-6.

You can deploy only those objects for which you have the `COMPILE` privilege. By default, you have this privilege on all objects in the repository. However, the repository owner may have instituted a different security policy.

You can deploy directly from the Design Center navigation tree or using the Control Center Manager.

Note: Always maintain objects using Warehouse Builder. Do not modify the deployed, physical objects manually in SQL. Otherwise, the logical objects and the physical objects will not be synchronized, which may cause unpredictable results.

Note: Whenever you deploy an object, Warehouse Builder automatically saves all changes to all design objects to the repository. You can choose to display a warning message by selecting **Prompt for commit** on the Preferences dialog box.

What is a Control Center?

A Control Center stores detailed information about every deployment, which you can access either by object or by job, including:

- The current deployment status of each object.
- A history of all deployment attempts for each object.
- A history of all ETL start attempts for each mapping and process flow.
- A complete log of messages from all deployment jobs.

A Control Center is implemented as a schema in the same database as the target location. Each repository has a default Control Center, which was created in the schema of the repository owner. For example, the `REP_OWNER` repository owner has a schema named `REP_OWNER` that stores the metadata from both the Design Center and the default Control Center.

You can use the default Control Center to deploy to the local system, or you can create additional Control Centers for deploying to different systems. Only one Control Center is active at any time.

The Control Center Manager offers a comprehensive deployment console that enables you to view and manage all aspects of deployment. It provides access to the information stored in the active Control Center.

You can also access deployment data reports using the Repository Browser, as described in Chapter 30, "Auditing Deployments and Executions".

To create a new Control Center:

1. In the Connection Explorer, right-click Control Centers and choose **New**.

   The Create Control Center dialog box is displayed.

2. Complete the dialog box. Click the **Help** button for additional details.

You can also create a Control Center using the Create Configuration wizard.

To make a Control Center active:

1. Create or edit a configuration so that it uses the Control Center.

   Refer to "Creating a New Configuration" on page 3-11.

2. Activate that configuration.

   Refer to "Activating a Configuration" on page 3-11.

Configuring the Physical Details of Deployment

Warehouse Builder separates the logical design of the objects from the physical details of the deployment. It creates this separation by storing the physical details in configuration parameters. An object called a **configuration** stores all of the configuration settings. You can create a different configuration for each deployment location, with different settings for the object parameters in each one.

Before deployment, be sure to check the configuration of the target objects, the mappings, and the modules.

For an object to be deployable:

- Its target location must be fully defined, valid, and selected for the object's module.
- Its Deployable parameter must be selected, which it is by default.

- It must validate and generate without errors.

See Also: "Creating Additional Configurations" on page 3-11

Deployment Actions

As soon as you define a new object in the Design Center, the object is listed in the Control Center Manager. Each object has a default deployment action, which you can display. The default is set by the previous action and varies depending on the type of object. You can override the default by choosing a different deployment action in the Control Center Manager.

These are the deployment actions:

- **Create**: Creates the object in the target location. If an object with that name already exists, then an error may result.
- **Upgrade**: Modifies the object without losing data, if possible. You can undo and redo an upgrade. This action is not available for some object types, such as schedules.
- **Drop**: Deletes the object from the target location.
- **Replace**: Deletes and recreates the object. This action is quicker than Upgrade, but it deletes all data.

The Deployment Process

During the life cycle of a data system, you typically will take these steps in the deployment process:

1. Select a configuration with the object settings and the Control Center that you want to use.
2. Deploy objects to the target location. You can deploy them individually, in stages, or all at once.
3. Review the results of the deployment. If an object fails to deploy, then fix the problem and try again.
4. Start the ETL process.
5. Revise the design of target objects to accommodate user requests, changes to the source data, and so forth.
6. Set the deployment action on the modified objects to Upgrade or Replace.
7. Repeat these steps.

Note: Warehouse Builder automatically saves all changes to the repository before deployment.

About Schedules

Use schedules to plan when and how often to execute operations that you designed within Warehouse Builder. You can apply schedules to mappings and process flows that you want to execute in an Oracle Database, version 10*g* or higher.

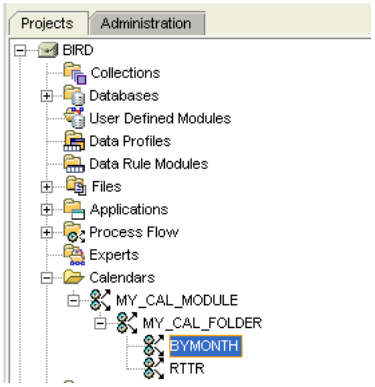
When you are in the development phase of using Warehouse Builder, you may not want to schedule mappings and process flows but rather start and stop them

immediately from a Control Center as described in Chapter 29, "Deploying Target Systems".

You can define schedules to execute once or to execute repeatedly based on an interval you define in the user interface. For each schedule you define, Warehouse Builder generates codes that follows the iCal calendaring standards.

Schedules are defined in the context of projects and contained in schedule modules under the **Schedules** node on the Project Explorer as shown in Figure 11–1.

Figure 11–1 Schedules on the Project Explorer



For every new project you create, Warehouse Builder creates a default schedule module, `MY_CAL_MODULE`. Create schedules under the default module or create a new module by right- clicking the **Schedules** node and selecting **New...**

Deploying Warehouse Builder Schedules to Oracle Workflow

To successfully deploy Warehouse Builder schedules to Oracle Workflow, ensure access to the correct version of Oracle Workflow as described in the Oracle Warehouse Builder Installation and Administration Guide.

Ensure that the target location has the 'CREATE SYNONYM' system privilege. If the Evaluation Location is specified or the deployment location references a different database instance from Control Center schema, then the deployment location must have the 'CREATE DATABASE LINK' system privilege.

Process for Defining and Using Schedules

1. To create a module to contain schedules, right-click the Schedules node and select **New**.
2. To create a schedule, right-click a schedule module and select **New**.

   Warehouse Builder launches the Schedule Wizard.
3. On the **Name and Description** page, type a name for the schedule that is 24 characters or less.

   The rules for most Warehouse Builder objects is that physical names can be 1 to 30 alphanumeric characters and business names can be 1 to 2000 alphanumeric characters.
4. Follow the instructions in the Schedule Wizard.

Use the wizard to specify values for Start and End Dates and Times, Frequency Unit, and Repeat Every. When you complete the wizard, Warehouse Builder saves the schedule under the schedule module you selected.

5. On the Project Explorer, right-click the schedule you created with the wizard and select **Open Editor.**

   Warehouse Builder displays the schedule editor. Review your selections and view the list of calculated execution times. For complex schedules, you can now enter values for the By Clauses.

6. To apply a schedule to a mapping or process flow, right-click the object in the Project Explorer and select **Configure.** In the Referred Calendar field, select the Ellipsis button to view a list of existing schedules.

   For any mapping or process flow you want to schedule, the physical name must be 25 characters or less and the business name must be 1995 characters or less. This restriction enables Warehouse Builder to append to the mapping name the suffix _ *job* and other internal characters required for deployment and execution.

7. Deploy the schedule.

   Recall that when you deploy a mapping, for example, you also need to deploy any associated mappings and process flows and any new target data objects. Likewise, you should also deploy any associated schedules.

   When properly deployed with its associated objects, the target schema executes the mapping or process flow based on the schedule you created.

Deploying Objects

Deployment is the process of creating physical objects in a target location from the metadata using your generated code. You can deploy an object from the Project Explorer or using the Control Center Manager. Warehouse Builder automatically validates and generates the object.

Deployment from the Project Explorer is restricted to the default action, which may be set to Create, Replace, Drop, or Update. To override the default action, use the Control Center Manager, which provides full control over the deployment process.

To deploy from the Project Explorer:

Select the object and click the Deploy icon on the toolbar.

Status messages appear at the bottom of the Design Center window. For notification that deployment is complete, select **Show Deployment Completion Messages** in your preferences before deploying.

To open the Control Center Manager:

1. Open a project.
2. Choose **Control Center Manager** from the Tools menu.

   If this menu choice is not available, then check that the appropriate configuration and Control Center are active. Refer to "Creating Additional Configurations" on page 3-11.

Starting ETL Jobs

ETL is the process of extracting data from its source location, transforming it as defined in a mapping, and loading it into target objects. When you start ETL, you

submit it as a job to the Warehouse Builder job queue. The job can start immediately or at a scheduled time, if you use a scheduler such as the one in Oracle Enterprise Manager. Like deployment, you can start ETL from the Project Explorer or using the Control Center Manager. You can also start ETL using tools outside of Warehouse Builder that execute SQL scripts.

To start ETL from the Project Explorer:

Select a mapping or a process flow, then choose **Start** from the Design menu.

Viewing the Data

After completing ETL, you can easily check any data object in Warehouse Builder to verify that the results are as you expected.

To view the data:

In the Project Explorer, right-click the object and select **Data**. The Data Viewer will open with the contents of the object.

Part II

Data Modeling Reference

This part contains the following chapters:

12

Reference for Using Oracle Data Objects

After you finish designing your data warehouse or data mart, you are ready to design your target system using Warehouse Builder. Most of your target schema modelling takes place within the Data Object Editor. This chapter shows you how to use the Data Object Editor to create data objects.

This chapter contains the following topics:

Using the Data Object Editor to Edit Oracle Data Objects

You edit a relational, dimensional, or business intelligence (item folders and business areas only) objects using the Data Object Editor. Use one of the following methods to open the editor for a data object:

- In the Project Explorer, double-click the data object that you want to edit.

  For example, to edit the SALES cube, double-click the SALES cube in the Project Explorer.

- In the Project Explorer, right-click the data object to be edited and select **Open Editor**.

  For example, to edit the SALES cube, right-click the SALE cube in the Project Explorer and select **Open Editor**.

After the Data Object Editor is displayed, use the tabs in the Details panel to edit the data object definition. For information about the Data Object Editor tabs for each data object, refer to the following sections:

- Creating Cubes on page 13-20
- Editing a Business Area on page 15-12
- Editing an Item Folder on page 15-5
- Creating Table Definitions on page 12-13
- Creating View Definitions on page 12-17
- Creating Materialized View Definitions on page 12-20

Using Constraints

Constraints enable you to enforce the business rules you want to associate with the information in a database. They prevent the entry of invalid data into tables. For more information about types of constraints, see "About Constraints" on page 4-22.

Creating Constraints

Use the Constraints tab of the Data Object Editor to create constraints. You can create the following types of constraints: primary key, foreign key, unique key, and check constraints.

Use the following steps to create constraints on a table, view, or materialized view:

1. Open the Data Object Editor for the data object to which you want to add constraints.

   In the Project Explorer, double-click the data object on which you want to define a constraint. Alternatively, you can right-click the data object in the Project Explorer and select **Open Editor**.

2. Navigate to the **Constraints** tab.
3. Depending on the type of constraint you want to create, refer to one of the following sections:
   - Creating Primary Key Constraints on page 12-2
   - Creating Foreign Key Constraints on page 12-3
   - Creating Unique Key Constraints on page 12-3
   - Creating Check Constraints on page 12-4

Creating Primary Key Constraints

To define a primary key constraint:

1. On the Constraints tab, click the **Add Constraint** button.

   A blank row is displayed with the cursor positioned in the Name column.

2. Enter the name of the constraint in the **Name** column.
3. In the **Type** column, select **Primary Key**.

   Press the tab key to navigate out of the Type column or use the mouse and click the empty space in the Constraints tab.

4. Click the **Add Local Column** button.

   A new row is added below the current row that contains the constraint name and constraint type. This new row displays a drop-down list in the Local Columns column.

5. In the **Local Columns** drop-down list of the new row, select the name of the column that represents the primary key.

   To create a composite primary key, repeat steps 4 and 5 for each column that you want to add to the primary key.

Creating Foreign Key Constraints

To define a foreign key constraint:

1. On the Constraints tab, click the **Add Constraint** button.

   A blank row is displayed with the cursor positioned in the Name field.

2. Enter the name of the constraint in the **Name** column.

3. In the **Type** column, select **Foreign Key**.

   Press the tab key to navigate out of the Type column or use the mouse and click the empty space in the Constraints tab.

4. In the References column, click the Ellipsis button.

   The Key Selector dialog is displayed. Use this dialog to select the table and the column that the current column references.

5. In the Key Selector dialog, select the primary key constraint that the foreign key references.

   For example, the DEPARTMENTS table has a primary key called DEPT_PK defined on the department_id column. To specify that the column department_id of the EMPLOYEES table is a foreign key that references the primary key DEPT_FK, select DEPT_FK under the node that represents the DEPARTMETNS table in the Key Selector dialog.

6. Click **OK**.

   You now return to the Constraints tab of the Data Object Editor and the foreign key constraint is added.

Creating Unique Key Constraints

To define a unique key constraint:

1. On the Constraints tab, click the **Add Constraint** button.

   A blank row is displayed with the cursor positioned in the Name field.

2. Enter the name of the constraint in the **Name** column and press the Enter key.

   You can also press the Tab key or click any other location in the editor.

3. In the **Type** column, select **Unique Key**.

   Press the tab key to navigate out of the Type column or use the mouse and click the empty space in the Constraints tab.

4. Click the **Add Local Column** button.

   A new row is added below the current row that contains the constraint name and constraint type. This new row displays a drop-down list in the Local Columns column.

5. In the **Local Columns** drop-down list of the new row, select the name of the column on which a unique key should be created.

Creating Check Constraints

To define a check constraint:

1. On the Constraints tab, click the **Add Constraint** button.

   A blank row is displayed with the cursor positioned in the Name field.

2. Enter the name of the constraint in the **Name** column and press the Enter key.

   You can also press the Tab key or click any other location in the editor.

3. In the **Type** column, select **Check Constraint**.

   Press the tab key to navigate out of the Type column or use the mouse and click the empty space in the Constraints tab.

4. In the Condition column, enter the condition to be applied for the check constraint. For example, salary > 2000. If you leave this field blank, an error is generated during validation and you cannot generate valid code for this constraint.

   The column name referenced in the check condition must exactly match the physical name defined for the table in its property sheet. Warehouse Builder does not check the syntax of the condition during validation. This may result in errors during deployment. If this happens, check the Repository Browser for details.

Editing Constraints

You edit constraints using the Constraints tab of the Data Object Editor. You can modify the following for a constraint:

- Rename a constraint
- Change the type
- Modify the check condition
- Modify the referenced column for a foreign key constraint.
- Modify the primary key column for a primary key.

Using Partitions

Partitions enable you to efficiently manage very large tables and indexes by dividing them into smaller, more manageable parts. Partitions improve query and load performance because operations work on subsets of data. Use partitions to enhance data access and improve overall application performance- especially for applications that access tables and indexes with millions of rows and many gigabytes of data.

You can create the following types of partitions:

- **Range Partitioning** on page 12-5
- **Hash Partitioning** on page 12-6
- **Hash By Quantity Partitioning** on page 12-7
- **List Partitioning** on page 12-7
- **Composite Partitioning** on page 12-9
- **Index Partitioning** on page 12-11

Range Partitioning

Range partitioning is the most common type of partitioning and is often used to partition data based on date ranges. For example, you can partition sales data into monthly partitions.

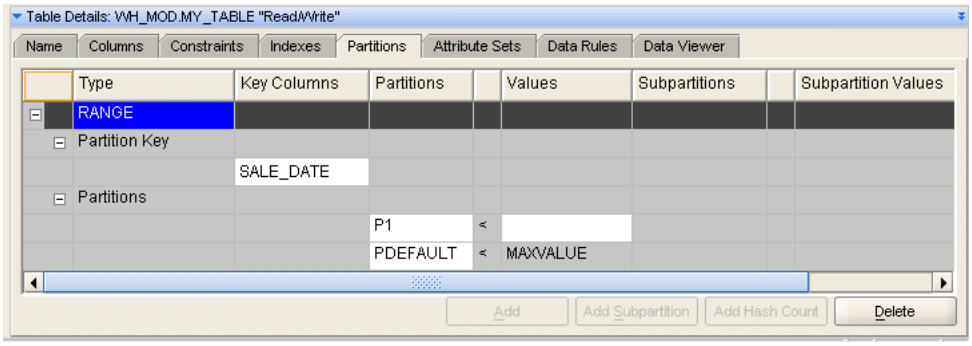
To use range partitioning in Warehouse Builder, go to the Partitions tab in the Data Object Editor to specify a partition key and assign a name and value range for each partition you want to create. Figure 12–2 on page 12-6 shows an example of a table partitioned into four range partitions based on the following instructions.

To partition data by ranges, complete the following steps:

1. On the Partitions tab in the Data Object Editor, click the first cell under **Type** and select **Range.**

   If necessary, click the plus sign to the left of Type to expand the template for the range partition shown in Figure 12–1.

Figure 12–1 Partition Tab with Range Partition Selected



2. Select a partition key.

   Warehouse Builder lists all the columns for the object you selected. You can select a column of any data type; however, DATE is the most common partition key for range partitioning.

   You can base the partition key on multiple key columns. To add another key column, select the partition key node and click Add.

3. Define the partitions.

   To assist you in defining the partitions, the template offers two partitions that you can edit but not delete. P1 represents the first partition and PDEFAULT represents the last partition. If you want to partition data based on month, you could rename P1 to *Jan* and PDEFAULT to *Dec.*

   The last partition is set to the keyword MAXVALUE, which represents a virtual infinite value that sorts higher than any other value for the data type, including the null value.

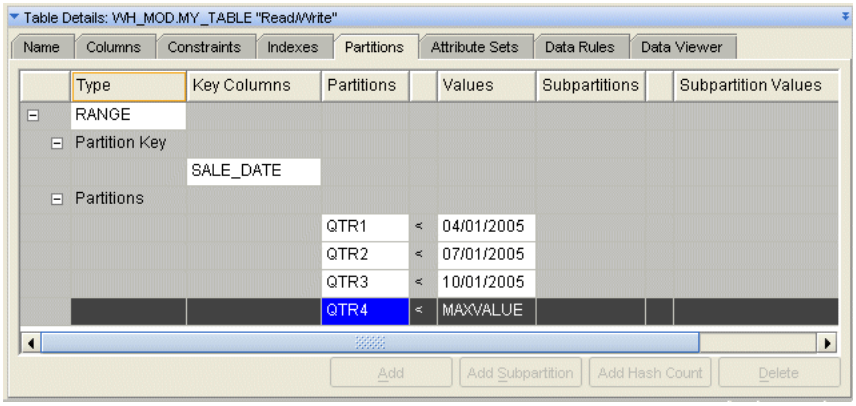
   To add more partitions between the first and last partitions, click the Partitions node and select **Add.**

   In **Values,** specify the greatest value for the first range and all the additional ranges you create. These values are the less than values. For example, if the first partition is for the first month of the year, then for that partition to contain values less than February 1st, type that date in the format DD/MM/YYYY.

Example of Range Partitioning

Figure 12–2 shows how to define a partition for each quarter of a calendar year. You could also partition data for each month or week. When you design mappings using such a table, consider enabling Partition Exchange Loading (PEL). PEL is a data definition language (DDL) operation that swaps existing partitions on the target table with new partitions. Since it is not a data manipulation language (DML) operation, the exchange of partitions occurs instantaneously.

Figure 12–2 Example Table with Range Partitioning



Hash Partitioning

Hash partitioning assigns data to partitions based on a hashing algorithm that Oracle applies to a partitioning key you identify. The hashing algorithm evenly distributes rows among partitions, giving partitions approximately the same size. Hash partitioning is a good and easy-to-use alternative to range partitioning when data is not historical and there is no obvious column or column list where logical range partition pruning can be advantageous.

To partition data based on the hash algorithm, complete the following steps:

1. On the Partitions tab in the Data Object Editor, click the cell below **Type** and select **Hash.**

   If necessary, click the plus sign to the left of Type to expand the template for defining hash partitions.

2. Select a partition key.

   Warehouse Builder lists all the columns for the object you selected. You can select a column of any data type.

3. Define the partitions.

   Warehouse Builder provides two partitions that you can rename. Click the Partitions node and select **Add** to add as many partitions as necessary.

   Oracle Database uses a linear hashing algorithm and, to prevent data from clustering within specific partitions, you should define the number of partitions by a power of two (for example, 2, 4, 8).

Hash By Quantity Partitioning

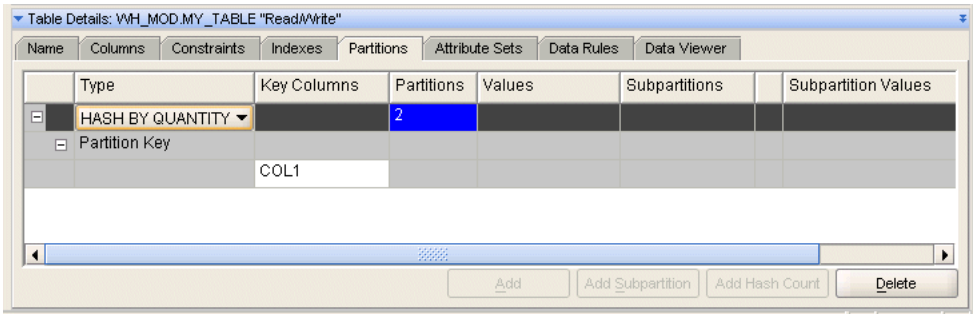
Use hash by quantity to quickly define hash partitioning. You define a partition key and the number of partitions and Warehouse Builder then creates and names the partitions. You can then configure the partitions to share the same tablespace list.

To partition data based on the hash algorithm, complete the following steps:

1. On the Partitions tab in the Data Object Editor, click the cell below **Type** and select **Hash by Quantity.**

   If necessary, click the plus sign to the left of Type to expand the template for defining hash by quantity partitions as shown in Figure 12–3.

Figure 12–3 Hash by Quantity Partitioning



2. Define the number of partitions. The default value is two partitions.

   Oracle Database uses a linear hashing algorithm and, to prevent data from clustering within specific partitions, you should define the number of partitions by a power of two (for example, 2, 4, 8).

3. Select a partition key.

   Warehouse Builder lists all the columns for the object you selected. You can select a column of any data type.

4. In the configuration window, define the Partition Tablespace List and Overflow Tablespace List.

List Partitioning

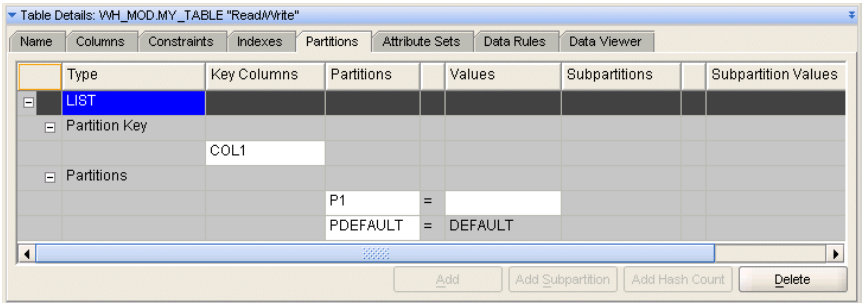
List partitioning enables you to explicitly assign rows to partitions. You do this by specifying a list of discrete values for each partition. The advantage of list partitioning is that you can group and organize unordered and unrelated sets of data in a natural way. Figure 12–5 on page 12-8 shows an example of a table partitioned into list partitions based the following instructions.

To partition data based on a list of values, complete the following steps:

1. On the Partitions tab in the Data Object Editor, click the cell below **Type** and select the partitioning method **List.**

   If necessary, click the plus sign to the left of Type to expand the template for defining list partitions as shown in Figure 12–4.

Figure 12–4 Partition Tab with List Partition Selected



2. Select a partition key.

   Warehouse Builder lists all the columns for the object you selected. You can select a column of any data type.

3. Define the partitions.

   PDEFAULT is set to the keyword DEFAULT and includes all rows not assigned to any other partition. A partition that captures all unassigned rows is essential for maintaining the integrity of the data.

   To assist you in defining the partitions, the template offers two partitions that you can edit but not delete. P1 represents the first partition and PDEFAULT represents the last partition.

   To add more partitions between the first and last partitions, click the Partitions node and select **Add.**

   In **Values,** type a comma separated list of values for each partition that correspond to data in the partition key you previously selected. For example, if the partition key is COUNTRY_ID, you could create partitions for Asia, Eastern Europe, Western Europe, and so on. Then, for the values for each partition, list the corresponding COUNTRY_IDs for each country in the region as shown in Figure 12–5.

Example of List Partitioning

Figure 12–5 shows a table with data list partitioned into different regions based on the COUNTRY_ID column. Each partition has a single comma separated list.

Figure 12–5 List Partitioning Based on a Single Key Column

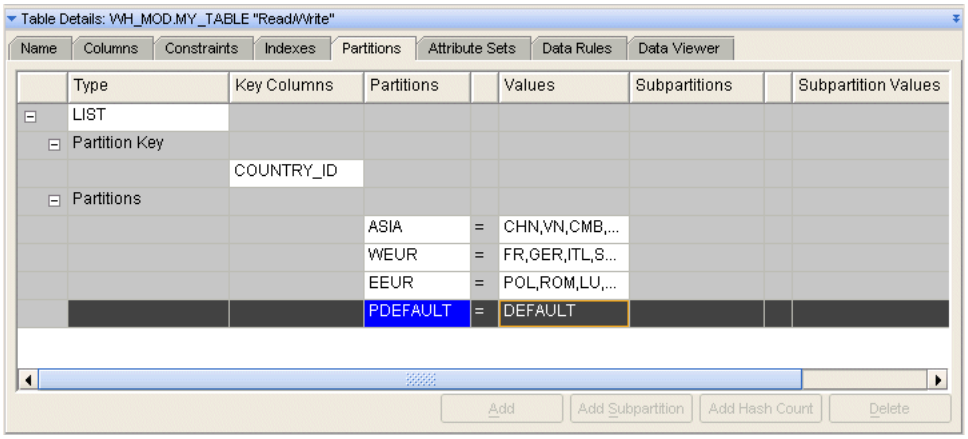
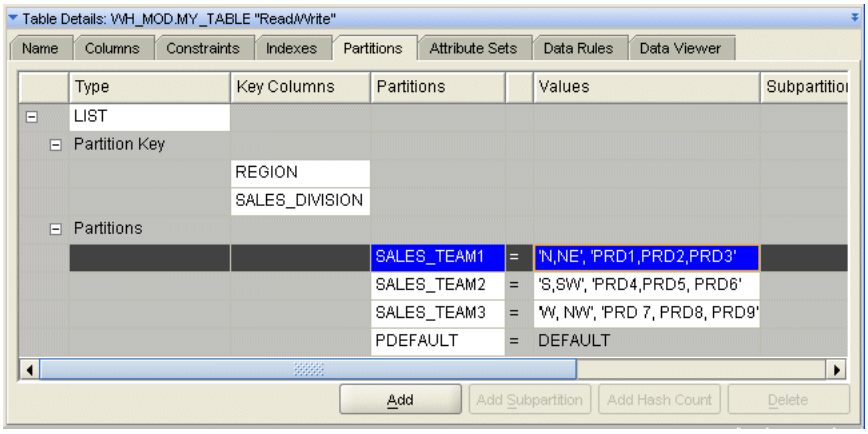


Figure 12–6 *shows a table with data partitioned based on key columns REGION and SALES_DIVISION. Each partition includes two comma separated lists enclosed by single quotes. In this example, N, NE, S, SW, W, and NW correspond to REGION while PRD1, PRD2, PRD3, and so on, corresponds to SALES_DIVISION.*

Figure 12–6 List Partitioning Based on Multiple Key Columns



Composite Partitioning

Composite partitioning methods include range-hash, range-hash by quantity, and range-list partitioning. The Oracle Database first performs the range partitioning and then further divides the data using the second partitioning you select.

The steps for defining composite partition methods are similar to defining simple partition methods such as range, hash, and list but include additional options.

To range partition data and then subpartition based on list, hash, or hash by quantity:

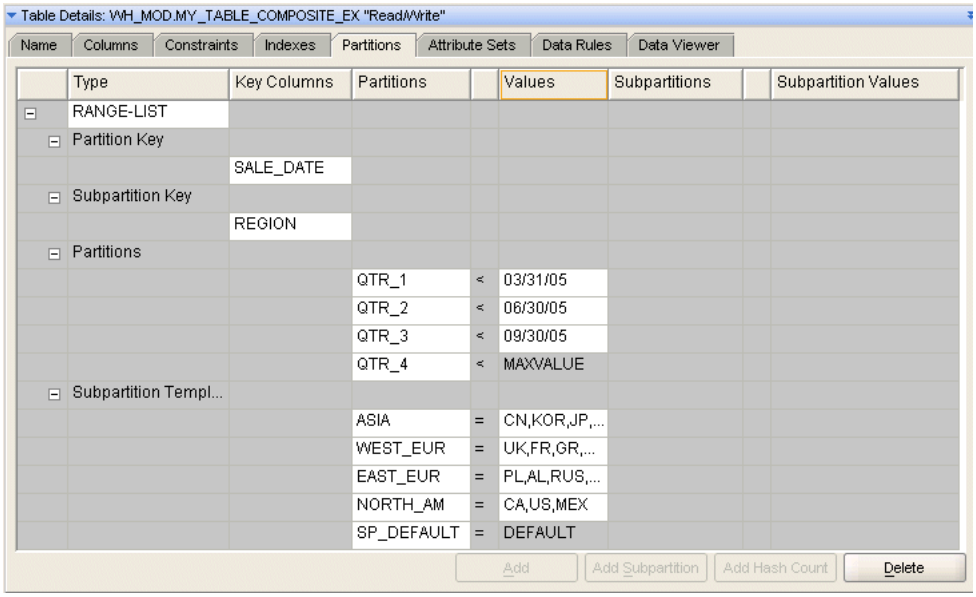
1. On the Partitions tab in the Data Object Editor, click the cell below **Type** and select one of the composite partitioning methods.

   If necessary, click the plus sign to the left of Type to expand the template.

2. Select a partition key and define partitions as described in "Range Partitioning" on page 12-5.

   In Figure 12–7, the partition key is SALE_DATE and its associated range partitions are QTR_1, QTR_2, QTR_3, and QTR_4.

Figure 12–7 Range-List Partitioning with List Defined under Subpartition Template



3. Select a column for the subpartition key.
4. Under the subpartition template node, define the values for the second partitioning method as described in "About the Subpartition Template" on page 12-10.
5. Define custom subpartitions. (optional)

   For range-list partitions, you can specify custom subpartitions that override the defaults you defined under the subpartition node. For details, see "Creating Custom Subpartitions" on page 12-10.
6. Configure the Partition Tablespace List and Overflow Tablespace List in the configuration window.

About the Subpartition Template

Use the subpartition template to specify the second partitioning method in composite partitioning. The steps you take depend on the type of composite partition you select.

For range-hash by quantity, type in a number of subpartitions only.

For range-hash, the subpartition template enables you to type names for the subpartitions only.

For range-list, name the lists and type in the comma separated values. Be sure to preserve the last subpartition as set to DEFAULT.

Figure 12–7 shows a list subpartition based on the REGION key column and subpartitions for groups of countries. Warehouse Builder divides each partition (such as QTR_1 and QTR_2) into subpartitions (such as ASIA, WEST_EUR,).

Creating Custom Subpartitions

Using the subpartition template is the most convenient and likely the most common way to define subpartitions. Entries you specify under the subpartition template apply uniformly to all the partitions under the partition node. However, in some cases, you may want to override the subpartition template.

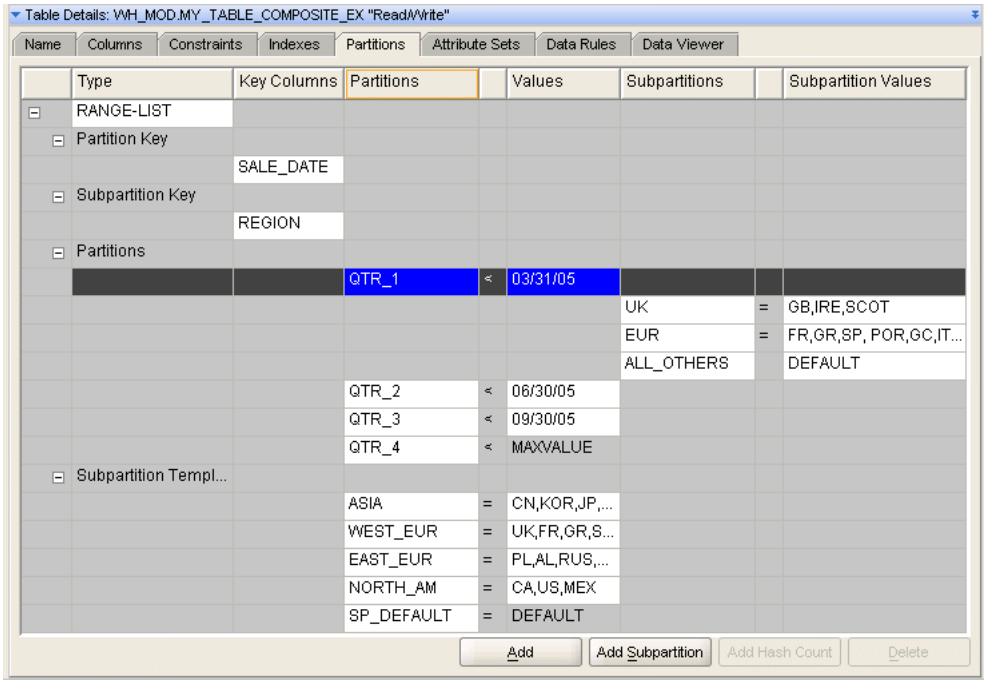
For range-hash by quantity, select a partition and then click **Add Hash Count.** Warehouse Builder expands the partition node to enable you to specify the number of hash subpartitions that uniquely apply to that partition.

For range-hash, select a partition and then click **Add Subpartition.** Warehouse Builder expands the partition node and you can name subpartitions for that partition only.

For range-list, select a partition and then click **Add Subpartition.** Warehouse Builder expands the partition node to enable you to specify list subpartitions for that partition only. Be sure to preserve the last subpartition as set to DEFAULT.

Figure 12–8 shows that partition QTR_1 is subpartitioned into lists for UK, EUR, and ALL_OTHERS while the other quarters are partitioned according to the subpartition template.

Figure 12–8 Subpartitions Overriding Subpartition Template



Index Partitioning

For all types of indexes except bitmap indexes, you can determine whether or it inherits the partitioning method of the underlying table. An index that inherits its partitioning method is known as a *local index* while an index with its own partitioning method is known as a *global index.*

Local Index

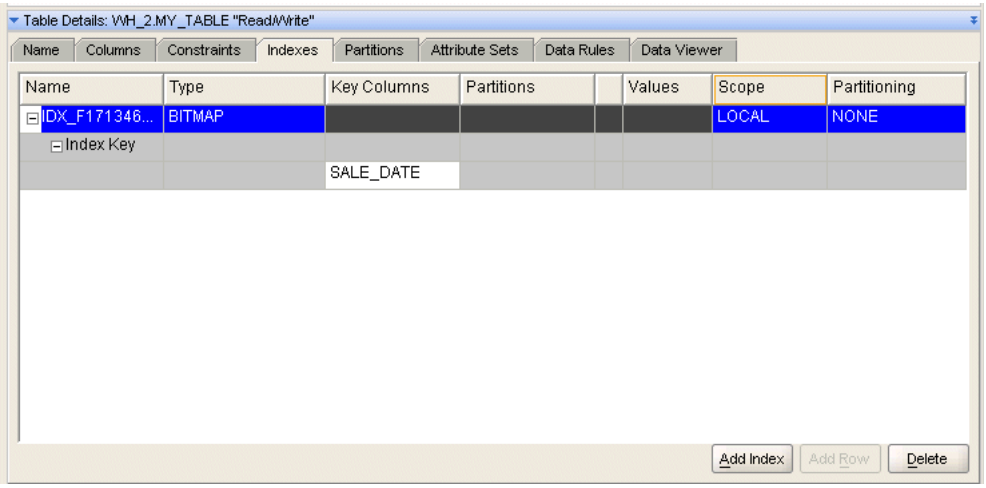
Local indexes are partitioned on the same columns and have the same definitions for partitions and subpartitions specified on the underlying table. Furthermore, local indexes share the same tablespaces as the table.

For example, if you used range-list partitioning to partition a table of sales data by quarter and then by region, a local index is also partitioned by quarter and then by region.

Bitmap indexes can only be defined as local indexes to facilitate the best performance for querying large amounts of data.

To define an index as local in Warehouse Builder, set the scope to LOCAL and partitioning to NONE as shown in Figure 12–9.

Figure 12–9 Bitmap Index with Scope Set to Local



Global Index

A global index is one in which you can partition the index independently of the partition strategy applied to the underlying table. You can choose between range or hash partitioning. The global index option is available for all indexes except bitmap indexes.

In previous releases, Oracle recommended that you not use global indexes for data warehouse applications because deleting partitions on the table during partition maintenance would invalidate the entire index and result in having to rebuild the index. Beginning the Oracle 10*g*, this is no longer a limitation as global indexes are no longer negatively affected by partitioning maintenance.

Nonetheless, local indexes are likely to be the preferred choice for data warehousing applications due to ease in managing partitions and the ability to parallelize query operations.

A global index is useful when that you want to specify an index partition key other than any of the table partition keys. In that case, ensure that there are no duplicate rows in the index key column and select unique for the index type.

Index Performance Considerations

As you decide which type of index to use, consider that indexes rank in performance in the following order:

1. Unique and local index
2. Unique and global index
3. All other non unique indexes (normal, bitmap, function based) and local index

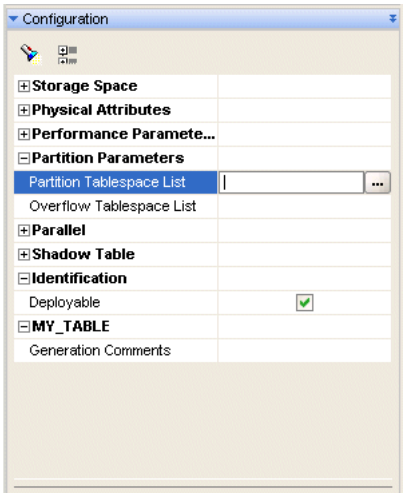
Configuring Partitions

For some but not all partitioning methods, you must configure partition tablespaces.

You can access the parameters for partitions either from the Project Explorer or the Data Object Editor. In the Project Explorer, right-click the table, select **Configure**, and scroll down to view Partition Parameters. Or, in the Data Object Editor, select the

configuration window and scroll down to view Partition Parameters as shown in Figure 12–10.

Figure 12–10 Partition Parameters in the Data Object Editor Configuration Window



Partition Tablespace List

Type a comma separated list of tablespaces when you partition by any of the following methods: Hash by Quantity, Range-List, Range-Hash, or Range-Hash by Quantity.

If you neglect to specify partition tablespaces, Warehouse Builder uses the default tablespaces associated with the table and the performance advantage for defining partitions is not realized.

Overflow Tablespace List

Type a comma separated list of tablespaces when you partition by the method Hash by Quantity. If you provide a list of tablespaces less than the number of partitions, the Oracle Database cycles through those tablespaces.

If you neglect to specify overflow tablespaces, Warehouse Builder uses the default tablespaces associated with the table and the performance advantage for defining partitions is not realized when the limits for the partition tablespaces are exceeded.

Using Tables

In Warehouse Builder, tables are metadata representations of relational storage objects. They can be tables from a database system such as Oracle tables or even tables from an SAP system. The following sections provide information about creating and using tables in Warehouse Builder:

- Creating Table Definitions on page 12-13
- Editing Table Definitions on page 12-16

Creating Table Definitions

The table you create in Warehouse Builder captures the metadata used to model your target schema. This table definition specifies the table constraints, indexes, partitions, attribute sets, and metadata about the columns and data types used in the table. This information is stored in the Warehouse Builder repository. You can later use these definitions to generate `.ddl` scripts in Warehouse Builder that can be deployed to

create physical tables in your target database. These tables can then be loaded with data from chosen source tables.

You use the Data Object Editor to create a table. Use the following the steps:

1. From the Project Explorer, expand the Databases node and then the Oracle node.
2. Expand the module where you want to create the table.
3. Right-click **Tables** and select **New**.

   Warehouse Builder displays the Data Object Editor. Use the following tabs in the Table Details panel of the Data Object Editor to define the table.

   - Name Tab on page 12-14
   - Columns Tab on page 12-14
   - Constraints Tab on page 12-15
   - Indexes Tab on page 12-15
   - Partitions Tab on page 12-15
   - Attribute Sets Tab on page 12-15
   - Data Rules Tab on page 12-15

   The Data Viewer tab enables you to view the data stored in the repository table. For more information on the Data Viewer tab, see "Data Viewer" on page 4-17.

   After you finish defining the table using these tabs, Warehouse Builder creates and stores the table definition in the repository. It also inserts the new table name in the Project Explorer.

   Note: You can also create a table from the Mapping Editor.

Name Tab

Follow the rules in "Naming Conventions for Data Objects" on page 4-4 to specify a name and an optional description.

Columns Tab

Use the Columns tab to define the columns in the table. This tab displays a table that you use to define columns. Each row in the table corresponds to the definition of one table column. Warehouse Builder generates the column position in the order in which you type in the columns. To re-order columns, see "Reordering Columns in a Table" on page 12-16.

Enter the following details for each column:

- **Name:** Type the name of the column. The column name should be unique within the table. Reserved words are not allowed.
- **Data Type:** Select the data type of the column from the **Data Type** drop-down list. Warehouse Builder assigns a default data type for each column based on the column name. For example, if you create a column named start_date, the data type assigned is DATE. You can change the default assignment if it does not suit your data requirement.

  For a list of supported Oracle Database data types, see "Supported Data Types" on page 4-4.

- **Length:** Specify the length of the attribute. Length is applicable for character data types only.
- **Precision:** Specify the total number of digits allowed for the column. Precision is applicable for numeric data types only.
- **Scale:** Specify the total number of digits to the right of the decimal point. Scale is applicable for numeric data types only.
- **Seconds Precision:** Used for TIMESTAMP data types only. Specify the number of digits in the fractional part of the datetime field.
- **Not Null:** Select this field to specify that the column should not contain null values. By default, all columns in a table allow nulls.
- **Default Value:** Specify the default value for this column. If no value is entered for this column while loading data into the table, the default value is used. If you specify a value for this column while loading data, the default value is overridden and the specified value is stored in the column.
- **Description:** Type an optional description for the column.

Constraints Tab

Use the Constraints tab to create constraints on the table columns. You can create primary keys, foreign keys, unique keys, and check constraints. For more information on creating constraints, see "Creating Constraints" on page 12-2.

Indexes Tab

Use the Indexes tab to create indexes on the table. You can create the following types of indexes: unique, normal, bitmap, function-based, composite, and reverse.

For more information on creating these indexes, see "Creating Indexes" on page 4-23.

Partitions Tab

Use the Partitions tab to create partitions for the table. You can create the following types of partitions: range, hash, hash by quantity, list, composite, and index.

For more information on these partitions and how to create each type of partition, see "Using Partitions" on page 12-4.

Attribute Sets Tab

Use the Attribute Sets tab to create attribute sets for the table. You can create user-defined and bridge attribute sets. For more information about creating attribute sets for a table, see "About Attribute Sets" on page 4-21.

Data Rules Tab

Use the Data Rules tab to apply data rules to a table. Data rules enable you to determine legal data within a table and legal relationships between tables. When you apply a data rule to a table, Warehouse Builder ensures that the data in the table is according to the specified data rule. For more information about data rules, see "About Data Rules" on page 10-16.

Before you apply a data rule to a table, the data rule should have been defined in the repository. For information about creating data rules, see "Creating Data Rules" on page 20-37.

To apply a data rule to a table, click the **Apply Rule** button on the Data Rules tab. The Apply Data Rule Wizard is displayed. Use this wizard to select the data rule and the column to which the data rule should be applied. For more information about using the Apply Data Rule Wizard, see "Applying Data Rules" on page 20-38.

The Applied Rules section of the Data Rules tab displays the data rules that are bound to the table. For a data rule to be applied to a table, ensure that the check box to the left of the data rule name is selected. Deselecting this option will result in the data rule not being applied to the table.

The **Binding** column of the Bindings section displays the table column to which the data rule is bound.

Editing Table Definitions

You use the Data Object Editor to edit table definitions. To launch the Data Object Editor, right-click the name of the table in the Project Explorer and select **Open Editor**. Alternatively, you can double-click the name of the table in the Project Explorer.

The following sections describe the table definitions that you can edit.

Renaming a Table

Navigate to the Name tab of the Data Object Editor. Click the **Name** field and enter the new name for the table. You can also modify the description stored in the **Description** field.

Adding, Modifying, and Removing Table Columns

Adding a column: Navigate to the Columns tab. Click the **Name** field in an empty row and enter the details that define a column. For more information, see "Columns Tab" on page 12-14.

Modifying a column: Use the Columns tab of the Data Object Editor to modify column definitions. You can modify any of the attributes of the column definition. For more information, see "Columns Tab" on page 12-14.

Removing a column: Navigate to the Columns tab. Right-click the grey cell to the left of the column name you want to remove and select **Delete**.

Adding, Modifying, and Deleting Table Constraints

Navigate to the Constraints tab of the Data Object Editor.

For details on adding and editing constraints, see "Creating Constraints" on page 12-2 and "Editing Constraints" on page 12-4 respectively.

To delete a constraint, select the row that represents the constraint by clicking the grey cell to the left of the column name. Click **Delete** at the bottom of the tab.

Adding, Editing, and Deleting Attribute Sets

For details about adding attribute sets, see "Creating Attribute Sets" on page 12-23. See "Editing Attribute Sets" on page 12-24 for instructions on how to edit an attribute set.

To delete an attribute set, navigate to the Attribute Sets tab. Right-click the cell to the left of the attribute set that you want to remove and select **Delete**.

Reordering Columns in a Table

By default, columns in a table are displayed in the order they are created. This order is also propagated to the DDL script generated by Warehouse Builder to create the table.

If this default ordering does not suit your application needs, or if you want to further optimize query performance, you can reorder the columns.

To change the position of a column:

1. If the Data Object Editor is not already open for the table, open the editor.

   You can do this by double-clicking the name of the table in the Project Explorer. Alternately, you can right-click the name of the table in the Project Explorer and select **Open Editor**.

2. On the Columns tab, select the gray square located to the left of the column name.

   The entire row is highlighted.

3. Use the buttons on the left of the Columns tab to move the column to the required position.

   The position of the column is now updated.

4. Close the Data Object Editor**.**

For the change in the column position to be reflected in the table stored in the repository you must deploy the changed table definition.

Using Views

In Warehouse Builder, you can define views and materialized views. This section describes views. For information on materialized views, see "Using Materialized Views" on page 12-20.

About Views

Views are used to simplify the presentation of data or restrict access to data. Often the data that users are interested in is stored across multiple tables with many columns. When you create a view, you create a query stored to retrieve only the relevant data or only data that the user has permission to access.

In Warehouse Builder, a view can be defined to model a query on your target data. This query information is stored in the Warehouse Builder repository. You can later use these definitions to generate `.ddl` scripts in Warehouse Builder that can be deployed to create views in your target system.

For information about using views, refer to the following:

- Creating View Definitions on page 12-17
- Editing View Definitions on page 12-19

Creating View Definitions

A view definition specifies the query used to create the view, constraints, attribute sets, data rules, and metadata about the columns and data types used in the view. This information is stored in the Warehouse Builder repository. You can generate the view definition to create `.ddl` scripts. These scripts can be deployed to create the physical views in your database.

The Data Object Editor enables you to create a view definition. Use the following steps to create a view definition:

1. From the Warehouse Builder Project Explorer expand the Databases node and then the Oracle node.

2. Expand the target module where you want to create the view.
3. Right-click **Views** and select **New.**

   Warehouse Builder displays the Data Object Editor for the view.

   Note: You can also define a View from the Mapping Editor and model your own query.

4. Define the view by specifying information on the following Data Object Editor tabs:
   - Name Tab on page 12-18
   - Columns Tab on page 12-18
   - Query Tab on page 12-18
   - Constraints Tab on page 12-18
   - Attribute Sets Tab on page 12-19
   - Data Rules Tab on page 12-19
   - Data Viewer Tab on page 12-19
5. Close the Data Object Editor.

   Warehouse Builder creates a definition for the view, stores this definition in the repository, and inserts its name in the Project Explorer.

Name Tab

Follow the rules in "Naming Conventions for Data Objects" on page 4-4 to specify a name and an optional description.

Columns Tab

Use the Columns tab to define the columns in the view. This tab displays a table that you use to define the view columns. Each row in this table corresponds to one view column definition. For each view column, enter the following details: Name, Data Type, Length, Precision, Scale, Seconds Precision, Not Null, Default Value, and Description. For an explanation of these fields see "Columns Tab" on page 12-14.

Query Tab

Use the Query tab to define the query used to create the view. A view can contain data from tables that belongs to a different module than the one to which the view belongs. You can also combine data from more then one table using joins.

Ensure that the query statement you type is valid. Warehouse Builder does not validate the text in the Query tab and will attempt to deploy a view even if the syntax is invalid.

Constraints Tab

Use this page to define logical constraints for a view. Although Warehouse Builder does not use these constraints when enumerating DDL for the view, these constraints can be useful when the view serves as a data source in a mapping. The Mapping Editor can use the logical foreign key constraints to include the referenced dimensions as secondary sources in the mapping.

Note: You cannot create check constraints for views.

For more information about creating constraints, see "About Constraints" on page 4-22.

Attribute Sets Tab

Use the Attribute Sets tab to define attribute sets for the view. For more information on attribute sets and how to create them, see "About Attribute Sets" on page 4-21.

Data Rules Tab

Use the Data Rules tab to specify the data rules that are applied to the view. For more information about the Data rules tab, see "Data Rules Tab" on page 12-15.

Data Viewer Tab

The Data Viewer tab enables you to view the data stored in the underlying database table on which the view is based. For more on this tab, see "Data Viewer" on page 4-17.

Editing View Definitions

Use the Data Object Editor to edit view definitions. To open the Data Object Editor, right-click the view in the Project Explorer and select **Open Explorer**. The following sections describe the view definitions that you can edit.

Renaming a View

Navigate to the Name tab of the Data Object Editor. Click the **Name** field and enter the new name for the view. You can also modify the description stored in the **Description** field. Type the new name over the highlighted object name.

Alternately, you can rename a view by right-clicking the view name in the Project Explorer and selecting **Rename**.

Adding, Editing, and Removing View Columns

Adding a column: Navigate to the Columns tab. Click the **Name** field in an empty row and enter the details that define a column. For more information on these details, see "Columns Tab" on page 12-18.

Removing a column: Navigate to the Columns tab. Right-click the grey cell to the left of the column name you want to remove and select **Delete**.

Adding, Editing, and Deleting View Constraints

Navigate to the Constraints tab of the Data Object Editor. For details on adding and editing constraints, see "Creating Constraints" on page 12-2 and "Editing Constraints" on page 12-4 respectively.

To delete a constraint, on the Constraints tab, select the row that represents the constraint. Click **Delete** at the bottom of the tab.

Adding, Editing, and Removing Attribute Sets

For details about adding attribute sets, see "Creating Attribute Sets" on page 12-23. See "Editing Attribute Sets" on page 12-24 for instructions on how to edit an attribute set.

To delete an attribute set, navigate to the Attribute Sets tab. Right-click the cell to the left of the attribute set that you want to remove and select **Delete**.

Using Materialized Views

In Warehouse Builder, you can define views and materialized views. This section discusses materialized views. For information on conventional views, see "Using Views" on page 12-17.

The following sections provide information about using materialized views:

About Materialized Views

In Warehouse Builder, you can create materialized views to improve query performance. When you create a materialized view, you create a set of query commands that aggregate or join data from multiple tables. Materialized views provide precalculated data that can be reused or replicated to remote data marts. For example, data about company sales is widely sought throughout an organization.

When you create a materialized view in Warehouse Builder, you can configure it to take advantage of the query rewrite and fast refresh features available in Oracle Database. For information on query rewrite and fast refresh, "Fast Refresh for Materialized Views" on page 12-37.

Creating Materialized View Definitions

A materialized view definition specifies the query used to create the materialized view, constraints, indexes, partitions, attribute sets, data rules, and metadata about the columns and data types used in the materialized view. You can generate the view definition to obtain `.ddl` scripts that are used to deploy the materialized view.

You use the Data Object Editor enables to create materialized views. Follow these steps to create a materialized view:

1. From the Warehouse Builder Project Explorer expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the materialized view.
3. Right-click **Materialized View** and select **New.**

   Warehouse Builder displays the Data Object Editor for this materialized view.

   Note: You can also define a Materialized View from the Mapping Editor.

4. Specify information on the following tabs of the Data Object Editor to create the materialized view definition:
   - Name Tab on page 12-21
   - Columns Tab on page 12-21
   - Query Tab on page 12-21
   - Constraints Tab on page 12-21

- Indexes Tab on page 12-21
- Partitions Tab on page 12-21
- Attribute Sets Tab on page 12-22
- Data Rules Tab on page 12-22

5. Close the Data Object Editor.

   The wizard creates a definition for the materialized view, stores this definition in the database module, and inserts its name in the warehouse module Project Explorer.

Name Tab

Follow the rules in "Naming Conventions for Data Objects" on page 4-4 to specify a name and an optional description.

Columns Tab

Use the Columns tab to define the materialized view columns. This tab displays a table that enables you to define the columns. A row in the table corresponds to one column in the materialized view. For each column specify the following details: Name, Data Type, Length, Precision, Scale, Seconds Precision, Not Null, Default Value, and Description. For more details, see "Columns Tab" on page 12-14.

Query Tab

Use the Query tab to define the query used to create the materialized view. Ensure that you type a valid query in the **Select Statement** field. For column names, use the same names that you specified on the Columns page in the previous step. If you change a column name on the columns page, you must manually change the name in the Query tab. Warehouse Builder does not validate the text in the Query tab and will attempt to deploy a materialized view even if the syntax is invalid.

Constraints Tab

Use the Constraints tab to define constraints for the materialized view. Defining constraints is optional. These constraints are for logical design purposes only and Warehouse Builder does not use these constraints when enumerating DDL for the view. For information on creating constraints, see "Creating Constraints" on page 12-2.

Note: You cannot create check constraints for views.

Indexes Tab

Use the Indexes tab to define indexes on the materialized view. Defining indexes is optional. You can create the following types of indexes: Unique, Normal, Bitmap, Function-based, Composite, Reverse.

For information on creating indexes, see "Creating Indexes" on page 4-23.

Partitions Tab

Use the Partitions tab to define partitions on the materialized view. Partitioning a materialized view is optional. You can perform Index Partitioning, Range Partitioning, Hash Partitioning, Hash By Quantity Partitioning, List Partitioning, or Composite Partitioning.

Attribute Sets Tab

Use the Attribute Sets tab to define attribute sets for the materialized view. Defining attribute sets is optional. The types of attribute sets you can create are user-defined and bridge. For information on how to define attribute sets, see "Creating Attribute Sets" on page 12-23.

Data Rules Tab

Use the Data Rules tab to specify data rules that should be applied to the materialized view data.

Editing Materialized View Definitions

Use the Data Object Editor to edit a materialized view definition. To open the Data Object Editor, right-click the materialized view and select **Open Editor**. The following sections describe the definitions that you can edit for a materialized view.

Renaming Materialized Views

Double click the **Name** field on the Name tab of the editor. This selects the name. Type the new name.

Alternately, in the Project Explorer, right-click the materialized view name and select **Rename.** Type the new name over the highlighted object name.

Adding, Editing, and Deleting Materialized View Columns

Adding a column: Navigate to the Columns tab. Click the Name field in an empty row and enter the details for the column. For more information on these details, see "Columns Tab" on page 12-18.

Removing a column: Navigate to the Columns tab. Right-click the grey cell to the left of the column name you want to remove and select **Delete**.

Adding, Editing, and Deleting Materialized View Constraints

Navigate to the Constraints tab of the Data Object Editor. For details on adding and editing constraints, see "Creating Constraints" on page 12-2 and "Editing Constraints" on page 12-4 respectively.

To delete a constraint, on the Constraints tab, select the row that represents the constraint. Click **Delete** at the bottom of the tab.

Adding, Editing, and Deleting Attribute Sets

For details about adding attribute sets, see "Creating Attribute Sets" on page 12-23. See "Editing Attribute Sets" on page 12-24 for instructions on how to edit an attribute set.

To delete an attribute set, navigate to the Attribute Sets tab. Right-click the cell to the left of the attribute set that you want to remove and select **Delete**.

Using Attribute Sets

An attribute set contains a chosen set of columns in the order you specify. Attribute sets are useful while defining a mapping or during data import and export. Warehouse Builder enables you to define attribute sets for tables, views, and materialized views. For the sake of brevity, in the following sections, the word *table* refers to all objects for which you can define attribute sets

For each table, Warehouse Builder generates a predefined attribute set containing all the columns in that table. In addition, Warehouse Builder generates predefined attribute sets for each defined constraint. Predefined attribute sets cannot be modified or deleted.

Creating Attribute Sets

You use the Attribute Sets tab of the Data Object Editor to create attribute sets. You can create the following types of attribute sets:

- **User-defined:** Optional attribute sets that can be created, modified, or deleted in the Table Properties window.
- **Bridge:** Optional attribute sets that can be can be exported to and viewed in another application such as Oracle Discoverer. You can create, modify, or delete bridge-type attribute sets in the Table Properties window. An object can only have one bridge-type attribute set.

To add an attribute set to a table:

1. From the Project Explorer, right-click the table name and select **Open Editor.**

   The Data Object Editor for the table is displayed.

2. Select the Attribute Sets tab.

   This tab contains two sections: *Attribute sets* and *Attributes of the selected attribute set*.

   The Attribute sets section displays the attribute sets defined for the table. It contains three columns that define each attribute set: Name, Type, and Description.

   The Attributes of the selected attribute set section lists all the attributes in the table. The attributes that are selected using the *Include* column are the ones that are part of the attribute set that is selected in the *Attribute sets of the entity* section.

3. In the **Attribute sets of the entity** section, click the **Name** field of an empty row and enter a name for the attribute set.

   In physical mode, you must type a name between 1 and 30 valid characters. Spaces are not allowed. In logical mode, you can type up to 200 valid characters. Spaces are allowed. The attribute set name must be unique within the object.

   Notice that all the table attributes are displayed in the *Attributes of the selected attribute set* section.

4. In the **Type** drop-down list, select the type of attribute set as USER_DEFINED or BRIDGE_TYPE.

5. Optionally, you can enter a description for the attribute set using the **Description** column.

6. In the **Attributes of the selected attribute set** section, click **Include** for each attribute you want to include in the attribute set. The order in which you select the columns determines their initial order in the attribute set.

   You can click **Select All** to select all the displayed columns in the attribute set. Click **Deselect All** to exclude all the columns from the attribute set. To remove a column from the attribute set, click the check box again to remove the check mark.

7. If you selected BRIDGE-TYPE, click **Advanced.**

   Warehouse Builder displays the Advanced Attribute Set Properties dialog.

8. For each attribute in the bridge-type attribute set, specify the following properties. These properties determine how the objects appear and display in Oracle Discoverer.

   Hidden: Click this check box to hide unused or obsolete columns when the table is viewed in another application. In the Discoverer Administration Edition, hidden columns are grayed out. In the Discoverer Plus Edition, hidden columns are not displayed.

   Aggregation: Select an aggregation for numerical attributes SUM, MIN, MAX, AVG, COUNT, or DETAIL for no aggregation. The default is SUM.

   Position: Select the default attribute position: DATA POINT, PAGE, SIDE, TOP, or TOP/SIDE. The default is DATA POINT.

   Item Class: Check for TRUE or uncheck for FALSE. The default is FALSE.

   Heading: Type the heading text.

   Format: Type the text for the format field.

9. Click **OK** to close the Advanced Attribute Set Properties dialog.

Editing Attribute Sets

Use the Attribute Sets tab of the Data Object Editor to edit attribute sets. You can perform the following actions when you edit an attribute set. Before you edit an attribute set, ensure that the Data Object Editor is open for the object that contains the attribute set. Also, navigate to the Attribute Sets tab of the Data Object Editor.

- Rename the attribute set

  Click the name of the attribute set in the **Name** column of the **Attribute sets of the entity** section and type the new name.

- Modify the type of attribute set

  Use the **Type** drop-down list in the **Attribute sets of the entity** section to modify the type of attribute set.

- Add or remove attributes from the attribute set

  Adding attributes to an attribute set: Select the attribute set to which you want to add attributes by clicking the grey cell to the left of the attribute set name in the *Attribute sets of the entity* section. In the *Attributes of the selected attribute set* section, click **Include** for each attribute that you want to include in the attribute set.

  Removing attributes from an attribute set: Select the attribute set from which you want to remove attributes by clicking the grey cell to the left of the attribute set. In the *Attributes of the selected attribute set* section, click **Include** for the attribute that you want to remove from the attribute set.

- Change the order in which the attributes appear in the attribute set

  Use the buttons to the left of the *Attributes of the selected attribute set* section to change the order of the attributes in the attribute set. Click the grey cell to the left of an attribute and use the buttons to move the attribute up or down in the order.

To delete an attribute set, right-click the grey cell to the left of the attribute set name and select **Delete**.

Using Sequences

A sequence is a database object that generates a serial list of unique numbers. You can use sequences to generate unique primary key values and to coordinate keys across multiple rows or tables. Sequence values are guaranteed to be unique. When you create a sequence in Warehouse Builder, you are creating sequence definitions that are saved in the repository. Sequence definitions can be used in mappings to generate unique numbers while transforming and moving data to your target system.

The following sections provide information about using sequences:

About Sequences

A sequence is referenced in SQL statements with the NEXTVAL and CURRVAL pseudo columns. Each new sequence number is incremented by a reference to the pseudo column NEXTVAL, while the current sequence number is referenced using the pseudo column CURRVAL. When you define a sequence, Warehouse Builder creates these attributes.

In Warehouse Builder, you can also import sequence definitions from existing source systems using the Import Object Wizard.

Creating a Sequence Definition

To create a new sequence:

1. From the Warehouse Builder Project Explorer, expand the warehouse module node.
2. Right-click Sequences and select **New** from the pop-up menu.

   Warehouse Builder displays the Create Sequence Wizard.
3. Use the Name tab to specify a name and an optional description for the table. In addition to the rules listed in "Naming Conventions for Data Objects" on page 4-4, the name must be unique across the module.
4. Click **OK.**

   Warehouse Builder stores the definition for the sequence and inserts its name in the Project Explorer.

Editing Sequence Definitions

You use the Edit Sequence dialog to edit a sequence definition. You can edit the name, description, and column notes of a sequence.

To edit sequence properties, right-click the name of the sequence from the Project Explorer and select **Open Editor** or double-click the name of the sequence. The Edit Sequence dialog is displayed. This dialog contains two tabs: Name Tab and Columns Tab.

Click these tabs to perform the following tasks:

- Rename a sequence
- Edit sequence columns

Name Tab

Follow the rules in "Naming Conventions for Data Objects" on page 4-4 to specify a name and an optional description.

You can also rename a sequence by right-clicking the sequence name in the Project Explorer and selecting **Rename**.

Columns Tab

The Columns tab displays the sequence columns CURRVAL and NEXTVAL. You can edit the descriptions of these columns.

Editing Sequence Column Descriptions

To edit the column descriptions of a sequence:

1. Right-click the name of a sequence and select **Open Editor.**

   The Sequence Editor dialog opens.

2. Select the Columns tab.
3. Scroll to the **Description** field and type or modify the description for the selected column.

Using User Defined Types

User-defined datatypes use Oracle built-in datatypes and other user-defined datatypes as the building blocks of object types that model the structure and behavior of data in applications. The built-in datatypes are mostly scalars and does not provide you the same flexibility that modelling an application specific data structure does.

Let us take a simple example of a customers table. The Customers address information is usually modeled as four or five separate fields , each with an appropriate scalar type. User defined types allow for a definition of 'address' as a composite type and also to define validation on that type.

A user defined data type extends the modeling capabilities of native data types. User defined data types specify both the underlying persistent data (attributes) and the related behaviors (methods).

With User defined types, you can create better models of complex entities in the real world by binding data attributes to semantic behavior.

Creating User Defined Types

User defined types are built from a combination of one or more simple data types. Integers, characters, and byte strings are examples of simple data types.

This section provides an overview of the following user data types

- Objects
- Varrays
- Nested Tables

About Object Types

Object types are abstractions of the real-world entities, such as purchase orders, that application programs deal with. It is a heterogeneous user defined type. It is made up of one or more user defined types or scalar types.

An object type is a schema object with the following components:

- **Name:** A name identifies the object type uniquely within that schema.
- **Attributes:** An attribute is used to create the structure and state of the real-world entity that is represented by an object. Attributes can be built-in types or other user-defined types.
- **Methods:** A method contains functions or procedures that are written in PL/SQL or Java and stored in the database, or written in a language such as C and stored externally. Methods are code-based representations of the operations that an application can perform on the real-world entity.

> **Note:** Methods are currently not supported.

For example, the address type definition can be defined as follows:

```
CREATE TYPE ADDRESS AS OBJECT ( street_name varchar2(240) ,
door_no varchar2(30) , po_box_no number , city varchar2(35) ,
state varchar2(30), country varchar2(30)).
```

Once the type has been defined it can be used across the schema for any table that requires the type definition 'address' as one of its fields.

Creating Object Types

To create an object type:

1. In Project Explorer, expand the Databases node and then the Oracle node.
2. Expand the module in which you want to create the object type.
3. Expand the User Defined Types node.
4. Right-click **Object Types**, and select **New**.

   The Data Object Editor is displayed. Use the following tabs on the ObjectType Properties panel of the Data Object Editor to define the object type:

   - Name Tab
   - Columns Tab

Name Tab

Follow the rules in "Naming Conventions for Data Objects" on page 4-4 to specify a name and an optional description.

Columns Tab

Use the Columns tab to define the columns in the object type. This tab displays a list of attributes that you can use to define columns. Each row in the attribute corresponds to the definition of one object column.

Specify the following details for each column:

- **Name:** Enter the name of the column. The column name must be unique within the object type. Reserved words are not allowed.
- **Data Type:** Select the data type of the column from the Data Type list. Warehouse Builder assigns a default data type for the column based on the column name. For example, if you create a column named `start_date`, the data type assigned is

DATE. You can change the default assignment if it does not suit your data requirement.

> **See also:** "Supported Data Types" on page 4-4 for a list of supported Oracle Database data types.

- **Length:** Specify the length of the column. Length is applicable for character data types only.
- **Precision:** Specify the total number of digits allowed for the column. Precision is applicable for numeric data types only.
- **Scale:** Specify the total number of digits to the right of the decimal point. Scale is applicable for numeric data types only.
- **Seconds Precision:** Specify the number of digits in the fractional part of the datetime field. Seconds precision is used for TIMESTAMP data types only.
- **Not Null:** Select this field to specify that the column should not contain NULL values. By default, all columns in a table allow nulls. This column is not applicable for Object types.
- **Default Value:** Specify the default value for this column. If no value is entered for this column while data is stored in the table, then the default value is used. If you specify a value for this column while loading data, then the default value is overridden and the specified value is stored in the column. This column is not applicable for Object types.
- **Description:** Type a description for the column. This is optional.

Editing Object Types

To edit an object type:

1. In Project Explorer, expand the Databases node and then the Oracle node.
2. Expand the module where you want to create the object type.
3. Expand the User Defined Types node.
4. Expand the **Object Types** node
5. Right-click the object type you want to edit and select Open Editor.

The Data Object Editor is displayed. Use the Name and Columns tabs as defined in the Creating Object Types section on page 12-27 to edit the definition of the object type.

About Varrays

A varray is an ordered collection of data elements. The position of each element in a varray is stored as an index number. You can use this number to access particular elements. When you define a varray, you specify the maximum number of elements it can contain. You can change this number later. Varrays are stored as opaque objects (such as RAW or BLOB).

If the customer has more than one address , for example three addresses, then you can create another type , a table type, that holds three addresses. The following example creates a table of address type:

```
TYPE address_store is VARRAY(3) of address;
```

Since Varrays is an ordered set of elements it can be considered that the first address in the list is the primary address, the remaining addresses are the secondary addresses.

Creating Varrays

To create a varray:

1. From the Project Explorer, expand the Databases node and then the Oracle node.
2. Expand the module where you want to create the varray.
3. Expand the User Defined Types node.
4. Right-click **Varrays** and select **New**.

   The Data Object Editor is displayed. Use the following tabs on the Varray Details panel of the Data Object Editor to define the object type:

   - Name Tab
   - Details Tab

Name Tab

Follow the rules in "Naming Conventions for Data Objects" on page 4-4 to specify a name and an optional description.

Details Tab

Use the Details tab to specify the value for the following fields:

- **Length:** Specify the length of the varray element. Length is applicable for character data types only.
- **Precision:** Specify the total number of digits allowed for the varray element. Precision is applicable for numeric data types only.
- **Scale:** Specify the total number of digits to the right of the decimal point. Scale is applicable for numeric data types only.
- **Seconds Precision:** Specify the number of digits in the fractional part of the datetime field. Seconds precision is used for TIMESTAMP data types only.
- **Size:** Specify the size of the varray.

Editing Varrays

To edit a varray, use the following steps:

1. From the Project Explorer, expand the Databases node and then the Oracle node.
2. Expand the module where you want to edit the Varray type.
3. Expand the User Defined Types node.
4. Expand the Varrays node.
5. Right-click the varray you want to edit and select **Open Editor**.

The Data Object Editor is displayed. Use the Name and Details tabs as defined in the Creating Varrays section on page 12-29 to edit the definition of the varray.

About Nested Tables

Nested table is an unordered collection of data elements. Nested tables enable you to have any number of elements. There is no maximum number of elements specified in the definition of the table. The order of the elements is not preserved. All the operations, such as SELECT, INSERT, and DELETE, that you perform on ordinary tables can be performed on nested tables. Elements of a nested table are stored in a

separate storage table containing a column that identifies the parent table row or object to which each element belongs. The elements may be built-in types or user-defined types. You can view a nested table as a single-column table, or if the nested table is an object type, as a multi-column table, with a column for each attribute of the object type.

Nested Tables are used to store an unordered set of elements that do not have a predefined size. For example, customer references.

Creating Nested Tables

To create a nested table:

1. From the Project Explorer, expand the Databases node and then the Oracle node.
2. Expand the module where you want to create the nested table.
3. Expand the User Defined Types node.
4. Right-click **Nested Tables**, and select **New**.

   The Data Object Editor is displayed. Use the following tabs on the Nested Table Details panel of the Data Object Editor to define the object type.

   - Name Tab
   - Details Tab

Name Tab

Follow the rules in "Naming Conventions for Data Objects" on page 4-4 to specify a name and an optional description.

Details Tab

Use the Details tab to specify the value for the following fields:

- **Length:** Specify the length of the nested table element. Length is applicable for character data types only.
- **Precision:** Specify the total number of digits allowed for the nested table element. Precision is applicable for numeric data types only.
- **Scale:** Specify the total number of digits to the right of the decimal point. Scale is applicable for numeric data types only.
- **Seconds Precision:** Specify the number of digits in the fractional part of the datetime field. Seconds precision is used for TIMESTAMP data types only.

Editing Nested Tables

To edit a nested table, use the following steps:

1. From the Project Explorer, expand the Databases node and then the Oracle node.
2. Expand the module where you want to edit the Nested Table.
3. Expand the User Defined Types node.
4. Expand the Nested Tables node.
5. Right-click the nested table you want to edit and select **Open Editor**.

The Data Object Editor is displayed. Use the Name and Details tabs as defined in the Creating Nested Tables section on page 12-30 to edit the definition of the nested table.

See also:

- Construct Object Operator
- Expand Object Operator

Configuring Data Objects

Earlier in the design phase, you defined a logical model for your target system using Warehouse Builder design objects. This section discusses how you assign physical properties to those design objects.

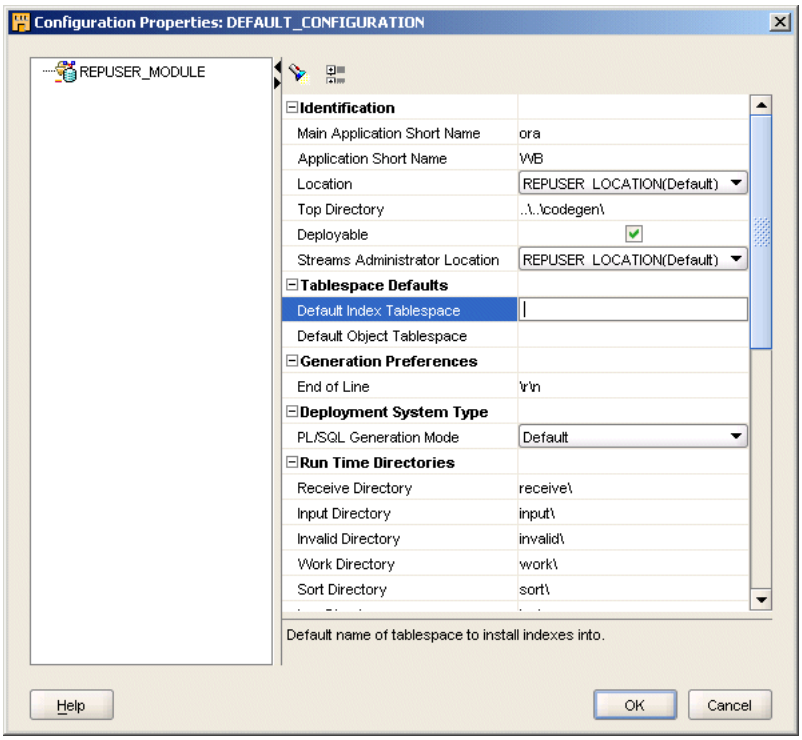
This section includes:

Configuring Warehouse Builder Design Objects

In this phase, you assign physical deployment properties to the object definitions you created in Warehouse Builder by configuring properties such as tablespaces, partitions, and other identification parameters. You also configure runtime parameters such as job names, and runtime directories.

Set these physical properties using the Configuration Properties dialog. Figure 12–11 displays the Configuration Properties dialog for an Oracle module. You can set properties for target modules or for individual design objects such as tables, dimensions, views, or mappings. The following sections show you how to assign physical properties to your logical design model.

Figure 12–11 Configuration Properties Dialog



Configuring Target Modules

Each target module provides top level configuration options for all the objects contained in that module.

To configure a Target Module:

1. From the Warehouse Builder Project Explorer, expand **Databases,** expand **Oracle,** and right-click a target module name and select **Configure.**

   Warehouse Builder displays the Configuration Properties dialog.

2. Choose the parameters you want to configure and click the space to the right of the parameter name to edit its value.

   For each parameter, you can either select an option from a list, type a value, or click the ellipses to display another properties dialog.

3. Configure the parameters listed in the following sections.

Identification

Main Application Short Name: This parameter is obsolete and is no longer used.

Application Short Name: This parameter is obsolete and is no longer used.

Location: Represents the location with which the module is associated. If the module is a source module, this value represents the location from which the data is sourced. If the module is a target module, this value represents the location to which the generated code and object data is deployed.

Top Directory: Represents the name of the directory to in which the generated code is stored. The default value for this parameter is `..\..\codegen\`. You can change this value to any directory in which you want to store generated code.

Deployable: Select this option to indicate that the objects contained in the module can be deployed.

Streams Administrator: This parameter will be used in future releases.

Tablespace Defaults

Default Index Tablespace: Defines the name of each tablespace where indexes are created. The default is null. If you configure an index tablespace at the target module level and not at the object level, Warehouse Builder uses the tablespace value configured at the target module level, during code generation. If you configure a tablespace for each index at the object level, Warehouse Builder overwrites the tablespace value configured at the target module level.

Default Object Tablespace: Defines the name of each tablespace where objects are created, for example, tables, views, or materialized views. The default is null. If you configure object tablespace at the target module level and not at the individual object level, Warehouse Builder uses the value configured at the target module level, during code generation. If you configure a tablespace for each individual object, Warehouse Builder overwrites the tablespace value configured at the target module level.

Generation Preferences

End of Line: Defines the end of line markers for flat files. This is dependent on the platform to which you are deploying your warehouse. For UNIX, use `\n`, and for NT, use `\r\n`.

Deployment System Type

PL/SQL Generation Mode: Defines the target database type. Code generation is based on the your choice in this field. For example, select Oracle 9*i* to ensure the use of Oracle 9*i* code constructs. If you select Oracle8*i*, Warehouse Builder generates row-based code.

Each release of Warehouse Builder introduces new functionality, some of which you may use only in conjunction with the latest version of the Oracle Database. For example, if you select Oracle8*i* as the PL/SQL Generation Mode, you cannot access some Oracle9*i* Warehouse Builder components such as Table Functions and External Tables. For a list of Oracle Warehouse Builder components not compatible with prior releases of the Oracle Database, refer to the *Oracle Warehouse Builder Release Notes.*

Run Time Directories

Receive Directory: Not currently used. The default is receive\.

Input Directory: Not currently used. The default is input\.

Invalid Directory: Directory for Loader error and rejected records. The default is invalid\.

Work Directory: Not currently used. The default is work\.

Sort Directory: Not currently used. The default is sort\.

Log Directory: Log directory for the SQL*Loader. The default is log\.

Archive Directory: Not currently used. The default is archive\.

Generation Target Directories

DDL Directory: Type a location for the scripts that create database objects in the target schema. The default is `ddl\`.

DDL Extension: Type a file name extension for DDL scripts. The default is `.ddl`.

DDL Spool Directory: Type a buffer location for DDL scripts during the script generation processing. The default is `ddl\log`.

LIB Directory: Type a location for the scripts that generate Oracle functions and procedures. The default is `lib\`.

LIB Extension: Type a suffix to be appended to a mapping name. The default is `.lib`.

LIB Spool Directory: Type a location for the scripts that generate user-defined functions and procedures. The default is `lib\log\`.

PL/SQL Directory: Type a location for the PL/SQL scripts. The default is `pls\`.

PL/SQL Run Parameter File: Type a suffix for the parameter script in a PL/SQL job. The default is `_run.ini`.

PL/SQL Spool Directory: Type a buffer location for PL/SQL scripts during the script generation processing. The default is `pls\log\`.

PL/SQL Extension: Type a file name extension for PL/SQL scripts. The default is `.pls`.

Staging File Directory: For all ABAP configuration related to SAP tables, see Chapter 18, "Importing Data From Third Party Applications".

ABAP Extension: File name extension for ABAP scripts. The default is `.abap`.

ABAP Run Parameter File: Suffix for the parameter script in an ABAP job. The default is _run.ini.

ABAP Spool Directory: The location where ABAP scripts are buffered during script generation processing.

LOADER Directory: Type a location for the control files. The default is `ctl\`.

LOADER Extension: Type a suffix for the loader scripts. The default is `.ctl`.

LOADER Run Parameter File: Type a suffix for the parameter initialization file. The default is `_run.ini`.

Configuring Tables

Warehouse Builder generates DDL scripts for each table defined in a target module. Follow these steps to configure a table.

To configure the physical properties for a table, right-click the name of a table and select **Configure.** Warehouse Builderdisplays the Configuration Properties dialog. Set the configuration parameters listed in the following sections.

Identification

- **Deployable:** Select this option to indicate that you want to deploy this table. Warehouse Builder generates scripts only for table constraints marked deployable.

Storage Parameters

Storage parameters enable you to define how the table is stored in the database. This category contains parameters such as BUFFER_POOL, FREELIST GROUPS, FREELISTS, INITIAL, MINEXTENTS, MAXEXTENTS, NEXT, and PCTINCREASE.

The **Tablespace** parameter defines the name of each tablespace where the table is created. The default value is null. If you accept the default value of null, Warehouse Builder generates the table based on the tablespace value set in the target module

configuration properties. If you configure the tablespace for individual objects, Warehouse Builder overwrites the tablespace value configured for the target module.

Parallel

- **Parallel Access Mode:** Enables parallel processing when the table has been created. The default is `PARALLEL`.
- **Parallel Degree:** Indicates the degree of parallelism. This is the number of parallel threads used in the parallel operation.

Performance Parameters

- **Buffer Cache:** Indicates how Oracle should store rows in the buffer cache.
- **Data Segment Compression:** Indicates whether data segments should be compressed. Compressing reduces disk use. The default is NOCOMPRESS.
- **Logging Mode:** Indicates whether the DML actions are logged in the redo log file. To improve performance, set this parameter to `NOLOGGING`. The default is `LOGGING`.
- **Statistics Collection:** Indicates if statistics should be collected for the table. Specify MONITORING if you want modification statistics to be collected on this table.
- **Row-level Dependency:** Indicates whether row-level dependency tracking.
- **Row Movement:** Indicates if the Oracle server can move a table row.

Partition Parameters

- **Partition Tablespace List:** Specify a comma-delimited list of tablespaces. For simple partitioned objects, it is used for a HASH BY QUANTITY partition tablespace. For composite partitioned tables, it is used for sub-partition template to store the list of tablespaces.
- **Overflow Tablespace List:** Specify a comma separated list of tablespaces for overflow data. For simple-partitioned objects, it is used for HASH BY QUANTITY partition overflow tablespaces. The number of tablespaces does not have to equal the number of partitions. If the number of partitions is greater than the number of tablespaces, then Oracle cycles through the names of the tablespaces.

Shadow Table

- **Shadow Table Name:** Indicates the name of the shadow table that stores the rows that were not loaded into the table during a load operation.
- **Tablespace:** Indicates the name of the tablespace in which the shadow table is stored.

Configuring Materialized Views

To configure the physical properties for a materialized view:

1. From the Project Explorer, right-click a materialized view name and select **Configure.**

   The Configuration Property window is displayed.

2. Follow the configuration guidelines listed for tables. For more information, see "Configuring Tables" on page 12-34.
3. Configure the Materialized View Parameters listed in the following section.

Materialized View Parameters

The following are parameters for materialized views:

Materialized View Parameters

- **Start With:** Indicates the first automatic refresh time. Specify a datetime value for this parameter.
- **Refresh On:** The options are COMMIT and DEMAND. Specify COMMIT to indicate that a fast refresh is to occur whenever the database commits a transaction that operates on a master table of the materialized view. Specify DEMAND to indicate that the materialized view should be refreshed on demand. You can do this by using one of the refresh procedures of the DBMS_MVIEW package. The default setting is DEMAND.
- **Query Rewrite:** Indicates if the materialized view is eligible for query rewrite. The options are ENABLE and DISABLE. the default is DISABLE.

  Enable: Enables query rewrite. For other query rewrite requirements, see "Fast Refresh for Materialized Views" on page 12-37.

  Disable: Disables query rewrite. You can disable query rewrite when you know that the data in the materialized view is stale or when you want to make changes to the query statement.
- **Default Rollback Segment:** The options are DEFAULT, DEFAULT MASTER, DEFAULT LOCAL, and NONE. The default setting is DEFAULT LOCAL. Specify DEFAULT to indicate that the Oracle Database should choose which rollback segment to use. Specify DEFAULT MASTER for the remote rollback segment to be used at the remote site. Specify DEFAULT LOCAL for the remote rollback segment to be used for the local refresh group that contains the materialized view. Specify NONE to name both master and local segments.
- **NEXT (date):** Indicates the interval between automatic refreshes. Specify a datetime value for this parameter.
- **Using Constraints:** The options you can select for this parameter are TRUSTED or ENFORCED. Choose TRUSTED to allow Oracle to use dimension and constraint information that has been declared trustworthy by the DBA but has not been validated by Oracle. ENFORCED is the default setting.
- **REFRESH:** Indicates the refresh method. The options are Complete, Fast, Force, and Never. The default setting is Force.

  Complete: The Oracle server truncates the materialized view and re-executes the query upon refresh.

  Fast: Uses materialized views to only apply changes to the base table data. There are a number of requirements for fast refresh to operate properly. For more information, see "Fast Refresh for Materialized Views" on page 12-37.

  Force: The Oracle server attempts to refresh using the fast mode. If unable to refresh in fast mode, the Oracle server re-executes the query upon refresh.

  Never: Prevents the materialized view from being refreshed.
- **WITH:** Choose PRIMARY_KEY to create a primary key materialized view. Choose ROWID to create a ROWID materialized view. The default setting is PRIMARY_KEY.
- **FOR UPDATE:** Choose Yes to allow a subquery, primary key, rowid, or object materialized view to be updated. The default setting is No.

- **Master Rollback Segment:** Indicates the name of the remote rollback segment to be used at the remote master site for the materialized view.
- **Base Tables:** Specify a comma-delimited list of base tables referenced by the materialized view. Separate each table name with a comma. If a table name is not in upper case, enclose the name in double quotes.
- **Local Rollback Segment:** Specify a named remote rollback segment to be used for the local refresh group of the materialized view. The default is null.
- **BUILD:** Indicates when the materialized view is populated. The options are Immediate (default), Deferred, and Prebuilt.

  Immediate: Populates the materialized view when it is created.

  Deferred: Delays the population of the materialized view until the next refresh operation. You can select this option when you are designing a materialized view and the metadata for the base tables is correct but the data is not.

  Prebuilt: Indicates that the materialized view is prebuilt.

Performance Parameters:

Logging Mode: Indicates whether the DML actions are logged in the redo log file. To improve performance, set this parameter to `NOLOGGING`. The default is `LOGGING`.

Shadow Table

- **Shadow Table Name:** Indicates the name of the shadow table that stores the rows that were not loaded into the table during a load operation.
- **Tablespace:** Indicates the name of the tablespace in which the shadow table is stored.

Parallel

- **Parallel Access Mode:** Enables parallel processing when the table has been created. The default is `PARALLEL`.
- **Parallel Degree:** Indicates the degree of parallelism. This is the number of parallel threads used in the parallel operation.

Identification

- **Deployable:** Select TRUE to indicate if you want to deploy this materialized view. Warehouse Builder generates scripts only for materialized views marked deployable.

Hash Partition Parameters

- **Hash Partition Tablespace List:** Indicates the tablespace that stores the partition or sub partition data. To specify multiple tablespaces, use a comma separated list.

Fast Refresh for Materialized Views

You can configure a materialized view in Warehouse Builder to refresh incrementally. When you update the base tables for a materialized view, the database stores updated record pointers in the materialized view log. Changes in the log tables are used to refresh the associated materialized views.

To ensure incremental refresh of materialized views in Warehouse Builder, verify the following conditions:

- The Refresh parameter must be set to 'Fast' and the Base Tables parameter must list all base tables.
- Each base table must have a PK constraint defined. Warehouse Builder generates a create statement based on the PK constraint and utilizes that log to refresh the dependent materialized views.
- The materialized view must not contain references to non-repeating expressions such as SYSDATE, ROWNUM, and non-repeatable PL/SQL functions.
- The materialized view must not contain references to RAW and LONG RAW data types.
- There are additional restrictions for materialized views with statements for joins, aggregations, and unions. For information on additional restrictions, refer to the Oracle9*i Data Warehousing Guide.*

Configuring Views

Warehouse Builder generates a script for each view defined in a target module. You can configure whether to deploy specific views or not by setting the Deployable parameter to TRUE or FALSE.

For more information on views, refer to the following sections:

- About Views on page 12-17
- Creating View Definitions on page 12-17

Configuring Sequences

Warehouse Builder generates a script for each sequence object. A sequence object has a Start With and Increment By parameter. Both parameters are numeric.

To configure the physical properties for a sequence:

1. Right-click the name of a sequence and select **Configure.**

   The Configuration Properties dialog is displayed.

2. Configure the following Sequence parameters:

   Increment By: The number by which you want to increment your sequence.

   Start With: The number at which you want the sequence to start.

3. Configure the following Identification parameters:

   Deployable: Select this option to indicate that you want to deploy this sequence. Warehouse Builder only generates scripts for sequences marked deployable.

13

Defining Dimensional Objects

Warehouse Builder enables you to define, deploy, and load dimensional objects. You can deploy dimensional objects either to a relational schema or to an analytical workspace in the database.

This chapter includes:

Creating Dimensions

To create a dimension in Warehouse Builder, use one of the following methods:

- Using the Create Dimension Wizard

  The wizard enables you to create a fully functional dimension object quickly. When you use the wizard, Warehouse Builder defaults many settings to the most commonly used values. You can modify these settings later using the Data Object Editor. If you choose a relational implementation for the dimension, the implementation tables are also created in the Warehouse Builder repository using auto binding.

  For more information about the defaults used by the Dimension wizard, see "Defaults Used By the Create Dimension Wizard" on page 13-7.

- Using the Data Object Editor

  The Data Object Editor gives you full control over all aspects of the dimension definition and implementation. This provides maximum flexibility. You use the editor to create a dimension from scratch or to edit a previously created dimension.

- Using the Time Dimension wizard

  The Time Dimension wizard enables you to create and populate time dimensions. For more information on the Time Dimension wizard, see "Creating Time Dimensions" on page 13-35.

Using the Create Dimension Wizard

To create a dimension using the Dimension wizard:

1. From the Warehouse Builder Project Explorer expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the dimension.
3. Right-click **Dimensions**, select **New**, and then **Using Wizard**.

   Warehouse Builder displays the Welcome page of the Create Dimension wizard. Click **Next** to proceed. The wizard guides you through the following pages:

   - Name and Description Page on page 13-2
   - Storage Type Page on page 13-2
   - Dimension Attributes Page on page 13-3
   - Levels Page on page 13-4
   - Level Attributes Page on page 13-5
   - Slowly Changing Dimension Page on page 13-6
   - Pre Create Settings Page on page 13-6
   - Dimension Creation Progress Page on page 13-7
   - Summary Page on page 13-7

Name and Description Page

You use the Name and Description page to describe your dimension. Enter the following information on this page:

- **Name:** This is the name that Warehouse Builder uses to refer to the dimension. The dimension name must be unique within a module.
- **Description:** You can type an optional description for the dimension.

Storage Type Page

Use the Storage Type page to specify the type of storage for the dimension. The storage type determines how the dimension data is physically stored in the database. The options you can select for storage type are as follows:

- Relational storage (ROLAP)
- Multidimensional storage (MOLAP)

You select the storage type based on the volume of data stored at the lowest level of the entire cube and the refresh rate required.

Relational storage (ROLAP) Warehouse Builder stores the dimension definition and its data in a relational form in the database. Select this option to create a dimension that uses a relational or ROLAP implementation.

Relational storage is preferable if you want to store detailed, high volume data or you have high refresh rates combined with high volumes of data. Use relational storage if you want to perform one of the following:

- Store detailed information such as call detail records, point of sales (POS) records and other such transaction oriented data.
- Refresh high volumes of data at short intervals.

- Detailed reporting such as lists of order details.
- Ad-hoc queries in which changing needs require more flexibility in the data model.

Operational data stores and enterprise data warehouses are typically implemented using relational storage. You can then derive multi-dimensional implementations from this relational implementation to perform different analysis types.

When you choose a relational implementation for a dimension, Warehouse Builder creates the implementation tables used to store the dimension data. The default implementation of the dimension is using a star schema. This means that the data for all the levels in the dimension is stored in a single database table.

Multidimensional storage (MOLAP) Warehouse Builder stores the dimension definition and dimension data in an analytic workspace in the database. Select this option to create a dimension that uses a MOLAP implementation.

Multidimensional storage is preferable when you want to store aggregated data for analysis. The refresh intervals for a multidimensional storage are usually longer than relational storage as data needs to be pre-calculated and pre-aggregated. Also, the data volumes are typically smaller due to higher aggregation levels. Use multidimensional storage to perform the following:

- Advanced analysis such as trend analysis, what-if analysis, or to forecast and allocate data.
- Constant analysis using a well defined consistent data model with fixed query patterns.

When you choose a MOLAP implementation, Warehouse Builder stores the dimension in an analytic workspace that uses the same name as the Oracle module to which the dimension belongs. The tablespace that is used to store the analytic workspace is the tablespace that is defined as the users tablespace for the schema that contains the dimension metadata.

Note: For information about certain limitations of deploying dimensions to the OLAP catalog, see the *Oracle Warehouse Builder Release Notes*.

Dimension Attributes Page

You use the Dimension Attributes page to define the dimension attributes. A dimension attribute is applicable to one or more levels in the dimension. By default, Warehouse Builder creates the following attributes for each dimension: ID, Name, and Description. You can rename the ID attribute or delete it.

Specify the following details for each dimension attribute:

- **Name:** This is the name of the dimension attribute. The name must be unique within the dimension.
- **Description:** Type an optional description for the dimension attribute.
- **Identifier:** Select the type of dimension attribute. Select one of the following options:

  Surrogate: Indicates that the attribute is the surrogate identifier of the dimension.

  Business: Indicates that the attribute is the business identifier of the dimension

Parent: In a value-based hierarchy, indicates that the attribute stores the parent value of an attribute. You can create value-based hierarchies only when you choose a MOLAP implementation for the dimension.

If the attribute is a regular dimension attribute, leave this field blank.

The options displayed in the Identifier list depend on the type of dimension. When you create a dimension with a relational or ROLAP implementation, only the Surrogate and Business options are displayed. For MOLAP dimensions, only the Business and Parent options are displayed.

- **Data Type:** Select the data type of the dimension attribute from the drop-down list.

  Note: The following data types are not supported for MOLAP implementations: BLOB, Interval Day to Second, Interval Year to Month, RAW, Timestamp with Time Zone, Timestamp with Local Time Zone.

- **Length:** For character data types, specify the length of the attribute.
- **Precision:** For numeric data types, define the total number of digits allowed for the column.
- **Scale:** For numeric data types, define the total number of digits to the right of the decimal point.
- **Seconds Precision:** Represents the number of digits in the fractional part of the datetime field. It can be a number between 0 and 9. The seconds precision is used only for TIMESTAMP data types.
- **Descriptor:** Select the type of descriptor. The options are: Short Description, Long Description, End date, Time span, Prior period, and Year Ago Period.

  Descriptors are very important for MOLAP implementations. For example, in a custom time dimension, you must have Time Span and End Date to allow time series analysis.

Levels Page

The Levels page defines the levels of aggregation in the dimension. A dimension must contain at least one level. the only exception is value-based hierarchies that contain no levels. You can create a value-based hierarchy using the Data Object Editor only.

Enter the following details on the Levels page:

- **Name:** This is the name of the level. The level name must be unique within the dimension.
- **Description:** Type an optional description for the level.

List the levels in the dimension such that the parent levels appear above the child levels. Use the arrow keys to move levels so that they appear in this order.

Warehouse Builder creates a default hierarchy called STANDARD that contains the levels in the same order that you listed them on the Levels page. The attributes used to store the parent key references of each level are also created. For a relational or ROLAP dimension, Warehouse Builder creates two attributes, one for the surrogate identifier and one for the business identifier, that correspond to the parent level of each level. For a MOLAP dimension, for each level, one attribute that corresponds to the business identifier of the parent level is created.

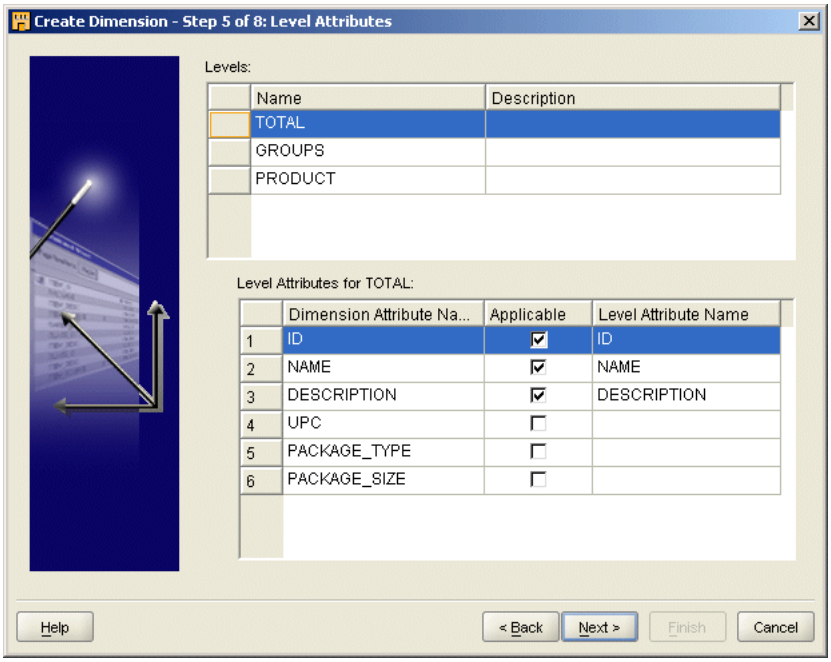
For example, the Products dimension contains the following levels: Total, Groups, and Product. If you choose a relational or ROLAP implementation, Warehouse Builder creates two attributes, in both the Product and Groups levels, that reference the surrogate identifier and business identifier of the parent level. If you choose a MOLAP implementation, Warehouse Builder creates an attribute in both the Product and Groups levels, that reference the business identifier of the parent level.

Note: To create additional hierarchies, you use the Hierarchies tab of the Data Object Editor as described in Hierarchies Tab on page 13-12.

Level Attributes Page

The Level Attributes page shown in Figure 13–1 defines the level attributes of each dimension level. You define level attributes by selecting the dimension attributes that apply to the level. The dimension attributes are defined on the Dimension Attributes page of the Create Dimension wizard.

Figure 13–1 Level Attributes Page of the Create Dimension Wizard



The Level Attributes page contains two sections: Levels and Level Attributes.

Levels The Levels section lists all the levels defined in the Levels page of the Create Dimension wizard. Select a level in this section to specify the dimension attributes that this level implements. You select a level by clicking the level name.

Level Attributes The Level Attributes section lists all the dimension attributes defined in the Dimension Attributes page. For each level, choose the dimension attributes that the level implements. To indicate that a dimension attribute is implemented by a level, select the **Applicable** option for the dimension attribute. The name of the level attribute can be different from that of the dimension attribute. Use the **Level Attribute Name** field to specify the name of the level attribute.

For example, to specify that the dimension attributes ID, Name, Description, and Budget are implemented by the State level:

1. Select the State level in the Levels section.
2. In the Level Attributes section, select the **Applicable** option for the attributes ID, Name, Description, and Budget.

By default, Warehouse Builder uses the following defaults:

- The attributes ID, Name, and Description are applicable to all levels.
- All dimension attributes are applicable to the lowest level in the dimension.

Slowly Changing Dimension Page

Use of this functionality requires the Warehouse Builder Enterprise ETL Option.

The Slowly Changing Dimension page enables you to define the type of slowly changing policy used by the dimension. This page is displayed only if you had chosen **Relational storage (ROLAP)** as the storage type on the Storage Type Page. For more information on slowly changing dimensions concepts, see "About Slowly Changing Dimensions" on page 4-38.

Select one of the following options for the slowly changing policy:

- **Type 1: Do not store history:** This is the default selection. Warehouse Builder creates a dimension that stores no history. This is a normal dimension.
- **Type 2: Store the complete change history:** Select this option to create a Type 2 slowly changing dimension. Warehouse Builder creates the following two additional dimension attributes and makes them applicable for the lowest level in the Type 2 SCD:
  - Effective date
  - Expiration date

  All the attributes of the lowest level in the Type 2 SCD, except the surrogate and business identifier, are defined as the triggering attributes.

  Note: You cannot create a Type 2 or Type 3 slowly changing dimension if the type of storage is MOLAP.

- **Type 3: Store only the previous value:** Select this option to create a Type 3 slowly changing dimension. Warehouse Builder assumes that all the level attributes at the lowest level, excluding the surrogate ID and business ID, should be versioned. For each level attribute that is versioned, Warehouse Builder creates an additional attribute to store the previous value of the attribute.

Pre Create Settings Page

The Pre Create Settings page displays a summary of the options selected on the previous pages of the Create Dimension wizard. This includes the attributes, levels, hierarchies, storage type, and the slowly changing policy used for the dimension. Warehouse Builder uses these settings to create the dimension definition and the database tables that implement the dimension. It also binds the dimension attributes to the table columns that store the attribute data.

Click **Next** to proceed with the implementation of the dimension. To change any of the options you previously selected, click **Back**.

Note: Review this page carefully as it summarizes the implementation and its objects.

Dimension Creation Progress Page

The Dimension Creation Progress page displays the progress of the dimension implementation that was started on the Pre-Create Settings page. The Message Log section on this page provides information about the individual tasks completed during the dimension implementation. Click **Next** to proceed.

Summary Page

The Summary page provides a brief summary of the options that you selected using the Create Dimension wizard. Use the Summary page to review the selected options. Click **Finish** to create the dimension. You now have a fully functional dimension. This dimension is displayed under the Dimensions node of the Project Explorer.

Warehouse Builder creates the metadata for the following in the repository:

- The dimension object.
- The tables that store the dimension data.

  For a relational implementation, a database table that stores the dimension data is created. Warehouse Builder binds the attributes in the dimension to the database columns used to store their values.

  For a MOLAP implementation, the analytic workspace that stores the dimension data is created.
- The database sequence used to generate the surrogate identifier for all the dimension levels.

Note that Warehouse Builder creates the definitions of these objects in the repository and not the objects themselves.

Deploying Dimensions To create the dimension in the target schema, you must deploy the dimension. For a ROLAP dimension, ensure that you deploy the sequence and the implementation tables before you deploy the dimension. Alternatively, you can deploy all these objects at the same time. For more information see "MOLAP Implementation of Dimensional Objects" on page 4-28.

Note: When you delete a dimension, the associated objects such as sequence, database tables, or AWs are not deleted. You must explicitly delete these objects.

Defaults Used By the Create Dimension Wizard

When you create a dimension using the Create Dimension wizard, Warehouse Builder sets default values for some of the attributes that are used to create the dimension. The following sections describe the defaults used.

Storage For a relational storage, Warehouse Builder uses the star schema as the default implementation method.

When you choose multidimensional storage, Warehouse Builder stores the dimension in an analytic workspace that has the same name as the Oracle module in which the dimension is defined. If the analytic workspace does not exist, Warehouse Builder

creates it. The analytic workspace is stored in the users tablespace of the schema that owns the Oracle module.

Dimension Attributes Warehouse Builder creates default dimension attributes with the properties specified in Table 13–1.

Table 13–1 Default Dimension Attributes

Dimension Attribute Name	Identifier	Data Type
ID	Surrogate	NUMBER
Name	Business	VARCHAR2
Description		VARCHAR2

You can add additional attributes. For your dimension to be valid, you must define the surrogate and business identifiers.

Hierarchies Warehouse Builder creates a default hierarchy called STANDARD that contains all the levels listed on the Levels page of the Create Dimension wizard. The hierarchy uses the levels in the same order that they are listed on the Levels page.

Level Attributes The ID, Name, and Description attributes are applicable to each level defined in the dimension. All the dimension attributes are applicable to the lowest level in the dimension. The lowest level is the level that is defined last on the Levels page.

Slowly Changing Dimensions When you create a Type 2 SCD, all the attributes of the lowest level, except the surrogate identifier and the business identifier, are versioned. Two additional attributes are created to store the effective date and the expiration date of each record. For example, if you create the Products dimension described in "Dimension Example" as a Type 2 SCD, the attributes UPC, Package_type, and Package_size are versioned. Warehouse Builder creates two additional attributes called EXPIRATION_DATE and EFFECTIVE_DATE, of data type DATE, to store the effective date and expiration date of versioned records.

For a Type 3 SCD, all level attributes of the lowest level, except the surrogate identifier and the primary identifier, are versioned. Warehouse Builder creates additional attributes to store the previous value of each versioned attribute. In addition to this, and attribute to store the effective date is created. For example, if you create the Products dimension described in "Dimension Example" as a Type 3 SCD, Warehouse Builder creates additional attributes called PREV_DESCRIPTION, PREV_PACKAGE_TYPE, PREV_PACKAGE_SIZE, and PREV_UPC to store the previous values of the versioned attributes. These data type for these attributes are the same the ones used to store the current value of the attribute. Warehouse Builder also creates an attribute EFFECTIVE_TIME to store the effective time of versioned records. This attribute uses the DATE data type.

Implementation Objects For each dimension, in addition to the dimension object, certain implementation objects are created. The number and type of implementation objects depends on the storage type of the dimension.

For a relational storage, the following implementation objects are created:

Table: A table with the same name as the dimension is created to store the dimension data. A unique key is created on the dimension key column. For example, when you define a dimension called CHANNELS, a table called CHANNELS is created to store

the dimension data. Also, a unique key called CHANNELS_DIMENSION_KEY_PK is created on the dimension key column.

Sequence: For a dimension that uses a relational storage, a sequence that loads the dimension key values is created. For example, for the dimension called CHANNELS, a sequence called CHANNELS_SEQ is created.

For a multidimensional storage, if it does not already exist, Warehouse Builder creates an analytic workspace with the same name as the Oracle module that contains the dimension. For example, if you create a dimension called PRODUCTS in the SALES_WH module, the dimension is stored in an analytic workspace called SALES_WH. If an analytic workspace with this name does not already exist, Warehouse Builder first creates it and then stores the dimension in this analytic workspace.

For time dimensions, irrespective of the storage type, Warehouse Builder creates a map that loads the time dimension. The name of the map is the dimension name followed by '_MAP'. For example, the map that loads a time dimension called TIMES will be called TIMES_MAP.

Using the Data Object Editor

The Data Object Editor enables advanced users to create dimensions according to their requirements. You can also edit a dimension using the Data Object Editor. Use the Data Object Editor to create a dimension if you want to perform one of the following:

- Use the snowflake implementation methods.
- Create value-based hierarchies.
- Create dimension roles.
- Skip levels in a hierarchy.
- Use existing database tables or views to store the dimension data. This is referred to as manual binding.

To define a dimension using the Data Object Editor:

1. From the Warehouse Builder Project Explorer expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the dimension.
3. Right-click **Dimensions**, select **New,** then **Using Editor**.

   Warehouse Builder displays the Add a New or Existing Dimension dialog.
4. Select the **Create a New Dimension** option. Enter the name of the dimension and select the module to which the dimension belongs.
5. Click **OK**.

   Warehouse Builder displays the Data Object Editor. To define a dimension, provide information on the following tabs of the Dimension Details panel:

   - Name Tab on page 13-10
   - Storage Tab on page 13-10
   - Attributes Tab on page 13-11
   - Levels Tab on page 13-11
   - Hierarchies Tab on page 13-12
   - SCD Tab on page 13-13

- Data Viewer Tab on page 13-14

When you use the Data Object Editor to create a dimension that has a relational implementation, Warehouse Builder does not automatically create the physical structures that store the dimension data. You must create these structures.

6. For dimensions that have a relational implementation, bind the attributes in the dimension to the database columns that store their data, see "Binding Attributes" on page 13-14.

Name Tab

You use the Name tab to describe your dimension. You also specify the type of dimension and the dimension roles on this tab.

The **Name** field represents the name of the dimension. The dimension name must be unique within the module. You use the **Description** field to enter an optional description for the dimension.

Dimension Roles You use the Dimension Roles section to define dimension roles. For more information about dimension roles, see "Dimension Roles" on page 4-33. You define the following for each dimension role:

- **Name:** Represents the name of the dimension role.
- **Description:** Specify an optional description for the dimension role.

Storage Tab

You use the Storage tab to specify the type of storage for the dimension. The storage options you can select are described in the following sections.

Relational: Relational data structures Select the Relational option to store the dimension and its data in a relational form in the database. Use this option to create a dimension that uses a relational or ROLAP implementation.

For a relational storage, you can select one of the following methods to implement the dimension:

- **Star schema:** Implements the dimension using a star schema. This means that the dimension data is stored in a single database table or view.
- **Snowflake schema:** Implements the dimension using a snowflake schema. This dimension data is stored in more than one database table or view.
- **Manual:** You must explicitly bind the attributes from the dimension to the database object that stores their data.

When you perform auto binding, Warehouse Builder uses these storage settings to perform auto binding.

Click **Create composite unique key** to create a composite unique key on the business identifiers of all levels. For example, if your dimension contains three levels, when you create a composite unique key, Warehouse Builder creates a unique key that includes the business identifiers of all three levels. Creating a composite unique key enforces uniqueness of a dimension record across the dimension at the database level.

MOLAP: Multi dimensional data structures Select the MOLAP option to store the dimension and its data in a multidimensional form in the database. Use this option to create a dimension that uses a MOLAP implementation. The dimension data is stored in an analytic workspace.

Enter values for the following fields:

- **AW Name:** Enter the name of the analytic workspace that stores the dimension data. Alternately, you can click the Ellipsis button to display a list of MOLAP objects in the current project. Warehouse Builder displays a node for each module in the project. Expand a module to view the list of dimensional objects in the module. Selecting an object from list stores the dimension in the same analytic workspace as the selected object.
- **AW Tablespace Name:** Enter the name of the tablespace in which the analytic workspace is stored.

Dimensions with multiple hierarchies can sometimes use the same source column for aggregate levels (that is, any level above the base). In such cases, you select the **Generate surrogate keys in AW** option. During a load operation, Warehouse Builder adds the level name as a prefix to each value. It is recommended that you select this option unless you know that every dimension member is unique.

If you are sure that dimension members are unique across levels, then you can use the exact same names in the analytic workspace as the source. For example, if your relational schema uses numeric surrogate keys to assure uniqueness, then there is no need to create new surrogate keys in the analytic workspace. The **Use natural keys from data source** option enables you to use the same natural keys from the source in the analytic workspace.

Note: If you edit a dimension and change the Storage type from ROLAP to MOLAP, Warehouse Builder changes the data type of the surrogate identifier to VARCHAR2.

Attributes Tab

Use the Attributes tab to define the dimension attributes. The Attributes tab contains two sections: Sequence and Dimension Attributes.

Sequence You need to select a sequence only for dimensions that have a relational implementation. Use the Sequence field to specify the name of the database sequence that populates the dimension key column. Click **Select** to the right of this field to display the Available Sequences dialog. This dialog contains a node for each module in the project. Expand a module node to view the sequences contained in the module. Select a sequence from the displayed list.

Dimension Attributes Use the Dimension Attributes section to define the details of the dimension attributes as described in "Dimension Attributes Page" on page 13-3.

Levels Tab

You use the Levels tab to define the level attributes for each level in the dimension. You also use this tab to create value-based hierarchies. Select the **Value-based** option to create a value-based hierarchy. A dimension with a value-based hierarchy can contain only one level.

Before you define level attributes, ensure that the dimension attributes are defined on the Dimension Attributes tab. To define the level attributes for a level, you need to select the dimension attributes that the level implements. The Levels tab contains two sections: Levels and Level Attributes.

Levels The Levels section displays the levels in the dimension. Provide the following details for each level:

- **Name:** Enter the name of the dimension level. The name must be unique within the dimension.
- **Description:** Enter an optional description for the level.

Level Attributes The Level Attributes section lists all the dimension attributes defined on the Attributes tab. The values that you specify in this section are applicable to the level selected in the Levels section. The Level Attributes section contains the following:

- **Dimension Attribute Name:** Represents the name of the dimension attribute.
- **Applicable:** Select the Applicable option if the level selected in the Levels section implements this dimension attribute.
- **Level Attribute Name:** Represents the name of the level attribute. You use this field to specify a name for the level attribute, a name that is different from that of the dimension attribute. This is an optional field. If you do not specify a name, the level attribute will have the same name as the dimension attribute.
- **Description:** Specify an optional description for the level attribute.

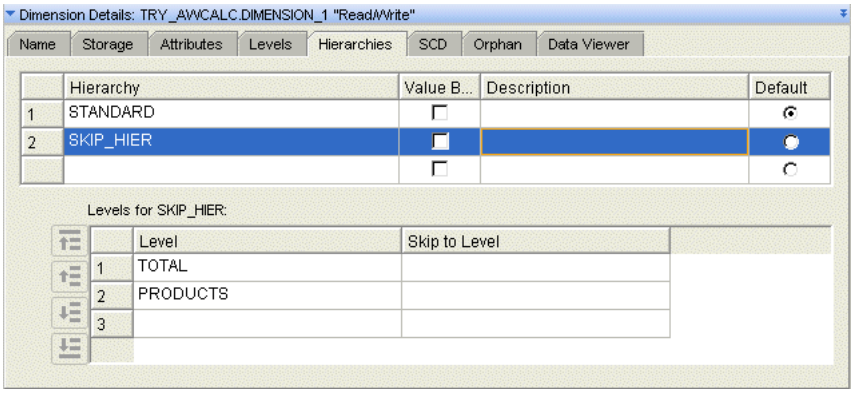
For example, to specify that the Groups level implements the dimension attributes ID, Name, and Description:

- Select the Groups level in the **Levels** section.
- In the Level Attributes section, select the **Applicable** option for the ID, Name, and Description attributes.

Hierarchies Tab

You use the Hierarchies tab to create dimension hierarchies. The Hierarchies tab, shown in Figure 13–2, contains two sections: Hierarchies and Levels.

Figure 13–2 Hierarchies Tab of the Data Object Editor



Hierarchies Use the Hierarchies section to define the hierarchies in the dimension. For each hierarchy, define the following:

- **Hierarchy:** Represents the name of the hierarchy. To create a new hierarchy, enter the name of the hierarchy in this field.
- **Value-based:** Select this option to create a value-based hierarchy. A value-based hierarchy contains no levels. It must have an attribute identified as the parent identifier. Since you can create value-based hierarchies only for MOLAP dimensions, this option is displayed only if you choose **MOLAP:**

Multidimensional data structures on the Storage tab. For more information on value-based hierarchies, see "Value-based Hierarchies" on page 4-34.

- **Description:** Enter an optional description for the hierarchy.
- **Default:** Select the Default option if the hierarchy is the default hierarchy for the dimension. When a dimension has more than one hierarchy, query tools show the default hierarchy. It is recommended that you set the most commonly used hierarchy as the default hierarchy.

To delete a hierarchy, right-click the cell to the left of the Hierarchy field and select **Delete**. Alternately, you can select the hierarchy by clicking the cell to the left of the Hierarchy field and press the Delete button.

When you create a hierarchy, ensure that you create the attributes that store the parent level references for each level. For a relational or ROLAP dimension, create two attributes to store the surrogate identifier reference and business identifier reference of each level. For a MOLAP dimension, create one attribute to store the reference to the business identifier of the parent level of each level.

Levels The Levels section lists all the levels defined on the Levels tab of the Data Object Editor. Use this section to specify the levels used in each hierarchy. The Levels section contains the following:

- **Level:** Represents the name of the level. Click the drop-down list to display all the levels defined in the dimension.
- **Skip to Level:** Represents the parent level of the level indicated by the Level field. Use this field to define skip-level hierarchies.

  For example, the Products dimension contains the following hierarchy:

  Total > Product

  This hierarchy does not include the Groups level. Thus the Product level must skip the Groups level and use the Total level as a parent. To create this hierarchy, select the Product level in the **Level** field and select Total from the **Skip to Level** drop-down list.

Use the arrows to the left of the Levels section to change the order in which the levels appear in the section.

SCD Tab

You use this tab to specify the type of slowly changing policy that the dimension implements. Since you can create a slowly changing dimension only for dimensions that use a relational implementation, the options on this tab are enabled only if you choose **Relational: relational data structures** on the Storage tab. The options that you can select for slowly changing policy are as follows:

- **Type 1: Do not keep history:** Creates a normal dimension that stores no history.
- **Type 2: Store the complete change history:** Select this option to create a Type 2 SCD. Click **Settings** to specify the additional details such as triggering attribute, effective date and expiration date for each level.as described in "Creating a Type 2 SCD" on page 13-15.
- **Type 3: Store only the previous value:** Select this option to create a Type 3 SCD. Click **Settings** to specify the additional details such as effective date and the attributes used to store the previous value of versioned attributes as described in "Creating a Type 3 SCD" on page 13-17.

Note: You cannot create a Type 2 or Type 3 slowly changing dimension if you have specified the type of storage as MOLAP.

When you create a Type 2 or Type 3 SCD using the Data Object Editor, you must create the dimension attributes that store the effective data and expiration date and apply them to the required levels.

Data Viewer Tab

You use the Data Viewer tab to view the dimension data. Click the **Execute** button to display the data viewer for the dimension. The data viewer displays the data stored in all the levels and hierarchies of the dimension. Before you attempt to view the data stored in the dimension, ensure that you have deployed the dimension and executed the map that loads the dimension.

Binding Attributes

After you define the dimension structure, you must specify the details of the database tables or views that store the dimension data. You can choose one of the following options for binding dimension attributes to the database columns that store their data:

- Auto binding
- Manual binding

Auto Binding When you perform auto binding, Warehouse Builder automatically maps the attributes in the dimension to the database columns that store their data.

To perform auto binding, in the Data Object Editor, select **Object** and then **Bind**. You can alternately right-click the dimension object in the Implementation tab of the graphical canvas and select **Bind**. For more information on the auto binding rules, see "Auto Binding" on page 4-27.

Manual Binding In manual binding, you must explicitly bind the attributes in each level of the dimension to the database columns that store their data. You can either bind to existing tables or create new tables and bind to them.

To perform manual binding:

1. In the Dimensional tab of the Data Object Editor, right-click the dimension and select **Detail View**.

   Warehouse Builder displays the Implementation tab. This tab displays the dimension.

2. Drag and drop the operator that represents the database object that stores the dimension data.

   For example, if the dimension data is stored in a table, drag and drop the Add New or Existing Table icon onto the Implementation canvas. Warehouse Builder displays the Add a new or existing Table dialog. To store the dimension data, you either select an existing table or create a new table.

3. Repeat Step 2 as many times as the number of database objects that are used to store the dimension data. For example, if the dimension data is stored in three database tables, perform Step 2 thrice.

4. Bind each attribute in the dimension to the database column that stores its data.

After you define a dimension and perform binding (for ROLAP dimensions only), you must deploy the dimension and its associated objects. For more information on deploying dimensions, see "Deploying Dimensions" on page 13-7.

Creating Slowly Changing Dimensions Using the Data Object Editor

You can create an SCD either using the Dimension wizard or the Data Object Editor. Creating SCD type 2 or 3 requires the Warehouse Builder Enterprise ETL Option.

To create an SCD using the Dimension Wizard, you use the Slowly Changing Dimension page of the Dimension Wizard. You need to only specify the type of SCD that you want to create on this page. Warehouse Builder assumes default values for all other required parameters. For more information on the Slowly Changing Dimension page, see "Slowly Changing Dimension Page" on page 13-6.

To create SCDs using the Data Object Editor, see the following sections:

- Creating a Type 2 SCD on page 13-15
- Creating a Type 3 SCD on page 13-17

Creating a Type 2 SCD

A Type 2 SCD stores the full history of values for each attribute and level relationship. You can create a Type 2 SCD only for dimensions that have a relational implementation.

To create a Type 2 SCD using the Data Object Editor, define the following:

- The attributes that trigger history saving.
- The attributes that store the effective date and the expiration date.

To create a Type 2 SCD using the Data Object Editor:

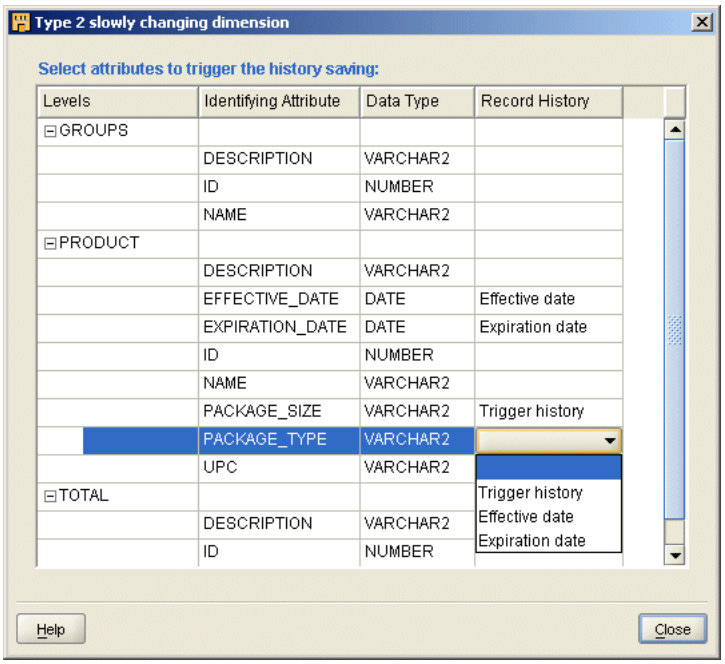
1. From the Warehouse Builder Project Explorer, expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the Type 2 SCD.
3. Right-click **Dimensions,** select **New**, then **Using Editor**
4. Provide information on the Name tab of the Data Object Editor as described in the "Name Tab" on page 13-10.
5. On the Attributes tab, for each level, create two additional attributes to store the effective date and the expiration date. For more information on creating attributes, see "Attributes Tab" on page 13-11.
6. Provide information on the following tabs of the Data Object Editor:
   - Levels Tab on page 13-11
   - Hierarchies Tab on page 13-12
7. On the Slowly Changing tab, select the **Type 2: Store the complete change history** option.
8. Click **Settings** to the right of this option.

   Warehouse Builder displays the Type 2 Slowly Changing Policy dialog. Specify the details of the Type 2 SCD as described in "Type 2 Slowly Changing Dimension Dialog" on page 13-16.
9. Provide information on the Storage Tab of the Data Object Editor.

Type 2 Slowly Changing Dimension Dialog

You use the Type 2 Slowly Changing Dimension dialog shown in Figure 13–3 to specify the effective date attribute, expiration date attribute, and the versioned attribute. This dialog displays a table that contains the following columns: Levels, Identifying Attribute, Data Type, and Record History.

Figure 13–3 Type 2 Slowly Changing Dialog



- **Levels:** Represents the levels in the dimension. Expand a level node to view its level attributes.
- **Identifying Attribute:** Represents the level attribute.
- **Data Type:** Represents the data type of the level attribute.
- **Record History:** Use this drop-down list to indicate that an attribute is versioned or that it stores the effective date or expiration date of the level record.
  - **Trigger History:** Select this option for an attribute if the attribute should be versioned.
  - **Effective Date:** Select this option for an attribute if it stores the value of the effective date of the level record.
  - **Expiration Date:** Select this option for an attribute id it stores the expiration date of the level record.

  The surrogate ID and the business ID of a level cannot be versioned.

For example, in Figure 13–3 the attributes that store the effective date and expiration date are EFFECTIVE_TIME and EXPIRATION_TIME respectively. Note that you must create these dimension attributes and apply them to the Product level. The attribute PACKAGE_TYPE should be versioned. Thus, for this attribute, you select the Record History option for the **Trigger History** property. When the value of the PACKAGE_TYPE attribute changes, the existing record is closed and a new record is created using the latest values.

Creating a Type 3 SCD

A Type 3 SCD stores two versions of values for certain selected attributes. You can create a Type 3 SCD only for dimensions that have a relational implementation. Specify the following:

- The attributes that should be versioned.
- The attributes that will store the previous value of each versioned attribute.

  For each versioned attribute, you must create an additional attribute to store the previous value of the attribute. For example, if you want to version the Population attribute, you create an additional attribute to store the previous value of population.

To create a Type 3 SCD using Warehouse Builder:

1. From the Warehouse Builder Project Explorer, expand the Database node and then the Oracle node.
2. Expand the target module where you want to create the Type 3 SCD.
3. Right-click **Dimensions**, select **New**, then using **Using Editor**.
4. Provide information on the Name tab of the Data Object Editor as described in "Name Tab" on page 13-10.
5. On the Attributes tab, for each level, create an additional attribute to store the expiration date of the attributes in the level as described in "Attributes Tab" on page 13-11.

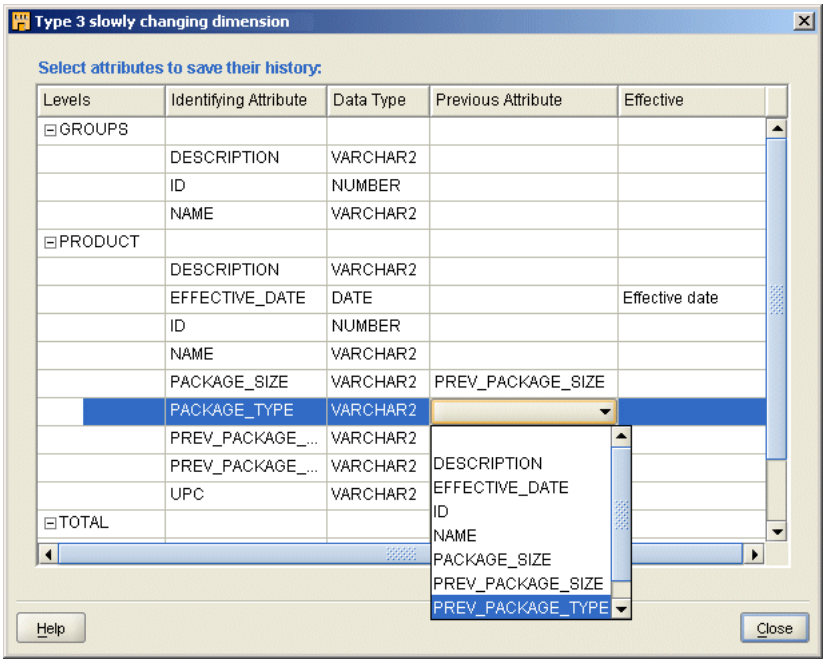
   Consider an example where you need to store previous values for the package_type and package_size attributes of the Products dimension. In this case, create two new attributes prev_package_type and prev_package_size to store the previous values of these attributes.
6. Provide information on the following tabs of the Data Object Editor:
   - Levels Tab on page 13-11
   - Hierarchies Tab on page 13-12
7. On the Slowly Changing tab, select the **Type 3: Store only the previous value** option. Click **Settings** to the right of this option.

   Warehouse Builder displays the Type 3 Slowly Changing Policy dialog. Specify the details of the Type 2 SCD using this dialog as described in "Type 3 Slowly Changing Dimension Dialog" on page 13-17.
8. Provide information on the Storage Tab of the Data Object Editor.

Type 3 Slowly Changing Dimension Dialog

Use the Type 3 Slowly Changing Dimension dialog shown in Figure 13–4 to specify the implementation details. You use this dialog to select the attribute that stores effective date, the attributes that should be versioned, and the attributes that store the previous value of the versioned attributes.

Figure 13–4 Type 3 Slowly Changing Dialog



This dialog displays a table that contains four columns: Levels, Identifying Attribute, Previous Attribute, and Record History.

- **Levels:** Displays the levels in the dimension. Expand a level node to view the level attributes.
- **Identifying Attribute:** Represents the level attribute.
- **Previous Attribute:** Represents the attribute that stores the previous value of the versioned attribute. You use the drop-down list to select the previous value attribute. Specify a previous value attribute only for versioned attributes. You must explicitly create the attributes that store the previous values of versioned attributes. Again, create these as dimension attributes and apply them to the required level.
- **Effective:** Indicates if an attribute stores the effective date. If the attribute stores the effective date, select **Effective date** from the **Effective** drop-down list.

  The surrogate ID of a level cannot be versioned.

Consider the Products Type 3 SCD whose slowly changing policy settings are shown in Figure 13–4. The EFFECTIVE_TIME attribute stores the effective date of the Product level records. The PACKAGE_TYPE attribute of the Product level should be versioned. The attribute that stores the previous value of this attribute, represented by the **Previous Attribute** property, is PREVIOUS_PACKAGE_TYPE. When the value of the PACKAGE_TYPE attribute changes, Warehouse Builder does the following:

- Moves the existing value of the PACKAGE_TYPE attribute the PREVIOUS_ PACKAGE_TYPE attribute.
- Stores the new value of population in the PACKAGE_TYPE attribute.

Editing Dimension Definitions

You use the Data Object Editor to edit the definition of a dimension. When you edit a dimension definition, the changes are made only in the object metadata. To update the physical object definition, deploy the modified dimension using the control Center.

You cannot modify the type of SCD using the Data Object Editor once the SCD has been created.

To edit a dimension definition, open the Data Object Editor using one of the following methods:

- Right-click the dimension in the Project Explorer and select **Open Editor**.
- Double-click the dimension in the Project Explorer.

Modify the definition of the dimension using the tabs in the Data Object Editor. For more information on the tabs in the Data Object Editor, see "Using the Data Object Editor" on page 13-9.

Configuring Dimensions

When you configure a dimension, you configure both the dimension and the underlying table.

To configure the physical properties for a dimension:

1. From the Project Explorer, right-click the dimension name and select **Configure**.

   The Configuration Properties window is displayed.

2. Configure the following dimension properties:

   Deployable: Select TRUE to indicate if you want to deploy this dimension. Warehouse Builder generates scripts only for table constraints marked deployable.

   Deployment Options: Select one of the following options: Deploy All, Deploy Data Objects Only, Deploy to Catalog Only. For more information on deployment options, see "Deployment Options for Dimensions" on page 13-19.

   View Name: Specify the name of the view that is created to hide the control rows in the implementation table that stores the dimension data. This is applicable for relational or ROLAP dimensions that use a star schema. The default view name, if you do not explicitly specify one, is the dimension name suffixed with "_v".

For a dimension that uses a relational or ROLAP implementation, you can also configure the implementation tables. For more information, see "Configuring Tables" on page 12-34.

Deployment Options for Dimensions

Use the **Deployment Option** configuration property to specify how the dimension metadata is deployed. The options that you can select for a dimension are as follows:

Following are the options that you can set for this property.

- Deploy All
- Deploy Data Objects Only
- Deploy to Catalog Only

For more information on these options, see "Deploying Dimensional Objects" on page 4-29.

Deployment Options for Different Dimension Implementations

Depending on the type of implementation you require, set the following configuration options:

Relational dimension: Select **Deploy Data Objects** as the Deployment Option. This creates the dimension metadata in a relational schema in the database.

ROLAP and MOLAP dimensions: Select **Deploy to Catalog Only** or **Deploy All** as the Deployment Option.

Creating Cubes

Warehouse Builder provides the following two methods of creating a cube:

- Using the Create Cube Wizard

  Use the Create Cube wizard to create a basic cube quickly. Warehouse Builder assumes default values for most of the parameters and creates the database structures that store the cube data.

- Using the Data Object Editor

  Use the Data Object Editor to create a cube when you want to specify certain advanced options such as aggregation methods and solve dependency order. These options are not available when you use the Create Cube wizard.

Alternatively, you can use the Create Cube wizard to quickly create a basic cube object. Then use the Data Object Editor to specify the other options.

Using the Create Cube Wizard

Use the following steps to create a cube using the wizard:

1. From the Warehouse Builder Project Explorer expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the cube.
3. Right-click **Cubes,** select **New,** then **Using Wizard**.

   Warehouse Builder displays the Welcome page of the Cube wizard. Click **Next** to proceed. The wizard guides you through the following pages:

   - Name and Description Page on page 13-20
   - Storage Type Page on page 13-21
   - Dimensions Page on page 13-21
   - Measures Page on page 13-22
   - Summary Page on page 13-22

Name and Description Page

You use the Name and Description page to describe the cube. Enter the following details on this page:

- **Name:** The name of the cube in Warehouse Builder. The cube name must be unique within the module.
- **Description:** Specify an optional description for the cube.

Storage Type Page

Use the Storage Type page to specify the type of storage for the cube. The storage type determines how the cube data is physically stored in the database. The options you can select for storage type are as follows:

- Relational storage (ROLAP)
- Multidimensional storage (MOLAP)

You select the storage type based on the volume of data stored at the lowest level of the entire cube and the refresh rate required.

Relational storage (ROLAP)

Warehouse Builder stores the cube definition and its data in a relational form in the database. Use this option to create a cube that has a relational or ROLAP implementation.

Relational storage is preferable if you want to store detailed, high volume data or you have high refresh rates combined with high volumes of data. Use relational storage if you want to perform one of the following:

- Store detailed information such as call detail records, point of sales (POS) records and other such transaction oriented data.
- Refresh high volumes of data at short intervals.
- Detailed reporting such as lists of order details.

Operational data stores and enterprise data warehouses are typically implemented using relational storage. You can then derive MOLAP implementations from this relational implementation to perform different types of analysis.

When you choose a relational implementation for a cube, Warehouse Builder automatically creates the implementation table used to store the cube data.

Multidimensional storage (MOLAP)

Warehouse Builder stores the cube definition and the cube data in an analytic workspace in the database. Use this option to create a cube that has a MOLAP implementation.

Multidimensional storage is preferable when you want to store aggregated data for analysis. The refresh intervals for a multidimensional storage are usually longer than relational storage as data needs to be pre-calculated and pre-aggregated. Also, the data volumes are typically smaller due to higher aggregation levels. Use multidimensional storage to perform the following:

- Advanced analysis such as trend analysis, what-if analysis, or to forecast and allocate data
- Drill and pivot data with instant results

When you choose a MOLAP implementation, Warehouse Builder generates the name used to store the cube in the analytic workspace. If no analytic workspace exists, Warehouse Builder creates one using the name you specify.

Dimensions Page

The Dimensions page defines the dimensionality of the cube. A cube must refer to at least one dimension. You define dimensionality by selecting the dimensions that the cube references. The Search For option enables you to search for a dimension using the

dimension name. You can use the same dimension to define multiple cubes. For example, the dimension TIMES can be used by the SALES cube and the COST cube.

The Dimensions page contains two sections: Available Dimensions and Selected Dimensions.

Available Dimensions The Available Dimensions section lists all the dimensions in the Warehouse Builder repository. Each module in the project is represented by a separate node. Expand a module node to view all the dimensions in that module.

Warehouse Builder filters the dimensions displayed in the Available Dimensions section based on the implementation type chosen for the dimension. If you select ROLAP as the storage type, Warehouse Builder lists only dimensions that have a relational implementation. If you select MOLAP as the storage type, only dimensions stored in an analytic workspace are listed.

Selected Dimensions The Selected Dimensions section lists the dimensions that you selected in the Available Dimensions section. Use the right arrow to the move a dimension from the Available Dimensions list to the Selected Dimensions list.

Measures Page

Use the Measures page to define the measures of the cube. For each measure, specify the following details:

- **Name:** The name of the measure. The name of the measure must be unique within the cube.
- **Description:** An optional description for the measure.
- **Data Type:** Select the data type of the measure.

> **Note:** The following data types are not supported for MOLAP implementations: BLOB, Interval Day to Second, Interval Year to Month, RAW, Timestamp with Time Zone, Timestamp with Local Time Zone.

- **Length**: Specify length only for character data types.
- **Precision**: Define the total number of digits allowed for the measure. Precision is defined only for numeric data types.
- **Scale:** Define the total number of digits to the right of the decimal point. Scale is defined only for numeric data types.
- **Seconds Precision:** Represents the number of digits in the fractional part of the datetime field. It can be a number between 0 and 9. The seconds precision is used only for TIMESTAMP data types.
- **Expression:** Use this option to define calculated measures. Click the Ellipsis button to display the Expression Builder. Define the expression used to calculate the measure.

Summary Page

You use the Summary page to review the options that you specified using the Cube wizard. Click **Finish** to complete defining the cube. This cube is displayed under the Cubes node of the Project Explorer.

Warehouse Builder creates the metadata for the following in the repository:

- The cube object.
- The definition of the table that stores the cube data.

  For a relational or ROLAP implementation, Warehouse Builder creates the definition of the database table that stores the cube data. It also creates foreign keys in the table that stores the cube data to each data object that stores the data relating to the dimension the cube references.

  For a MOLAP implementation, Warehouse Builder creates the analytic workspace that stores the cube data. Note that the wizard only creates the definitions for these objects in the repository. It does not create the objects in the target schema.

Deploying Cubes To create the cube and its associated objects in the target schema, you must deploy the cube. Before you deploy a ROLAP cube, ensure that you successfully deploy the database table that stores the cube data. Alternatively, you can deploy both the table and the cube together. For more information, see "MOLAP Implementation of Dimensional Objects" on page 4-28.

Note: When you delete a cube, the associated objects such as the database table or analytic workspace is not deleted. You must explicitly delete these objects.

Defaults Used by the Create Cube Wizard

When you create a cube using the Create Cube wizard, Warehouse Builder uses the following defaults:

- **MOLAP Storage:** The cube is stored in an analytic workspace that has the same name as the Oracle module in which the cube is created. The analytic workspace is stored in the users tablespace of the schema that owns the Oracle module.
- **Solve:** By default, the cube is solved on demand.
- **Aggregation Function:** The default aggregation function for all dimensions that the cube references is SUM.

Using the Data Object Editor

The Data Object Editor enables advanced users to create cubes according to their requirements. You can also use the Data Object Editor to edit a cube.

Use the Data Object Editor to create a cube if you need to:

- Specify the dimensions along which the cube is sparse.
- Define aggregation methods for the cube measures.
- Precompute aggregations for a level.

To create a cube using the Data Object Editor:

1. From the Warehouse Builder Project Explorer expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the cube.
3. Right-click **Cubes**, select **New** and then **Using Editor**.

   Warehouse Builder displays the Add a New or Existing Cube dialog.
4. Select the **Create a New Cube** option. Enter the name of the cube and select the module to which the cube belongs.

5. Click **OK**.

   Warehouse Builder displays the Data Object Editor. To define a cube, provide information on the following tabs of the Cube Details panel:

   - Name Tab on page 13-24
   - Storage Tab on page 13-24
   - Dimensions Tab on page 13-25
   - Measures Tab on page 13-26
   - Aggregation Tab on page 13-30
   - Data Viewer Tab on page 13-31

   When you use the Data Object Editor to create a cube, Warehouse Builder does not automatically create the physical objects that stores the cube data. You will need to create these objects.

6. To bind the cube measures and the dimension references to the database columns that store their data, see "Binding Cube Attributes" on page 13-31. You need to perform this step only for cubes that use a ROLAP implementation.

Name Tab

You use the Name tab to describe the cube. Specify the following details on this tab:

- **Name**: Specify a name for the cube. The cube name must be unique within the module.
- **Description**: Specify an optional description for the cube.

Storage Tab

The Storage tab specifies how the cube and its data should be stored. You can select either Relational or MOLAP as the storage type.

Relational Select the **Relational: Relational data structures** option to store the cube definition and its data in a relational form in the database. Use this option to create a cube that has a relational or ROLAP implementation. The cube data is stored in a database table or view.

Select the **Create bitmap indexes** option to generate bitmap indexes on all the foreign key columns in the fact table. This is required for a star query. For more information, refer to the *Oracle Database Data Warehousing Guide 10g Release 2*.

Select the **Create composite unique key** option to create a unique key on the dimension foreign key columns.

MOLAP Select the **MOLAP: Multidimensional data structures** option to store the cube data in an analytic workspace. Use this option to create a cube with a MOLAP implementation. You use the Analytic Workspace section to specify the storage details. Enter the following details in this section:

- **AW Name:** This field specifies the name of the analytic workspace that stores the cube definition and cube data. Use the **Select** button to display the Analytic Workspaces dialog. This dialog lists the dimensional objects in the current project. Selecting an object from list stores the cube in the same analytic workspace as the selected object.
- **AW Tablespace Name:** Represents the name of the tablespace in which the analytic workspace is stored. If you do not specify a name, Warehouse Builder

stores the analytic workspace in the default users tablespace of the owner of the Oracle module.

Dimensions Tab

You use the Dimensions tab to define the dimensionality of the cube. This tab displays a table that you use to select the dimensions that the cube references and the **Advanced** button. You can change the order of the dimensions listed in this tab by using the arrows on the left of this tab.

Use the **Advanced** button to define the sparsity of the dimensions referenced by the cube. Clicking this button displays the Advanced dialog. Since you can define sparcity only for MOLAP cubes, the Advanced button is enabled only if the Storage type is MOLAP. For more information about the Sparcity dialog, see "Advanced Dialog" on page 13-25.

The table on the Dimensions tab contains the following columns:

- **Dimension:** This field represents the name of the dimension that the cube references. Click the Ellipsis button in this field to display the Available Modules dialog. This dialog displays the list of dimensions in the current project. Select a dimension from this list.

  Warehouse Builder filters the dimensions displayed in this list based on the storage type specified for the cube. If you define a relational implementation for the cube, Warehouse Builder displays only those dimensions that use a relational implementation. If you define a MOLAP implementation for the cube, Warehouse Builder displays only the dimensions that use a MOLAP implementation.

- **Level:** The Levels drop-down displays all the levels in the dimension selected in the Dimension field. Select the dimension level that the cube references.

- **Role:** The Role drop-down list displays the dimension roles, if any, that the selected dimension contains. Select the dimension role that the cube uses. You can specify dimension roles for relational dimensions only.

Advanced Dialog Use the Advanced dialog to specify the sparsity of the dimensions that the cube references. Sparsity is applicable for MOLAP cubes only. For more information on sparsity, refer to Chapter 3 of the *Oracle OLAP Application Developer's Guide 10g Release 2(10.2)*.

This dialog displays a table that contains two columns: Dimensions and Sparsity.

- **Dimensions:** This column displays all the dimensions listed on the Dimension tab of the Data Object Editor. The dimensions are listed in the order in which they appear on the Dimensions tab. To change the order in which the dimensions appear on this dialog, you must change the order in which the dimensions are listed on the Dimensions Tab of the Data Object Editor.

- **Sparsity:** Select **Sparsity** for a dimension reference if the cube data is space along that dimension. Sparsity specifies that the cube data is sparse along a particular dimension. For example, if the data in the SALES cube is sparse along the Promotions dimension, select Sparsity for the Promotions dimension.

  All the sparse dimensions in a cube must be grouped together starting from the least sparse to the most sparse. For example, the SALES cube references the dimensions Times, Promotions, Products, and Channels. This is the order in which the dimensions are listed in the Advanced dialog. The cube data is sparse along the dimensions Promotions, Channels, and Times, with Promotions being the least sparse and Times being the most sparse. Then all these dimensions should appear as a group in the following order: Promotions, Channels, Times. You cannot have

any other dimension listed in between these dimensions. Thus the order listed originally on the Advanced dialog is wrong because the Products dimension appears in between the dimensions Promotions and Channels. The following order of listing the dimensions are correct:

- Promotions, Channels, Times, Products
- Products, Promotions, Channels, Times

There is a correlation between the storage option used for the cube and the order in which you specify the dimensions on the Sparcity dialog. When the cube is not stored in compressed form (that is, when the Compress Cube option on the Advanced dialog is not selected), list the sparse dimensions in order from the one with the most members to the one with the least. When the cube is stored in compressed form, the order in which you list the dimensions is unimportant. This is because the OLAP engine determines the optimal order. When the cube uses a normal storage (not compressed), list the dimensions such that the least sparse dimension appears at the top of the list.

Defining sparsity for a cube provides the following benefits:

- Improves data retrieval speed.
- Reduces the storage space used by the cube.

Compress Cube Select this option to compress the cube data and then store it. Compressed storage uses less space and results in faster aggregation than a normal space storage. For more details on compressing cubes, refer to the *Oracle OLAP Application Developer's Guide 10g Release 2(10.2)*

Compressed storage is normally used for extremely sparse cubes. A cube is said to be extremely sparse if the dimension hierarchies contain levels with little change to the number of dimension members from one level to the next. Thus many parents have only one descendent for several contiguous levels. Since the aggregated data values do not change from one level to the next, the aggregate data can be stored once instead of repeatedly.

For compressed composites, you can only choose SUM and non-additive aggregation operators.

Partition Cube Select this option to partition the cube along one of its dimensions. Partitioning a cube improves the performance of large measures.

Use the table below the Partition Cube option to specify the dimension along which the cube is partitioned. The specified dimension must have at least one level-based hierarchy and its members must be distributed evenly, such that every parent at a particular level has roughly the same number of children. Use the **Dimension** column to select the dimension along which the cube is partitioned. Use the **Hierarchy** and **Level** columns to select the dimension hierarchy and level.

Time is usually the best choice to partition a cube because it meets the required criteria. In addition, data is loaded and rolled off by time period, so that new partitions can be created and old partitions dropped as part of the data refresh process.

Use a Global Index Select this option to create a global partitioned index.

Measures Tab

You use the Measures tab to define the cube measures. Specify the following details for each measure:

- **Name:** The name of the measure. The measure name must be unique within the cube.
- **Description:** An optional description for the measure.
- **Data Type:** The data type of the measure.
- **Length:** The maximum number of bytes for the measure. Length is specified only for character data.
- **Precision:** Define the total number of digits allowed for the column. Precision is defined only for numeric data types.
- **Scale:** Define the total number of digits to the right of the decimal point. Scale is defined only for numeric data types.
- **Seconds Precision:** Represents the number of digits in the fractional part of the datetime field. It can be a number between 0 and 9. The seconds precision is used only for TIMESTAMP data types.
- **Expression:** Use this field to define a calculated measure. A calculated measure is a measure whose data is not stored. Its value is calculated when required using the expression defined. Click the Ellipsis button to display the Calculated Measure wizard. For more information on the Calculated Measure wizard, see "Calculated Measure Wizard" on page 13-27.

  You can use any other measures defined in the cube to create an expression for a measure. The expression defined can be validated only at deploy time.

  Note: You can create calculated measures for MOLAP dimensions only.

Click the **Generate Standard Measures** button to generate a series of standard calculations for a base measure. This is a time-saver operation for creating share, rank and time based calculations. Any calculated measure that you create using this option can also be created manually using the Calculated Measure wizard.

Calculated Measure Wizard

Use the Calculated Measure wizard to create calculated measures in a cube that uses a MOLAP implementation. These calculated measures, just like the other measures defined on the cube, are deployed to an analytic workspace. The wizard enables you create certain extra calculations that are not created when you click Generate Calculated Measures.

Select Calculated Measure Type Use this page to define the type of calculated measure. You can create the following types of calculated measures:

- Standard Calculation
- Custom Expression

Standard Calculation Select the Standard Calculation option to define standard calculations based on the templates. Warehouse Builder enables you to define the following standard calculations:

- **Basic Arithmetic:** This type enables you to perform basic arithmetic calculations such as the following:
  - Addition

Use this calculation to add either two measures or a measure and a number.

- Subtraction

  Use this calculation to subtract two measures or a measure and a number.

- Multiplication

  Use this calculation to multiply two measures or a measure and a number.

- Division

  Use this calculation to divide two measures or a measure and a number.

- Ratio

- **Advanced Arithmetic:** This type enables you to create the following advanced calculations:

  - Cumulative Total

    Use this calculation to return the cumulative total of measure data over time periods within each level of a specified dimension.

    For example, Cumulative Sales for 2001= Sales Q1 + Sales Q2 + Sales Q3 + Sales Q4

  - Index

    Use this calculation to return the ratio of a measure's value as a percentage of a baseline value for the measure. The formula for the calculation is:

    (Current member / Base Line member)

    For example, Consumer Price Index (assuming baseline cost of goods is 1967) = (2001 Cost of Goods/1967 Cost of Goods)) * 100

  - Percent Markup

    Use this calculation to return the percentage markup between two measures where the basis for the calculation is the older measure. the formula used for this calculation is: (y-x)/x.

    For example, the new price is 110 and the old price is 100. The percentage markup is calculated as follows:

    (110-100)/100 = +10%.

  - Percent Variance

    Use this calculation to return the percent difference between a measure and a target for that measure. For example, the percentage variance between sales and quota is as follows:

    (Sales-Quota)/Quota.

  - Rank

    Use this calculation to return the numeric rank value of each dimension member based on the value of the specified measure. For example, the rank of TV sales where DVD is 150, TV is 100, and Radio is 50 would be 2.

  - Share

    Use this calculation to return the ratio of a measure's value to the same measure value for another dimension member or level. The formula for this calculation is as follows:

    (Current member / Specified member).

  - Variance

    Use this calculation to calculate the variance between a base measure and a target for that measure. An example of variance is:

    Sales Variance = Sales - Sales Forecast.

- **Prior/Future Comparison:** Use this type to define prior and future value calculations such as the following:
  - Prior Value

    Use this calculation to return the value of a measure from an earlier time period.

  - Difference from Prior Period

    Use this calculation to return the difference between the current value of a measure and the value of that measure from a prior period. The formula for this calculation is as follows:

    (Current Value - Previous Value)

  - Percent Difference from Prior Period

    Use this calculation to return the percentage difference between the current value of a measure and the value of that measure from a prior period. The formula for this calculation is as follows:

    ((Current Value - Previous Value) / Previous Value)

  - Future Value

    Use this calculation to return the value of an item for a future time period. For example, Sales a Year from Now = Sales from October 2006 if the current time is October 2005.

- **Time Frame:** This type enables you to create the following time series calculations:
  - Moving Average

    Use this calculation to return the average value for a measure over a rolling number of time periods. And example of this calculation is as follows:

    Moving average sales for the last 3 months = (Jan Sales + Feb Sales + March Sales)/3

  - Moving Maximum

    Use this calculation to return the maximum value for a measure over a rolling number of time periods. An example of this calculation is:

    Moving maximum sales for the last 3 months = the largest Sales value for Jan, Feb, and March.

  - Moving Minimum

    Use this calculation to return the minimum value for a measure over a rolling number of time periods. An example of this calculation is:

    Moving minimum sales for the last 3 months = the smallest Sales value for Jan, Feb, and March.

  - Moving Total

    Use this calculation to return the total value for a measure over a rolling number of time periods. An example of this calculation is:

Moving total sales for the last 3 months = (Jan Sales + Feb Sales + March Sales).

– Year to Date

  Use this calculation to sum measure data over time periods, to create cumulative measure data. An example of this calculation is:

  Year-to-date sales to March = Jan Sales + Feb Sales + March Sales.

Custom Expression Select the **Custom Expression** option to specify an expression that is used to compute the calculated measure.

Define Calculated Measure Details Use this page to define the details of the calculated measure. The contents of this page depend on the type of calculation you chose on the Select Calculated Measure Type page. For example, if you choose addition as the calculated measure type, this page displays the two drop-down lists that enable you to select the measure s that should be added.

If you chose Custom Expression on the Select Calculated Measure Type page, the Expression Builder interface is displayed. Use this interface to define a custom measure. For more information on the Expression Builder, see "The Expression Builder" on page 26-3.

Reviewing the Summary Information Use the Finish page to review the information defined for the calculated measure. Click **Back** to change any of the selected values. Click **Finish** to complete the calculated measure definition.

Aggregation Tab

Use the Aggregation tab to define the aggregations that must be performed for each dimension that the cube references. You select the aggregate function that is used to aggregate data. You can also precompute measures along each dimension that the cube references. By default, Warehouse Builder performs aggregation for every alternate level starting from the lowest level. The default aggregate function is SUM. For more details on the rules for summarizing data, refer to Chapter 3 of the *Oracle OLAP Application Developer's Guide 10g Release 2(10.2)*.

Specify the following options on the Aggregations tab:

- **Cube Aggregation Method:** Select the aggregate function used to aggregate the cube data. The default selection is SUM.
- **Summary Refresh Method:** Select the data refresh method. The options you can select are On Demand and On Commit.

Summary Strategy for Cube Use this section to define levels along which data should be precomputed for each dimension. The **Dimension** column lists the dimensions that the cube references. To select the levels in a dimension for which data should be precomputed, click the Ellipsis button in the **PreCompute** column to the right of the dimension name. The PreCompute dialog is displayed. Use this dialog to select the levels in the dimension along which the measure data is precomputed. You can specify the levels to be precomputed for each dimension hierarchy. By default, alternate levels, starting from the lowest level, are precomputed.

Note: You cannot define aggregations for pure relational cubes (cubes implemented in a relational schema in the database only and not in OLAP catalog).

Precomputing ROLAP Cubes For ROLAP cubes, Warehouse Builder implements aggregation by creating materialized views that store aggregated data. These materialized views improve query performance. For MOLAP implementations, the aggregate data is generated and stored in the analytic workspace along with the base-level data. Some of the aggregate data is generated during deployment and the rest is aggregated on the fly in response to a query, following the rules defined in the Aggregation tab.

> **Note:** The materialized views that Warehouse Builder creates to implement ROLAP aggregation are not displayed under the Materialized Views node in the Project Explorer.

Data Viewer Tab

You use the Data Viewer tab to view cube data. This tab contains the following three buttons: Execute, Query Builder, and Calculation Builder.

Execute Click the **Execute** button to display the data viewer for the cube. The data viewer displays the cube data, 100 rows at a time. To view the remaining rows click **Next**.

Query Builder Click the **Query Builder** button to launch the query builder. You use the query builder to customize the query that fetches data from the cube. For example, you can swap edges or select dimension members.

Calculation Builder The **Calculation Builder** is used to create a calculation that is based on the existing cube measures. Once you create a calculation, you can use it just like any other measure.

Binding Cube Attributes

After you define the cube structure, you must specify the details of the database tables or views that store the cube data. You can choose one of the following options to bind the cube to the database object that stores its data:

- Auto binding
- Manual binding

Auto Binding When you perform auto binding, Warehouse Builder automatically maps the measures and dimension references of the cube to the database columns that store their data.

To perform auto binding:

1. In the Data Object Editor, select **Object** and then **Auto Binding**. You can alternately right-click the cube object in the Implementation tab of the graphical canvas and select Auto Binding.

   Warehouse Builder displays the Auto binding dialog.

2. Select **Yes** to perform auto binding.

   Warehouse Builder maps the measures and dimension references in the cube to the table that stores the cube data.

For more information on the auto binding rules, see "Auto Binding" on page 4-27.

Manual Binding In manual binding, you must explicitly map the measures and dimension references in the cube to the database objects that store their data. You can either store the cube data in existing tables or create new tables.

To perform manual binding:

1. In the Data Object Editor, right-click the cube object in the Dimensional tab of the Graphical canvas and select **Detail View**.

   Warehouse Builder displays the Implementation tab. This tab contains the cube object.

2. Drag and drop the mapping operator that represents the database object that will store the cube data. For example, if the cube data will be stored in a table, drag and drop the Add New or Existing Table icon onto the Implementation canvas.

   Warehouse Builder displays the Add a new or Existing table dialog. You either select an existing table or create a new table to store the cube data.

3. Map each attribute in the dimension to the database column that stores its data.

After you define the cube using the Data Object and perform binding (for ROLAP cubes only), you must deploy the cube. For more information on deploying cubes, see "Deploying Cubes" on page 13-23.

Cubes Stored in Analytic Workspaces

Cubes that use a MOLAP implementation are stored in analytic workspaces. The analytic workspace engine in Oracle Database 10g provides APIs called AWXML. These APIs enable both client-server usage (as in Analytic Workspace Manager) and batch-like usage with java stored procedures (like Oracle Warehouse Builder). This section describes implementation details for MOLAP cubes.

Ragged Cube Data in Warehouse Builder

If you select **Use natural keys from data source** on the Storage tab of a dimension, Warehouse Builder generates mapping code (AWXML mapping code) that can handle ragged fact data for any cube that uses this dimension. The source column for the cube dimension level is actually mapped to every parent level also. This enables ragged fact data to be loaded.

If you select **Generate surrogate keys in the analytic workspace** on the Storage tab of a dimension, when you create a mapping that loads data at the level of this dimension, you will be loading cube dimension members for this level only.

Defining Aggregations

Warehouse Builder enables you to reuse existing dimensions without the need of defining additional hierarchies. Aggregations are generated based on the cube dimension level references you define. Only hierarchies where the cube dimension level is a member will be included in the aggregation. If the cube dimension level referenced is a non-leaf level of the hierarchy, then levels lower in the hierarchy will be excluded when the cube or measures are solved. For example, if you have two cubes, BUDGET and SALES, they can share the same dimension definitions without additional dimension hierarchy definitions.

Auto Solving MOLAP Cubes

An important attribute of the OLAP AWXML engine is its ability to auto-solve cubes that are stored in analytic workspaces. You can auto-solve both compressed and

non-compressed cubes. A compressed cube is one for which the **Compress Cube** option on the Advanced Dialog is selected.

A cube is auto-solved if any of the following conditions are satisfied:

- The cube is compressed
- The cube is not compressed, and the following additional conditions are true:
  - The solve property for all the measures is set to Yes.
  - The dimension levels that the cube references are at the leaf level of all hierarchies the level is a member of.
- Mapping that contains the cube is executed

Incremental Aggregation of cube is dependent on auto-solve (load and aggregate in one operation). Incremental aggregation is a property of the cube operator in the mapping and applies only to auto-solved cubes.

Warehouse Builder can generate cubes that are not auto-solved cubes if any of the following conditions are true:

- The cube is solved by the mapping that loads the cube
- Warehouse Builder transformations are used to solve the cube
- The cube is non-compressed and any of the following conditions are true:
  - Some of the measures have the Solve property set to No.
  - The dimension levels that the cube references are non-leaf levels of a hierarchy the level is a member of.

Solving Cube Measures

You can choose to solve only one cube measure for both compressed and non-compressed cubes. A compressed cube is one for which the Compress Cube option on the Advanced Dialog is selected.

To solve only one measure in a compressed cube, use the following steps:

1. Open the Data Object Editor for the cube and navigate to the Aggregation tab.

   You can open the Data Object Editor by double-clicking the cube name in the Project Explorer.

2. Select the measure that you want to solve on the **Measures** section of the Aggregation tab.

3. The **Aggregation for measure** section displays a row for each dimension that the cube references. In the row that represents the dimension along which you want to solve the cube, select **NOAGG** in the **Aggregation Function** column.

To solve only one measure in a non-compressed cube, you will need the latest database patch 10.2.0.2. If you have Oracle Database 10g Release 1 (10.1), refer to bug 4550247 for details about a patch. The options defined on cube measures for solve indicate which measures will be included in the primary solve. The solve indicator on the cube operator in the map however indicates whether this solve will be executed or not. So the map can just load data or load and solve the data.

Solving Cubes Independent of Loading

You can solve cubes independent of loading using the predefined transformation WB_OLAP_AW_PRECOMPUTE. This function also enables you to solve measures independently of each other. This transformation function is available in the Global

Explorer under the Public Transformations node in the OLAP category of the Predefined node.

The following example solves the measure SALES in the SALES_CUBE:

```
declare
    rslt VARCHAR2(4000);
begin
    rslt:=WB_OLAP_AW_PRECOMPUTE('MART','SALES_CUBE','SALES');
end;
/
```

This function contains parameters for parallel solve and maximum number of job queues. If the cube is being solved in parallel, an asynchronous solve job is launched and the master job ID is returned through the return value of the function.

Calculation Plans Generated by Warehouse Builder The following calculation plans are generated by Warehouse Builder:

- Calculation plan for the cube
- Calculation plan for each stored measure

This allows measures to be solved individually after a data load, or entire cubes to be solved. The actual calculation plan can also exclude levels based on Warehouse Builder metadata.

Parallel Solving of Cubes

You can enable parallel solving of cubes using by configuring the mapping that loads the cube. The cube operator has a property called **Allow Parallel Solve** and also a property for the **Max Job Queues Allocated**. These two configuration properties determine if parallel solving is performed and also the size of the job pool. The default is to let the AWXML engine determine this value.

Output of a MOLAP Cube Mapping

When you execute a mapping that loads a cube, one of the output parameters is AW_EXECUTE_RESULT. When the map is executed using parallel solve, this output parameter will contain the job ID. You can then use the following data dictionary views to determine when the job is complete and what to do next:

- ALL_SCHEDULER_JOBS
- ALL_SCHEDULER_JOB_RUN_DETAILS
- ALL_SCHEDULER_RUNNING_JOBS

If the mapping is not executed using parallel solve, the AW_EXECUTE_RESULT output parameter will return the 'Successful' tag or an error. For more details about the error, refer to the OLAPSYS.XML_LOAD_LOG table.

Editing Cube Definitions

You can edit a cube and alter its definition using the Data Object Editor. When you edit a dimension definition, the changes are made only in the object metadata. To update the physical object definition, deploy the modified dimension using the Control Center.

To edit a cube definition, open the Data Object Editor using one of the following methods:

- Right-click the cube in the Project Explorer and select **Open Editor**.
- Double-click the cube in the Project Explorer.

You use the tabs of the Data Object Editor to edit the cube definition. For more information on the tabs in the Data Object Editor, see Using the Data Object Editor on page 13-23.

Configuring Cubes

When you configure a cube, you configure both the cube and the underlying table.

To configure the physical properties for a cube:

1. From the Project Explorer, right-click the cube name and select **Configure** from the pop-up menu.

   The Configuration Properties window is displayed.

2. Configure the following cube parameters:

   Deployable: Select TRUE to indicate if you want to deploy this cube. Warehouse Builder generates scripts only for table constraints marked deployable.

   Deployment Options: Use this property to specify where the dimension is deployed. The options are:

   - **Deploy Aggregations:** Deploys the aggregations defined on the cube measures.
   - **Deploy All:** For a relational implementation, the cube is deployed to the database and a CWM definition to the OLAP catalog. For a MOLAP implementation, the cube is deployed to the analytic workspace.
   - **Deploy Data Objects only:** Deploys the cube only to the database. You can select this option only for cubes that have a relational implementation.
   - **Deploy to Catalog only:** Deploys the CWM definition to the OLAP catalog only. Use this option if you want applications such as BI Beans or Discoverer for OLAP to access the dimension data after you deploy data only.

   Materialized View Index Tablespace: The name of the tablespace that stores the materialized view indexes.

   Materialized View Tablespace: The name of the tablespace that stores the materialized view created for the cube.

Although there are no additional configuration parameters for cubes, the following are some guidelines for configuring a cube.

- Foreign Key constraints exist for every dimension.
- Bitmap indexes have been generated for every foreign key column to its referenced dimension.

Creating Time Dimensions

Warehouse Builder provides the Create Time Dimension wizard that enables you to create a fully functional time dimension quickly. The mapping that populates the time dimension is also created automatically. When you choose a relational implementation for a time dimension, Warehouse Builder also creates the implementation objects that store the time dimension data.

You can also use the Data Object Editor to define a time dimension with your own specifications. In this case, you need to create the implementation objects and the map that loads the time dimension.

Creating a Time Dimension Using the Time Dimension Wizard

Use the following steps to create a time dimension using the Create Time Dimension wizard:

1. From the Warehouse Builder Project Explorer expand the **Databases** node and then the **Oracle** node.
2. Expand the target module where you want to create a time dimension.
3. Right-click **Dimensions**, select **New**, then **Using Time Wizard**.

   Warehouse Builder displays the Welcome page of the Create Time Dimension wizard. Click **Next** to proceed. The wizard guides you through the following pages:

   - Name and Description Page on page 13-36
   - Storage Page on page 13-36
   - Data Generation Page on page 13-37
   - Levels Page (Calendar Time Dimension Only) on page 13-37
   - Levels Page (Fiscal Time Dimension Only) on page 13-38
   - Pre Create Settings Page on page 13-38
   - Time Dimension Progress Page on page 13-38
   - Summary Page on page 13-38

Name and Description Page

The Name page describes the time dimension. Provide the following details on the Name page:

- **Name:** Type the name of the time dimension. The name must be unique within a module.
- **Description:** Type an optional description for the time dimension.

Storage Page

Use the Storage page to specify how the time dimension data should be stored in the database. You select the storage type based on the volume of data stored at the lowest level of the entire cube and the refresh rate required. The storage type options are:

- **Relational storage (ROLAP):** Stores the time dimension definition in a relational form in the database. Select this option to create a time dimension that uses a relational or ROLAP implementation.

  Warehouse Builder automatically creates the underlying tables required to implement this time dimension. A star schema is used to implement the time dimension.
- **Multidimensional storage (MOLAP):** Stores the time dimension definition and data in an analytic workspace. Select this option to create a time dimension that uses a MOLAP implementation.

Warehouse Builder stores the time dimension in an analytic workspace with same name as the module. The tablespace that is used to store the analytic workspace is the tablespace that is defined as the users tablespace for the schema that contains the dimension metadata.

For more information on these options, see "Storage Type Page" on page 13-2.

Data Generation Page

Use the Data Generation page to specify additional information about the time dimension such as the type of time dimension and the range of data stored in it. This page contains details about the range of data stored in the time dimension and the type of temporal data.

Range of Data The Range of Data section specifies the range of the temporal data stored in the time dimension. To specify the range, define the following:

- **Start year:** The year from which to store data in the time dimension. Click the drop-down list to select a starting year.
- **Number of years:** The total number of years, beginning from Start Year, for which the time dimension stores data. Specify the number of years by selecting a value from the drop-down list.

Type of Time Dimension Use the Type of Time Dimension section to specify the type of time dimension to create. Select one of the following options for type of time dimension:

- **Calendar:** Creates a calendar time dimension.
- **Fiscal:** Creates a fiscal time dimension. Enter the following additional details to create a fiscal time dimension:
  - **Fiscal Convention:** Select the convention that you want to use to represent the fiscal months. The options available are 544 and 445.
  - **Fiscal Year Starting:** Select the date and month from which the fiscal year starts.
  - **Fiscal Week Starting:** Select the day from which the fiscal week starts.

Levels Page (Calendar Time Dimension Only)

Use the Levels page to select the calendar hierarchy that should be created and the levels that it contains. Since there is no drill-up path from the Calendar Week level to any of the levels above it, Warehouse Builder provides the following two options to create a calendar hierarchy:

- Normal Hierarchy
- Week Hierarchy

Normal Hierarchy The Normal Hierarchy contains the following levels:

- Calendar year
- Calendar quarter
- Calendar month
- Day

Select the levels to be included in the calendar hierarchy. You must select at least two levels.

Week Hierarchy The Week Hierarchy contains two levels: Calendar Week and Day. Use this hierarchy to create a hierarchy that contains the Calendar Week level. When you select the Week Hierarchy option, both these levels are selected by default.

Levels Page (Fiscal Time Dimension Only)

Use the Levels page to select the levels that should be included in the fiscal hierarchy. The levels you can select are:

- Fiscal year
- Fiscal quarter
- Fiscal month
- Fiscal week
- Day

You must select a minimum of two levels. Warehouse Builder creates the fiscal hierarchy that contains the selected levels. To create additional hierarchies, use the Data Object Editor. For more information on using the Data Object Editor, see "Editing Time Dimension Definitions" on page 13-39.

Pre Create Settings Page

The Pre Create Settings page displays a summary of the options you selected on the previous pages of the Create Time Dimension wizard. This includes the attributes, levels, hierarchies, and the name of the map that is used to populate the time dimension. Warehouse Builder uses these settings to create the objects that implement the time dimension. Click **Next** to proceed with the implementation of the wizard. Click **Back** to change any options that you selected on the previous wizard pages.

Time Dimension Progress Page

The Time Dimension Progress page displays the progress of the time dimension implementation. The progress status log on this page lists the activities that are performed by the Time Dimension wizard to implement the time dimension. After the process is completed, click **Next** to proceed.

Summary Page

The Summary page summarizes the options selected in the wizard pages. You use this page to review the options you selected.

Click **Finish** to complete the creation of the time dimension. You now have a fully functional time dimension. This dimension is displayed under the Dimensions node of the Project Explorer. The mapping that loads this time dimension is displayed under the Mappings node in the Project Explorer.

Warehouse Builder creates the following objects:

- The time dimension object.
- The sequence that populates the surrogate ID of the time dimension levels
- The physical structures that store the time dimension data.

  For a relational implementation, the database tables that store the dimension data are created in the repository. Warehouse Builder also binds the time dimension attributes to the database columns that store their values. For a MOLAP implementation, the analytic workspace that stores the time dimension and its data is created.

- A mapping that populates the time dimension.

> **Note:** When you delete a time dimension, Warehouse Builder does not delete the table, sequence, and the mapping associated with the time dimension. You need to explicitly delete these objects.

Defaults Used by the Time Dimension Wizard

When you create a time dimension using the Time Dimension wizard, Warehouse Builder uses the following defaults:

- **Storage:** The default implementation for the relational storage is the star schema. For a MOLAP implementation, the dimension is stored in an analytic workspace that has the same name as the Oracle module in which the time dimension is created. The analytic workspace is stored in the tablespace that is assigned as the users tablespace for the schema that owns the Oracle module containing the dimension.
- **Hierarchy:** Warehouse Builder creates a standard hierarchy that contains all the levels listed on the Levels page of the Create Dimension wizard. The hierarchy contains the levels in the same order that they are listed on the Levels page.

Editing Time Dimension Definitions

To edit a time dimension:

1. From the Warehouse Builder Project Explorer expand the Databases node then the Oracle node.
2. Expand the target module that contains the time dimension to be edited.
3. Right-click the time dimension that you want to edit and select **Open Editor**. You can also double-click the time dimension. Warehouse Builder displays the Data Object Editor for the time dimension.
4. Edit the information on the following tabs:
   - Name Tab on page 13-39
   - Storage Tab on page 13-40
   - Attributes Tab on page 13-40
   - Levels Tab on page 13-41
   - Hierarchies Tab on page 13-41

When you modify a time dimension, a new population map and new implementation tables are created. You can choose to either delete the existing population map and implementation tables or to retain them.

You use the mapping editor to modify the time dimension population map. You must deploy the mapping that populates the time dimension.

If you delete the population map before deploying the map, you cannot populate data into the time dimension. The work around is to run the time dimension wizard again and create another dimension population map.

Name Tab

You use the Name tab to describe the Time dimension. Enter the following details on the Name tab:

- **Name:** The name of the time dimension. The name must be unique within the module. For more information about naming conventions, see "Naming Conventions for Data Objects" on page 4-4.
- **Description:** An optional description for the time dimension.
- **Range of Data:** Specifies the range of the data stored in the time dimension. To specify the range, define the following:
  - **Starting year:** The year from which data should be stored in the time dimension. Click the drop-down list to select a starting year.
  - **Number of years:** The total number of years, beginning from Starting Year, for which the time dimension stores data. Select a value from the drop-down list.

Storage Tab

You use the Storage tab to specify the type of storage for the time dimension. The storage options you can use are Relational or MOLAP.

Relational Selecting the **Relational** option stores the time dimension definition in a relational form in the database. Select one of the following options for the relational implementation of the time dimension:

- **Star schema:** The time dimension is implemented using a star schema. This means that the time dimension data is stored in a single database table or view.
- **Snowflake schema:** The time dimension is implemented using a snowflake schema. This means that the time dimension data is stored in multiple tables or views.

MOLAP Select MOLAP to store the time dimension definition and data in an analytic workspace in the database. This method uses an analytic workspace to store the time dimension data. Provide the following details for a MOLAP implementation:

- **AW Name:** Enter the name of the analytic workspace that stores the time dimension. Click the Ellipsis button to display a list of available AWs. Warehouse Builder displays a node for each module in the current project. Expand a module to view the list of dimensional objects in the module. Selecting an object from list stores the time dimension in the same analytic workspace as the selected object.
- **Tablespace Name:** Enter the name of the tablespace that stores the analytic workspace. If you do not enter a value, Warehouse Builder stores the analytic workspace in the tablespace that is defined as the users tablespace for the schema containing the time dimension metadata.

Attributes Tab

The Attributes tab defines the dimension attributes and the sequence used to populate the dimension key of the time dimension. The Sequence field represents the name of the sequence that populates the dimension key column of the time dimension. You use the **Select** to the right of this field to select a sequence from the Available Sequences dialog. This dialog lists all the sequences that belong to the current project.

Dimension Attributes The Dimension Attributes section lists the dimension attributes of the time dimension. You also use this page to create new dimension attributes. For each attribute, you specify the following details:

- **Name:** The name of the dimension attribute. The attribute name must be unique within the dimension.

- **Description:** An optional description for the attribute.
- **Identifier:** Represents the type of identifier of the attribute. The drop-down lists displays two options: Surrogate and Business. Select the type of identifier.
- **Data Type:** Select the data type of the attribute.
- **Length**: Specify length only for character data types.
- **Precision**: Define the total number of digits allowed for the column. Precision is defined only for numeric data types.
- **Scale:** Define the total number of digits to the right of the decimal point. Scale is defined only for numeric data types.
- **Seconds Precision:** Represents the number of digits in the fractional part of the datetime field. It can be a number between 0 and 9. The seconds precision is used only for `TIMESTAMP` data types.
- **Descriptor:** Select the type of descriptor. The options are: Short Description, Long Description, Start date, End date, Time span, and Prior period.

Levels Tab

The Levels tab defines the levels in the time dimension. You can create additional levels by entering the name and an optional description for the level in the Levels section. For more information on the contents of the Levels tab, see "Level Attributes Page" on page 13-5.

Hierarchies Tab

You use the Hierarchies tab to create additional hierarchies in the time dimension. When you modify the time dimension definition, the map that populates it must reflect these changes. Click **Create Map** to re-create the map that populates the time dimension. For a fiscal time dimension, you can modify the fiscal settings by clicking **Fiscal Settings**. The Fiscal Information Settings dialog is displayed. Use this dialog to modify the fiscal convention, fiscal year start, and fiscal week start.

The Hierarchies tab contains two sections: Hierarchies and Levels.

- **Hierarchies:** Use this section to create hierarchies. Warehouse Builder displays any existing hierarchies in the time dimension. You create additional hierarchies by specifying the name of the hierarchy and type of hierarchy. The options for type of hierarchy are None, Fiscal, Calendar Week, and Calendar Year. Use the **Default** property to indicate which of the hierarchies is the default hierarchy.
- **Levels:** The Levels section lists the levels in the time dimension. When you create a new hierarchy, choose the levels that you want to include in your hierarchy by selecting the Applicable option.

14

Defining Flat Files and External Tables

You can use data from ASCII and binary flat files as sources and targets in Warehouse Builder. For flat file sources, you have the additional option of creating a SQL representation of the data by using an external table.

This chapter includes the following topics:

About Flat Files in Warehouse Builder

You can use flat files as either sources or targets in mappings.

Flat Files as Sources

To use a flat file as a source, first introduce the metadata into the repository. The steps you take depend upon if the source metadata is in ASCII or binary format. For ASCII format, see "Importing ASCII Files into the Repository" on page 14-1. For binary format, see "Adding Existing Binary Files to the Repository" on page 14-2.

After you import the file, you can use it as a source in a mapping. Mappings require files to have a permanent name. However, if your source file is a generated file, that is, given a unique name each time it is generated on the main frame for example, you can provide the new name at runtime.

Importing ASCII Files into the Repository

To import flat file definitions, complete the following steps:

1. Establish network connectivity to the files you wish to import.

   If you are accessing the data files directly, and if the Warehouse Builder client and the data files reside on different types of operating systems, contact your system administrator to establish the required connectivity through NFS or other network protocol.

If the Warehouse Builder client and data files reside on Windows operating systems, store the data files on any drive the Warehouse Builder client machine can access.

2. Creating Flat File Modules on page 14-3

   Create a module for each folder in your file system from which you want to sample existing files.

3. Using the Import Metadata Wizard for Flat Files on page 14-10

   Use the Import Metadata Wizard to select the flat files you wish to import. On the Connection Information page, select the option to launch the Flat File Sample Wizard.

4. Using the Flat File Sample Wizard on page 14-12

   The Flat File Sample Wizard enables you to view a sample of the flat file and define record organization and file properties. The wizard enables you to sample and define common flat file formats such as string and ascii.

   For files with complex record structures, the Flat File Sample Wizard may not be suitable for sampling the data. In such cases, see "Adding Existing Binary Files to the Repository" on page 14-2.

5. Use the flat file as either a source or a target in a mapping.

   If you intend to use the file as a source, consider the advantages of creating an external table based on the file. For more information, refer to "External Table Operators versus Flat File Operators" on page 14-3 and"Creating a New External Table Definition" on page 14-26.

6. Updating a File Definition on page 14-24

   If changes are made to the structure of the flat file, update the file definition.

Adding Existing Binary Files to the Repository

To add existing binary flat files, complete the following steps:

1. Creating Flat File Modules

   Create a module for each folder in your file system to which you want to create new files.

2. Using the Create Flat File Wizard

   Follow the prompts in the wizard, taking notice of the special instructions for naming the file in "Describing a New Flat File" on page 14-5.

3. Use the newly created flat file as a target in a mapping.

About External Tables

An external table is a read-only table that is associated with a single record type in external data such as a flat file. External tables represent data from non-relational source in a relational table format. When you use an external table in a mapping, columns properties are based on the SQL properties you defined when importing the flat file. For more information on SQL properties for flat files, see "SQL Properties" on page 14-9.

When you use an external table as a source in a mapping, you can use it as you would a regular source table. Warehouse Builder generates SQL code to select rows from the external table. You can also get parallel access to the file through the table.

Note: Use external tables are source tables only.

You can either import an existing external table from another database or define a new external table as described in "Creating a New External Table Definition" on page 14-26.

External Table Operators versus Flat File Operators

You can introduce data from a flat file into a mapping either through an external table or a flat file operator. To decide between the two options, consider how the data must be transformed.

When you use an external table operator, the mapping generates SQL code. If the data is to be joined with other tables or requires complex transformations, use an external table.

When you use a flat file operator, Warehouse Builder generates SQL*Loader code. In cases where large volumes of data are to be extracted and little transformation is required, you can use the flat file operator. From the flat file operator, you could load the data to a staging table, add indexes, and perform transformations as necessary. The transformations you can perform on data introduced by a flat file operator are limited to SQL*Loader transformations only.

In Warehouse Builder, you can use an external table to combine the loading and transformation within a single set-based SQL DML statement. You do not have to stage the data temporarily before inserting it into the target table.

For more information on differences between external tables and SQL*Loader (flat file operators in Warehouse Builder), see the *Oracle Database Utilities Manual*.

Flat Files as Targets

When you define a new flat file, typically you do so to create a new target.

Creating a New Flat File as a Target

To design a new flat file, complete the following steps:

1. Creating Flat File Modules

   Create a module for each folder in your file system to which you want to create new files.

2. Using the Create Flat File Wizard

3. Use the newly created flat file as a target in a mapping.

Creating Flat File Modules

Create flat file modules to store definitions of flat files you either import or define yourself.

To create a flat file module:

1. Right-click the **Files** node in the Project Explorer and select **New.**

   Warehouse Builder displays the welcome page for the Create Module Wizard.

2. Define the module in the following steps:

   Describing the Flat File Module

Defining Locations for Flat File Modules

3. The Finish page summarizes the information you provided on each of the wizard pages. When you click **Finish,** the wizard creates the flat file module and inserts its name in the Project Explorer.

   If you checked **Import after finish** earlier in the wizard on the Connection Information Page, Warehouse Builder launches the Import Metadata Wizard described in "Using the Import Metadata Wizard for Flat Files" on page 14-10.

   Or, to define a new flat file, refer to "Using the Create Flat File Wizard" on page 14-5.

Describing the Flat File Module

Type a name and an optional description for the flat file module.

Recall that you create a module for each folder in your file system from which you want to sample existing files or to which you want to create new files. Therefore, consider naming the module based on the folder name. Each file module corresponds to one folder in the file system.

For example, to import flat files from `c:\folder1` and a subfolder such as `c:\folder1\subfolder`, create two file modules such as *C_FOLDER1* and *C_FODLER1_SUBFOLDER*.

Defining Locations for Flat File Modules

Locations for flat file modules are the folders in the file system from which you sample existing files or to which you create new files. You can define a new location or select an existing location.

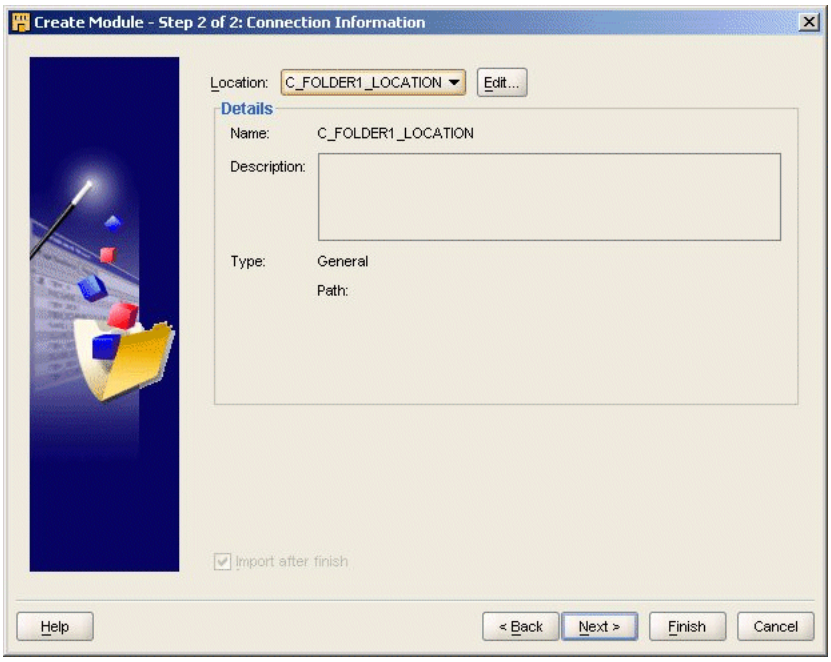
Note that a flat file location identifies a folder in the file system and does not include subfolders.

Connection Information Page

The Connection page displays with a default location named based on the module name you typed. Notice in Figure 14–1 that description and path are blank.

If you intent to import files into this module, click **Edit** to assign values for the location. This location becomes the folder from which you sample existing files or create new files.

Figure 14–1 Connection Information Page in the Create Flat File Modules Wizard



Edit File System Location Dialog

On the Edit File System Location dialog, enter the fully qualified directory, including the drive letter, if appropriate. This

Using the Create Flat File Wizard

Use the Create Flat File Wizard to design a new flat file. The primary use of this wizard is to create a flat file for use as a target in a mapping. After you complete the wizard, you can follow the steps described in "Creating a Source or Target Based on an Existing Flat File" on page 6-4.

Also consider using this wizard as a secondary method for introducing metadata from an existing flat file. This could be the case when the existing file has complex record formats and using the Flat File Sample wizard is not a viable solution.

The Create Flat File Wizard guides you in completing the following steps:

- Describing a New Flat File
- Defining File Properties for a New Flat File
- Defining the Record Type for a New Flat File
- Defining Field Properties for a New Flat File

Describing a New Flat File

Use the Name and Description page to describe the new flat file to create. After you complete the file set up information, click **Next** to continue with the wizard.

- **Name:** This name uniquely identifies the file in the repository. Type a name that does not include a space or any punctuation. You can include an underscore. You can use upper and lower case letters. Do not start the name with a digit. Do not use a Warehouse Builder reserved word. For a list of reserved words, see Chapter 37, "Reserved Words".

- **Default Physical File Name:** If you are creating a new file, you can leave this name blank. If you are defining an existing binary file, type the name of the file. Do not include the file path as you specify this later in the wizard.
- **Character set:** Character sets determine what languages can be represented in database objects and files. Select a character set or accept the default which is the character set defined for the machine hosting Warehouse Builder. For complete information on NLS character sets, see the *Oracle Database Globalization Support Guide.*
- **Description:** You can type in an optional description for the file.

Defining File Properties for a New Flat File

Use the File Properties page to specify Record Organization, Logical Record Definition, Number of Rows to Skip, and the Field Format for the new flat file.

Record Organization

Indicate how to organize the records in the file. Select between the two options to indicate how the length of each record in the file is determined:

- **Records delimited by:** Select this option to designate the end of each record by a delimiter. Then specify that record delimiter. You can accept the default record delimiter, new line (\n), or you can type in a new value.
- **Record length (in characters):** Select this option to create a file with all records having the same length. Then specify the number of characters in each record. For files with multibyte characters, count a multibyte character as one character.

Logical Record Definition

By default, the wizard creates a file in which each physical record corresponds to one logical record. You can override the default to create a file composed of logical records that correspond to multiple physical records.

- **Number of physical records for each logical record:** The data file contains a fixed number of physical records for each logical record.

```
PHYSICAL_RECORD1
PHYSICAL_RECORD2
PHYSICAL_RECORD3
PHYSICAL_RECORD4
```

  In the preceding example, if the number of physical records for each logical record is 2, then `PHYSICAL_RECORD1` and `PHYSICAL_RECORD2` form one logical record and `PHYSICAL_RECORD3` and `PHYSICAL_RECORD4` form a second logical record.

- **End character of the current physical record:** The data file contains a variable number of physical records with a continuation character at the end that signifies that the record is associated with the next physical record.

  In the following example, the continuation character is a percentage sign (%) at the end of the record.

```
PHYSICAL_RECORD1%
PHYSICAL_RECORD2       end log rec 1
PHYSICAL_RECORD3%
PHYSICAL_RECORD4       end log rec 2
```

- **Start character of the next physical record:** The data file contains a variable number of physical records with a continuation character at the beginning of each physical record that signifies that the record is associated with the previous physical record.

  The following example shows two logical records with a continuation character at beginning of the record.

```
PHYSICAL_RECORD1
%PHYSICAL_RECORD2      end log rec1
PHYSICAL_RECORD3
%PHYSICAL_RECORD4      end log rec 2
```

  More than two records can be joined with this technique. The following example shows four physical records for each logical record using continuation at beginning.

```
PHYSICAL_RECORD1
%PHYSICAL_RECORD2
%PHYSICAL_RECORD25
%PHYSICAL_RECORD26   (end log record 1)
PHYSICAL_RECORD3
%PHYSICAL_RECORD4
%PHYSICAL_RECORD45
%PHYSICAL_RECORD46 (end log record 2)
```

Number of Rows to Skip

When creating a new file, you can leave this value blank.

When defining an existing file, indicate the number of records to skip at execution time in **Skip rows.** This is useful for skipping over unwanted header information. If one of the records includes field names, skip the preceding header records so that the record containing field names is first in the file.

Field Format

Select between **Fixed Length** and **Delimited** formats for the file.

To create a delimited file, specify the following properties:

- **Field delimiter:** Field delimiters designate where one field ends and another begins. You can type in a field delimiter or select one from the drop-down list. The drop-down list displays common field delimiters. However, you may type in any character as a delimiter except the ones used for enclosures. The default is the comma (,).
- **Enclosures (Left and Right):** Some delimited files contain enclosures that denote text strings within a field. If the file contains enclosures, enter an enclosure character in the text box or select one from the drop-down list. The drop-down list displays common enclosures. However, you may type in any character. The default for both the left and right enclosure is the double quotation mark (").

Defining the Record Type for a New Flat File

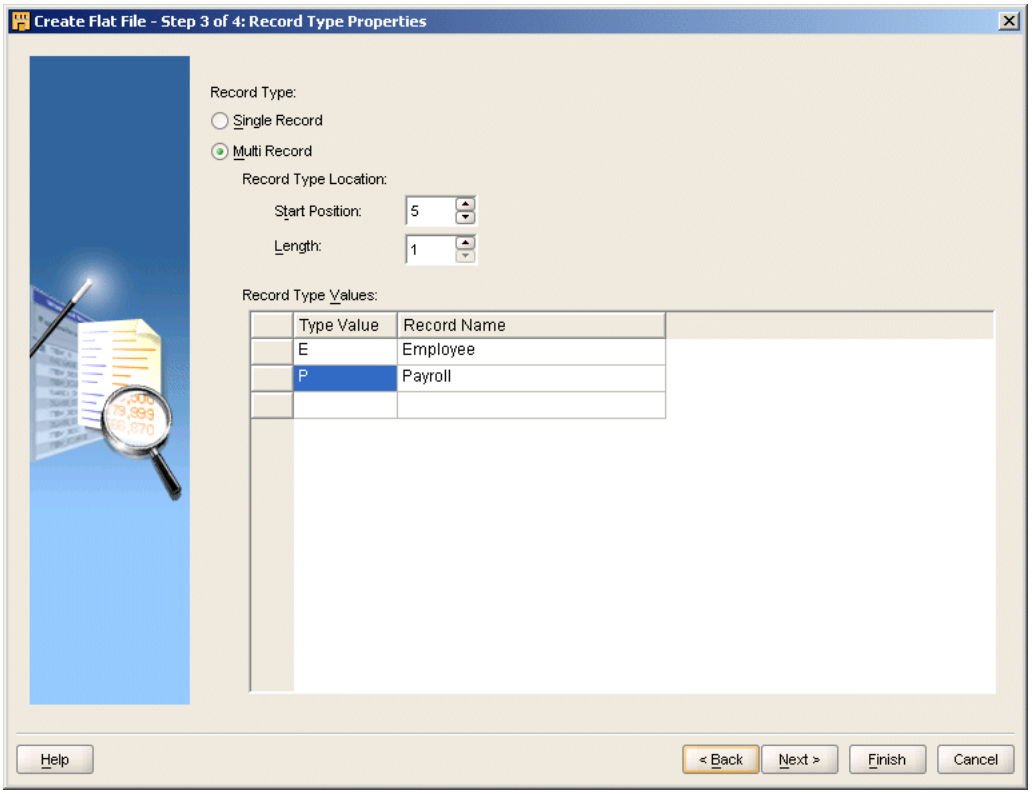
Indicate whether the file you create is to contain a single record type or multiple record types. If the file should contain only one type of record, select **Single.**

If the file should contain more than one record type, select **Multiple.** Use multiple record types, for example, to create a flat file with employee data and payroll data as

shown in Figure 14–2. For each record type you want to create, specify values under **Record Type Location** and then its type value and record name.

Valid entries for Record Type Location depend the field format you selected on the File Properties page, fixed length or delimited fields. For example, if you specified the fields are delimited, indicate the start position and length for the record type. For fixed-length files, indicate the field position.

Figure 14–2 Record Type Properties Page in the Create Flat File Wizard



Defining Field Properties for a New Flat File

Use the Field Properties page to define properties for each field.

Since you can use a flat file in a mapping either directly as a flat file source or indirectly through an external table, the Field Properties page shows both SQL*Loader Properties and SQL Properties. Use the scroll bar to scroll to the right and view all the properties.

The wizard deactivates properties that do not apply to a given data type. For example, you can edit the Length for a CHAR, but precision and scale are not available. Deactivated properties are grayed out.

SQL*Loader Properties

The first set of properties the wizard displays are for the SQL*Loader utility. When you use the flat file directly as a source in a mapping, Warehouse Builder relies upon SQL*Loader and the properties you set here. SQL*Loader properties include Type, Start, End, Length, Precision, Scale, Mask, NULLIF, and DEFAULTIF.

By default, the wizard displays these properties as they appear in the source file. Edit these properties or accept the defaults to specify how the fields should be handled by the SQL*Loader.

Type

Describes the data type of the field for the SQL*Loader. You can use the wizard to import many data types such as CHAR, DATE, DECIMAL EXTERNAL, FLOAT EXTERNAL, INTEGER EXTERNAL, ZONED, AND ZONED EXTERNAL. For complete information on SQL*Loader field and data types, refer to Oracle Database *Utilities*. Currently, only portable data types are supported.

Start

In fixed length files, indicates the field start position.

End

In fixed length files, indicates the field end position.

Length

For delimited files, specifies the maximum field length to be used by SQL* Loader.

Precision

Defines precision for DECIMAL and ZONED data types.

Scale

Defines the scale for DECIMAL and ZONED data types.

Mask

The SQL*Loader uses dd-mon-yy as its default date mask. You can override this default by entering a valid date mask when you describe the file. For example, if the input data has the format DD-Mon-YYYY rather than the SQL*Loader default, you can enter the true format as a mask.

NULLIF

You can override the default action of the SQL*Loader by placing a NULLIF condition on a field. For example, when a character field contains all blanks, you can direct SQL*Loader to mark the field as null rather than storing the blanks. Valid syntax for this field includes *=BLANKS*, *='quoted string'*, *=X'ff'* to indicate hexadecimal values, and *!=* for 'not equal to' logic.

DEFAULTIF

You can override the default action of the SQL*Loader by placing a DEFAULTIF condition on a field. For example, when a numeric or DATE field contains all blanks, SQL*Loader rejects the entire record. To override this action, type *=BLANKS* in the DEFAULTIF property. When SQL*Loader evaluates this condition, it sets the numeric field to zeros and loads the record. Valid syntax for this field includes *=BLANKS*, *='quoted string'*, *=X'ff'* to indicate hexadecimal values, and *!=* for 'not equal to' logic.

SQL Properties

The second set of properties are the SQL properties that include SQL Type, SQL Length, SQL Precision, and SQL Scale. These properties specify how the fields in a flat file translate to the columns in a relational table for example.

The SQL properties you set here have the following implications for mapping design, validation, and generation:

- **External table:** If you create an external table based on a single flat file record type, the columns properties are based on the SQL properties you defined for the flat file. For more information about external tables, see "Using External Tables" on page 14-26.
- **Populating an Empty Mapping Table.** In a mapping, if you populate an empty relational table with the metadata, the table inherits the SQL properties you defined for the flat file source.
- **Flat file target:** If you use the flat file as a target in a mapping, the target does not inherit the SQL properties. Instead, all fields inherit the default SQL*Loader data type. For more information about using a flat file operator as a target, see "Flat File Operator" on page 25-29.

SQL Type

Warehouse Builder supports many SQL data types such as CHAR, DATE, FLOAT, and BLOB.

The wizard assigns a default value for the SQL type based on the SQL*Loader properties you set. If you accept the default SQL type, Warehouse Builder updates that default in the event you later change the SQL*Loader properties. However, if you override the SQL type by selecting a new SQL type from the drop down list, it then becomes independent of the flat file SQL*Loader data type.

SQL Length

This property defines the length for the SQL column.

SQL Precision

When the SQL type is for example NUMBER and FLOAT, this property defines the precision for the SQL column.

SQL Scale

When the SQL type is for example NUMBER and FLOAT, this property defines the scale for the SQL column.

Using the Import Metadata Wizard for Flat Files

If you have existing flat files to use as sources or targets, you can import and then sample the metadata.

To import flat files:

1. Right-click a file module and select **Import.**

   Warehouse Builder displays the welcome page for the Import Metadata Wizard.

2. Click **Next.**

   The wizard displays the Filter Information page. This page gives you the option to filter file names.

   All Data Files: This option returns all the data files available for the directory you specified for the flat file module.

   Data files matching this pattern: Use this option to select only data files that match the pattern you type. For example, if you select this option and enter (*.dat),

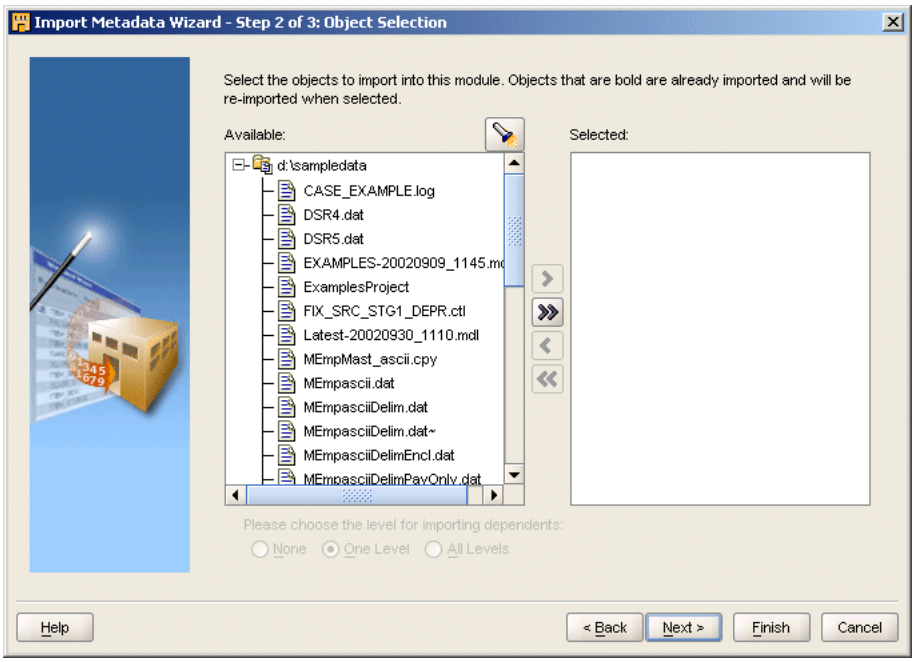
only files with .dat file extensions will be displayed on the next wizard page. If you type % as part of a filter string, it is interpreted as a wild card match for multiple characters. If you type '_' as part of a filter string, it is interpreted as a wild card match for a single character.

3. Click **Next.**

   The wizard displays the Object Selection page, as shown in Figure 14–3.

   Because Warehouse Builder does not provide inbound synchronization for flat files, the available objects will never appear in bold like other objects when they are reimported. When you reimport flat files, you always need to sample the flat file objects again.

Figure 14–3 Object Selection Page



4. Move the name of the files to be described from Available Objects on the left to the Selected Objects window pane on the right.

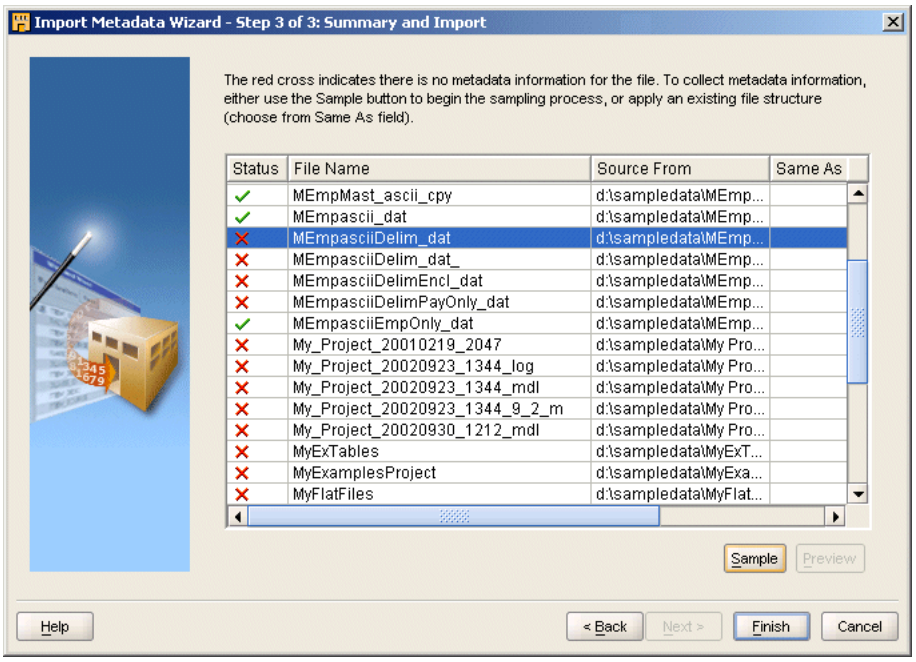
5. Click **Next.**

   The wizard displays the Summary and Import page. The left-most column of this page contains a status ball which indicates if Warehouse Builder has the metadata for the file, as shown in Figure 14–4.

   If the status ball is red, Warehouse Builder does not have the metadata. Proceed to the next step.

6. For each file on the Summary and Import page, either click **Sample** or select a file with a matching format from the **Same As** list box. If the format for a file matches that of another file on the Summary and Import page, you can select the file name from the **Same As** list box.

Figure 14–4 Summary and Import Page Showing File Status



7. Select a file that has a red x mark and click **Sample** at the bottom of the Summary and Import page.

   The wizard displays the welcome page for the Flat File Sample Wizard. Complete the Flat File Sample Wizard. For more information on the Sample Wizard, see Using the Flat File Sample Wizard.

8. After you complete the Flat File Sample Wizard for one file, Warehouse Builder returns to the Summary and Import page. The file you sampled is now displayed with a green check.

9. Once you sample all the files you want to import, click **Finish**. Warehouse Builder creates a definition for file, stores the definition in the source module, and inserts the format name in the source module Project Explorer.

Using the Flat File Sample Wizard

Each time you use the Import Metadata Wizard to sample data from existing flat files, you have the option to launch the Flat File Sample Wizard.

Use the Flat File Sample Wizard as an aid in defining metadata from flat files. Note that although this wizard samples delimited and fixed length files, the Flat File Sample Wizard does not sample multibyte character file with a fixed record format. For these and other complex record formats such as binary files, consider "Using the Create Flat File Wizard" on page 14-5.

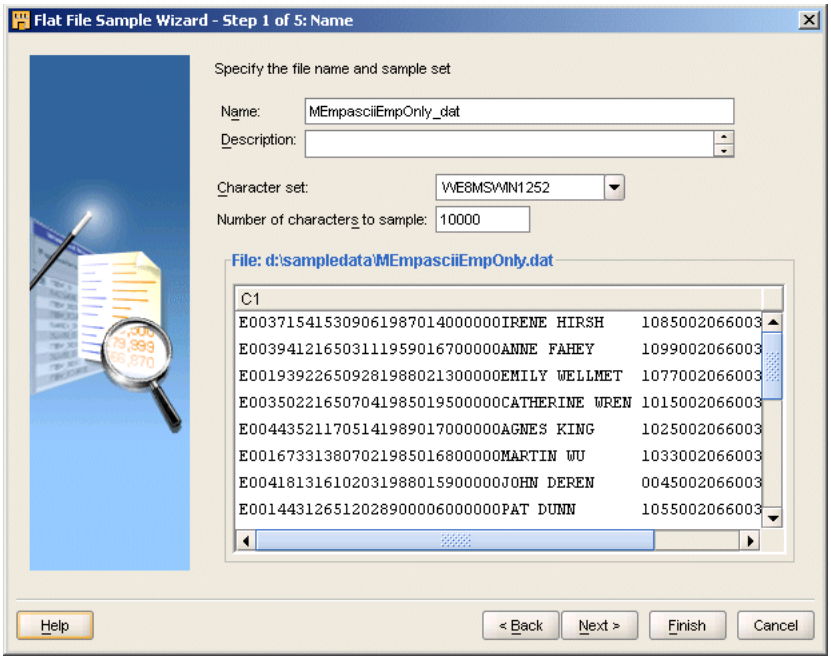
After you complete the Flat File Sample Wizard, it stores the metadata in the Warehouse Builder repository and you can use the flat files as sources or targets in a mapping as described in "Creating a Source or Target Based on an Existing Flat File" on page 6-4.

The Flat File Sample Wizard guides you in completing the following steps:

Describing the Flat File

Use the Name page shown in Figure 14–5 to describe the flat file you are sampling.

Figure 14–5 Name Page for the Flat File Sample Wizard



- **Name:** This name uniquely identifies the file in the Warehouse Builder repository. By default, the wizard creates a name based on the name of the source file by replacing invalid characters with an underscore. For example, if the file name is *myfile.dat*, the wizard assign the repository name *myfile_dat*.

  If you rename the file, do not include a space or any punctuation in the name. You can include an underscore. You can use upper and lower case letters. Do not start the name with a digit. Do not use a Warehouse Builder reserved word. For a list of reserved words, see Chapter 37, "Reserved Words"

- **Description:** You can type in an optional description for the file.

- **Character set:** Character sets determine what languages can be represented in database objects and files. In the Warehouse Builder client, the default Globalization Support or National Language Support (NLS) character set matches the character set defined for the machine hosting Warehouse Builder. If the character set differs from that of the source file, the data sample might appear unintelligible. You can display the data sample in the character set native to the source by selecting it from the drop-down list. For complete information on NLS character sets, see the *Oracle Database Globalization Support Guide.*

- **Number of characters to sample:** You can indicate the number of characters for the wizard to sample from the data file. By default, the wizard samples the first 10,000 characters. To determine an optimum value for this field, see "Example: Flat File with Multiple Record Types" on page 14-17.

After you complete the file set up information, click **Next** to continue with the wizard.

- Describing the Flat File
- Selecting the Record Organization

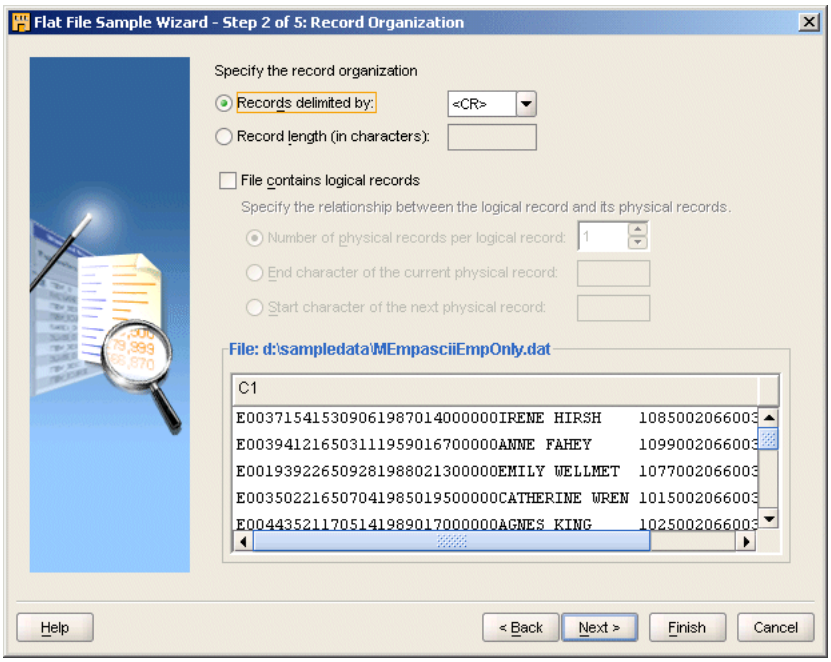
- Selecting the File Format
- Selecting the File Layout
- Selecting Record Types (Multiple Record Type Files Only)
- Specifying Field Lengths (Fixed-Length Files Only)
- Specifying Field Properties

Selecting the Record Organization

Use the Record Organization page to indicate how records are organized in the file you are sampling.

Figure 14–6 displays the Record Organization Page for the Flat File Sample Wizard.

Figure 14–6 Record Organization Page for the Flat File Sample Wizard



Select between the two options to indicate how the length of each record in the file is determined:

- **Records delimited by:** Select this option if the end of each record is designated by a delimiter. Then specify that record delimiter. You can accept the default record delimiter, carriage return (<CR>), or you can type in a new value.
- **Record length (in characters):** Select this option if each record in the file is the same length. Then specify the number of characters in each record. For files with multibyte characters, count a multibyte character as one character.

Specifying Logical Records

The Flat File Sample Wizard enables you to sample files composed of logical records that correspond to multiple physical records. If the file contains logical records, click **File contains logical records.** Then select one of the options to describe the file.

The wizard updates the display of the logical record in the lower panel to reflect your selection. The default selection is one physical record for each logical record.

After you complete the logical record information, click **Next** to continue with the wizard.

- **Number of physical records for each logical record:** The data file contains a fixed number of physical records for each logical record.

  ```
  PHYSICAL_RECORD1
  PHYSICAL_RECORD2
  PHYSICAL_RECORD3
  PHYSICAL_RECORD4
  ```

  In the preceding example, if the number of physical records for each logical record is 2, then `PHYSICAL_RECORD1` and `PHYSICAL_RECORD2` form one logical record and `PHYSICAL_RECORD3` and `PHYSICAL_RECORD4` form a second logical record.

- **End character of the current physical record:** The data file contains a variable number of physical records with a continuation character at the end that signifies that the record is associated with the next physical record.

  In the following example, the continuation character is a percentage sign (%) at the end of the record.

  ```
  PHYSICAL_RECORD1%
  PHYSICAL_RECORD2       end log rec 1
  PHYSICAL_RECORD3%
  PHYSICAL_RECORD4       end log rec 2
  ```

- **Start character of the next physical record:** The data file contains a variable number of physical records with a continuation character at the beginning of each physical record that signifies that the record is associated with the previous physical record.

  The following example shows two logical records with a continuation character at beginning of the record.

  ```
  PHYSICAL_RECORD1
  %PHYSICAL_RECORD2      end log rec1
  PHYSICAL_RECORD3
  %PHYSICAL_RECORD4      end log rec 2
  ```

  More than two records can be joined with this technique. The following example shows four physical records for each logical record using continuation at beginning.

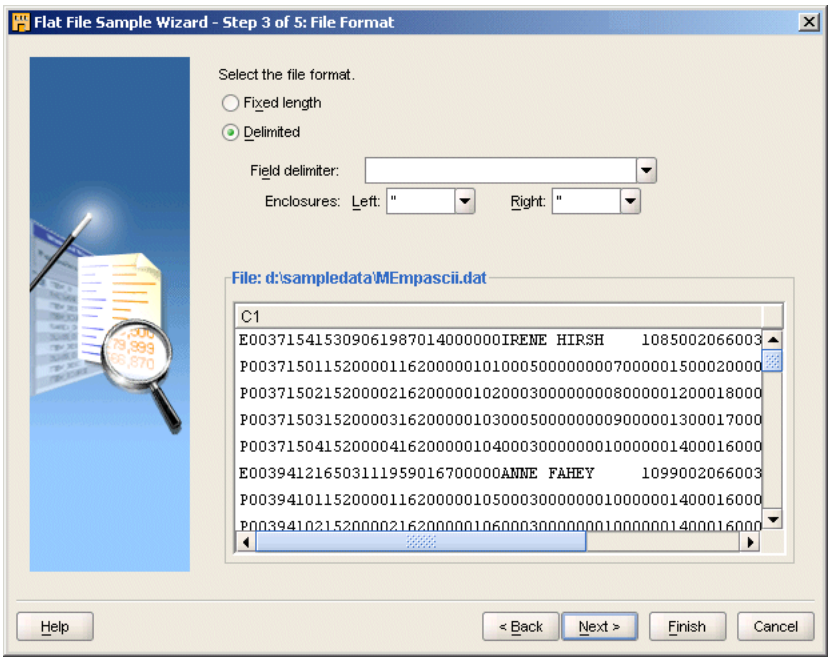
  ```
  PHYSICAL_RECORD1
  %PHYSICAL_RECORD2
  %PHYSICAL_RECORD25
  %PHYSICAL_RECORD26   (end log record 1)
  PHYSICAL_RECORD3
  %PHYSICAL_RECORD4
  %PHYSICAL_RECORD45
  %PHYSICAL_RECORD46 (end log record 2)
  ```

Selecting the File Format

Use the File Format page to select between **Fixed Length** and **Delimited** formats for the file. The Flat File Sample Wizard does not sample multibyte character file with a fixed record format. Instead, consider "Using the Create Flat File Wizard" on page 14-5.

Figure 14–7 displays the File Format page of the Flat File Sample Wizard.

Figure 14–7 File Format Page for the Flat File Sample Wizard



When you select a file format, the wizard updates the sample displayed at the bottom of the wizard page. You can use the scroll bars to navigate the sample data.

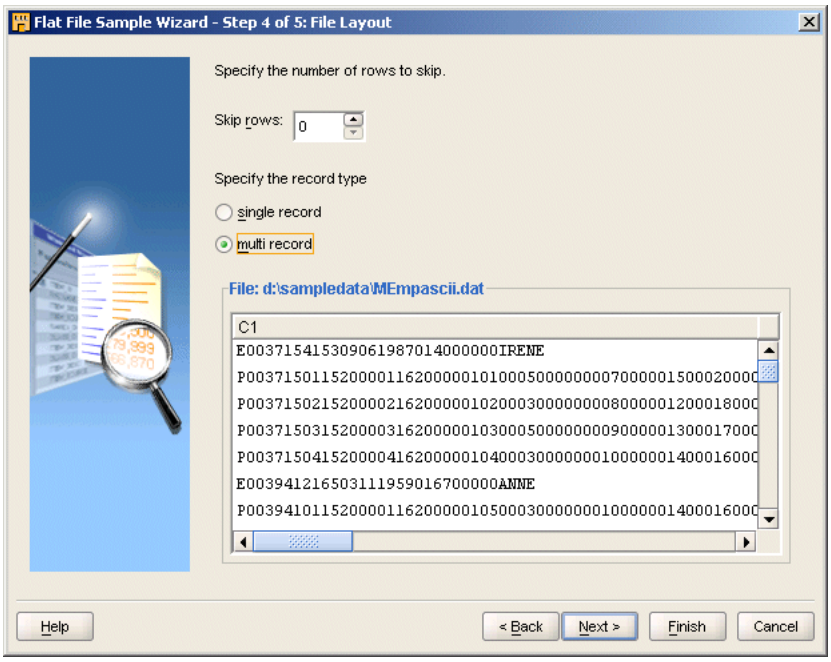
When your file is delimited, specify the following properties:

- **Field delimiter:** Field delimiters designate where one field ends and another begins. You can type in a field delimiter or select one from the drop-down list. The drop-down list displays common field delimiters. However, you may type in any character as a delimiter except the ones used for enclosures. The default is the comma (,).
- **Enclosures (Left and Right):** Some delimited files contain enclosures that denote text strings within a field. If the file contains enclosures, enter an enclosure character in the text box or select one from the drop-down list. The drop-down list displays common enclosures. However, you may type in any character. The default for both the left and right enclosure is the double quotation mark (").

Selecting the File Layout

Use the File Layout page shown in Figure 14–8 to specify the number of rows to skip and to select between a single record type versus multiple record types.

Figure 14–8 File Layout Page for the Flat File Sample Wizard



Indicate the number of records to skip in **Skip rows.** This is useful for skipping over unwanted header information. If one of the records includes field names, skip the preceding header records so that the record containing field names is first in the file. Later in the wizard, on the Field Properties page, you can instruct the wizard to use that record for field names if you are defining a single record file type.

Indicate whether the file contains a single record type or multiple record types.

- If the file contains only one type of record, select **Single.**
- If the file contains more than one record type, select **Multiple.** Later in the wizard you can instruct the wizard to scan the file for the record types. For more information on multiple record types, see "Selecting Record Types (Multiple Record Type Files Only)" on page 14-17.

Selecting Record Types (Multiple Record Type Files Only)

Use the Record Types wizard page to scan the flat file for record types, add or delete record types, and assign type values to the record types.

> **Note:** This step in not necessary for files with a single record type. If the data file has a single record type and fixed length file format, proceed to "Specifying Field Lengths (Fixed-Length Files Only)" on page 14-20. If the data file has a single record type and delimited file format, proceed to "Specifying Field Properties" on page 14-21.

Example: Flat File with Multiple Record Types

In files with multiple record types, one of the fields distinguishes one record type from the next. Figure 14–9 shows an example of a comma delimited file with two record types, "E" and "P". When you use the Flat File Sample Wizard, you instruct the wizard to scan a specified field of every record for the record type values. In this case, instruct the wizard to scan the first field. The wizard returns "E" and "P" as the type values.

Figure 14–9 Example of a File with Multiple Record Types

```
"E],003715,4,153,09061987,0140000.00,"IRENE HIRSH    ],1,085.00,2,066.00,3,088.00,4,125.00
"P],003715,01152000,01162000,00101,0005000.00,0007000.00,150.00,200.00,133.00,075.00,055.00,066.00,077.00
"P],003715,02152000,02162000,00102,0003000.00,0008000.00,120.00,180.00,120.00,065.00,044.00,075.00,055.00
"P],003715,03152000,03162000,00103,0005000.00,0009000.00,130.00,170.00,110.00,055.00,033.00,065.00,066.00
"P],003715,04152000,04162000,00104,0003000.00,0010000.00,140.00,160.00,100.00,045.00,056.00,075.00,065.00
"E],003941,2,165,03111959,0167000.00,"ANNE FAHEY     ],1,099.00,2,066.00,3,088.00,4,125.00
"P],003941,01152000,01162000,00105,0003000.00,0010000.00,140.00,160.00,100.00,045.00,056.00,075.00,065.00
"P],003941,02152000,02162000,00106,0003000.00,0010000.00,140.00,160.00,100.00,045.00,056.00,075.00,065.00
"P],003941,03152000,03162000,00107,0003000.00,0010000.00,140,00.160.00,100.00,045.00,056.00,075.00,065.00
"E],001939,2,265,09281988,0213000.00,"EMILY WELLMET   ],1,077.00,2,066.00,3,088.00,4,125.00
"P],001939,01152000,01162000,00108,0003000.00,0010000.00,140.00,160.00,100.00,045.00,056.00,075.00,065.00
"P],001939,02152000,02162000,00109,0003000.00,0010000.00,140.00,160.00,100.00,045.00,056.00,075.00,065.00
```

When you use the wizard to sample flat files with multiple record types, ensure that the sample size you specified on the Name page is large enough to include each record type at least once. The default is 10,0000 characters.

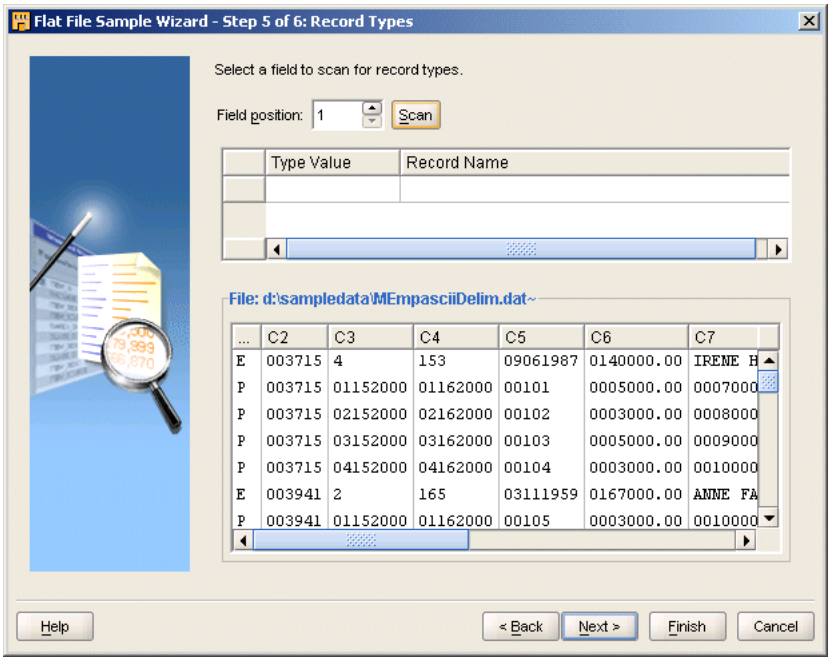
Because sampling cannot be cancelled after it has been started, make sure you pick a "reasonable" number of characters for optimum performance. If all record types do not appear within a reasonable number of characters, you can mock up a sample file with rows selected from different parts of the master file to provide a representative set of data. If you do not know your data well, you may choose sample the entire file. If you know your data well, you can scan a representative sample and then manually add new record types.

Defining Multiple Record Organization in a Delimited File

When a delimited flat file contains several different types of records, you can use the scanning feature within the Flat File Sample Wizard to search and label record types.

Figure 14–10 shows the first field position selected for scanning for multiple record types.

Figure 14–10 Record Types Page for Delimited Files



To complete the Records Type page for a delimited file:

1. Select the one field that identifies the record types in the file.

   The wizard displays all the fields in a sample in the lower panel of the page. Select the field from the sample box. Or, in Field position, you can type in the position as it appears in the sample. Unless you specify otherwise, the wizard defaults to the first field in the file.

   The wizard scans the file for the field and displays the type values. The wizard assigns default record names (RECORD1, RECORD2...) to each type value.

2. You can edit the record names.

   Click a record name to rename it or select a different record name from the drop down list. You can associate a record name with multiple record type values. You can also add or delete type values using the **New** and **Delete** buttons.

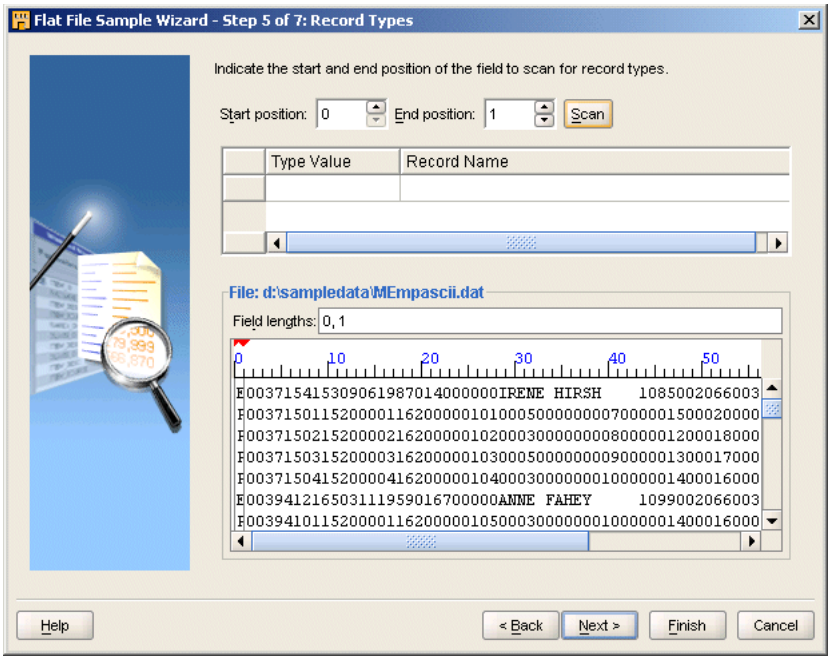
3. Click **Next** to continue with the wizard.

Defining Multiple Record Organization in a Fixed-Length File

When a fixed-length flat file contains several different types of records, you can use the scanning feature within the Flat File Sample Wizard to search for record types and assign a type value to each record type.

Figure 14–11 shows the results from scanning for record types based on the first field position.

Figure 14–11 Record Types Page for Fixed Length Files



To complete the Records Type page for a fixed-length file:

1. Specify the one field that identifies the record types in the file. Type in values for the **Start position** and **End position.** If you want to scan for records based on the first field, enter 0 for **Start Position.**

   The wizard indicates the selected field with a red tick mark in the ruler in the file sample in the lower panel of the page.

2. Click **Scan.**

The wizard scans the file field and displays the type values. The wizard assigns default record names (RECORD1, RECORD2...) to each type value.

3. You can edit the record names.

   Click a record name to rename it or select a different record name from the drop down list. You can associate a record name with multiple record type values. You can also add or delete type values using the **New** and **Delete** buttons.

4. Click **Next** to continue with the wizard.

Specifying Field Lengths (Fixed-Length Files Only)

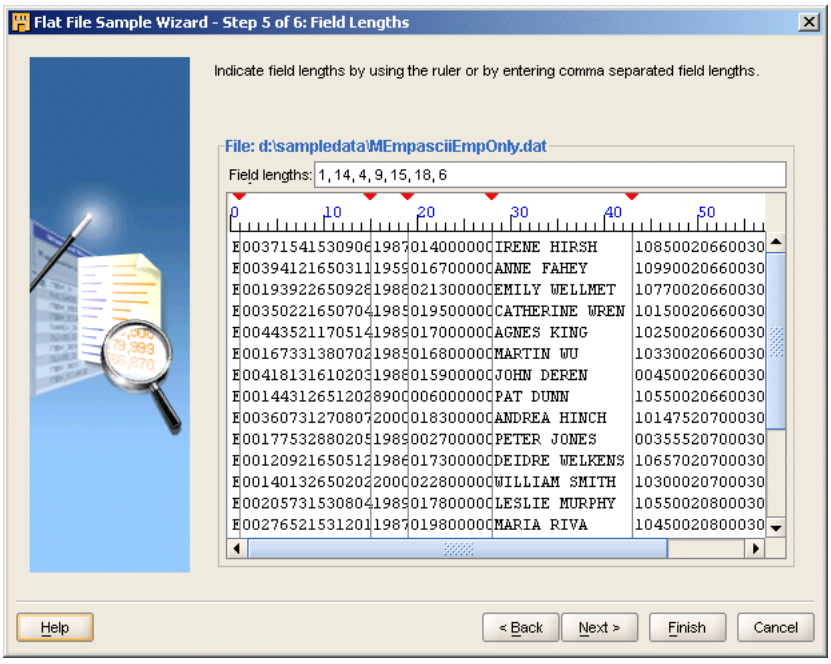
When you use the Flat File Sample Wizard to define a fixed-length flat file, you also need to define the length of each field in the file.

> **Note:** This step is not necessary for delimited files. Proceed to "Specifying Field Properties" on page 14-21.

You can define field lengths by typing in the field lengths or by using the ruler.

Figure 14–12 displays the Field Lengths page from the Flat File Sample Wizard.

Figure 14–12 Field Lengths Page for the Flat File Sample Wizard



If you know the length of each field, type in the field length in **Field Lengths.** Separate each length by commas. The wizard displays the changes to the sample at the bottom of the wizard page.

To use the ruler, click any number or hash mark on the ruler. The wizard displays a red tick mark on top of the ruler and marks the boundary with a red line. If you make a mistake, double-click the marker to delete it or move the marker to another position. Use the ruler to create markers for each field in the file.

Note that when you specify a field length using the ruler, your tick markers indicate the starting and ending borders for each field. From this information, Warehouse

Builder determines the positions occupied by each field. For example, a three-character field occupying positions 6, 7, and 8 is internally identified with the beginning and ending values of '5,8'.

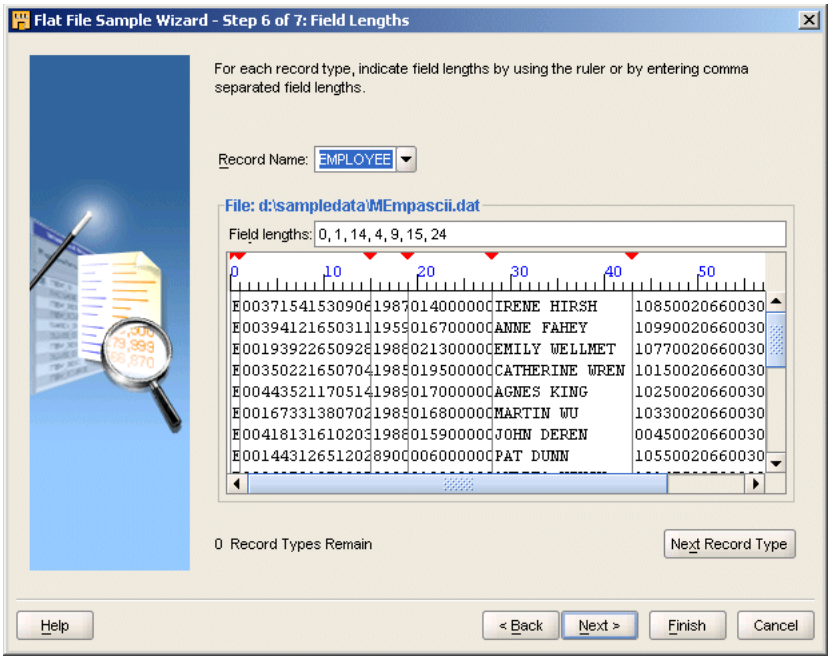
Specifying Field Lengths for Multiple Record Files

You can select the record type by name from **Record Name.** Or, you can select **Next Record Type** from the lower right corner of the wizard page. The number of records with unspecified field lengths is indicated on the lower left corner of the wizard page.

If the flat file contains multiple record types, the wizard prompts you to specify field lengths for each record type before continuing.

Figure 14–13 shows the Field Lengths page for a fixed length file with multiple record types.

Figure 14–13 Field Lengths for Multiple Record Files page



Specifying Field Properties

Use the Field Properties page in the Flat File Sample Wizard to define properties for each field. The wizard assigns a name to each field. It assigns 'C1'to the first field, 'C2' to the second, and so on. To rename fields, click a field and type a new name.

For single record file types, you can instruct the wizard to use the first record in the file to name the fields. Indicate this by checking the box entitled **Use the first record as the field names.** If you choose this option, all the field data type attributes default to CHAR.

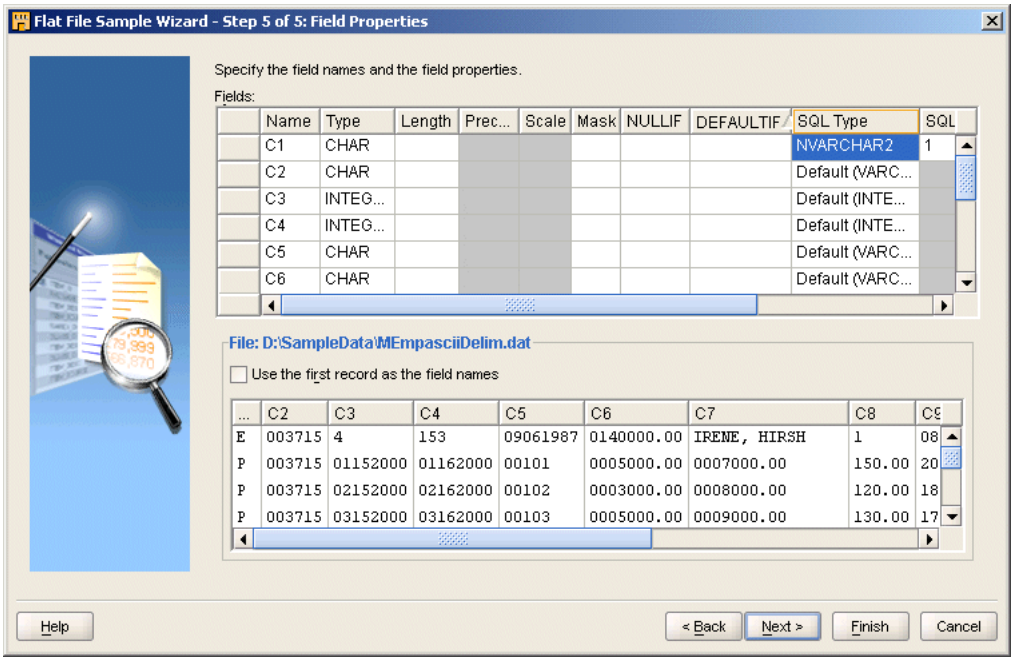
Since you can use a flat file in a mapping either directly as a flat file source or indirectly through an external table, the Field Properties page shows both SQL*Loader Properties and SQL Properties. Use the scroll bar to scroll to the right and view all the properties.

The wizard deactivates properties that do not apply to a given data type. For example, you can edit the length for a CHAR, but precision and scale are not available. Deactivated properties are grayed out.

Note: Once you complete the Field Properties page, verify your selections on the **Summary** page and select **Finish.** The Flat File Sample Wizard returns you to the Import Metadata Wizard described in "Using the Import Metadata Wizard for Flat Files" on page 14-10. You can select more files to sample or select **Finish** to begin the import.

Figure 14–14 shows the Field Properties page for the Flat File Sample Wizard.

Figure 14–14 Field Properties Page for the Flat File Sample Wizard



SQL*Loader Properties

The first set of properties the wizard displays are for the SQL*Loader utility. When you use the flat file directly as a source in a mapping, Warehouse Builder relies upon SQL*Loader and the properties you set here. SQL*Loader properties include Type, Start, End, Length, Precision, Scale, Mask, NULLIF, and DEFAULTIF.

By default, the wizard displays these properties as they appear in the source file. Edit these properties or accept the defaults to specify how the fields should be handled by the SQL*Loader.

Type

Describes the data type of the field for the SQL*Loader. You can use the wizard to import many data types such as CHAR, DATE, DECIMAL EXTERNAL, FLOAT EXTERNAL, INTEGER EXTERNAL, ZONED, AND ZONED EXTERNAL. For complete information on SQL*Loader field and data types, refer to Oracle Database *Utilities*. Currently, only portable data types are supported.

Start

In fixed length files, indicates the field start position.

End

In fixed length files, indicates the field end position.

Length

Specifies the length of the field to be used by SQL* Loader. For delimited files, the field length is not populated, but you can manually edit it if you know the maximum length of the field.

Precision

Defines precision for DECIMAL and ZONED data types.

Scale

Defines the scale for DECIMAL and ZONED data types.

Mask

The SQL*Loader uses dd-mon-yy as its default date mask. You can override this default by entering a valid date mask when you describe the file. For example, if the input data has the format DD-Mon-YYYY rather than the SQL*Loader default, you can enter the true format as a mask.

NULLIF

You can override the default action of the SQL*Loader by placing a NULLIF condition on a field. For example, when a character field contains all blanks, you can direct SQL*Loader to mark the field as null rather than storing the blanks. Valid syntax for this field includes *=BLANKS, ='quoted string', =X'ff'* to indicate hexadecimal values, and *!=* for 'not equal to' logic.

DEFAULTIF

You can override the default action of the SQL*Loader by placing a DEFAULTIF condition on a field. For example, when a numeric or DATE field contains all blanks, SQL*Loader rejects the entire record. To override this action, type *=BLANKS* in the DEFAULTIF property. When SQL*Loader evaluates this condition, it sets the numeric field to zeros and loads the record. Valid syntax for this field includes *=BLANKS, ='quoted string', =X'ff'* to indicate hexadecimal values, and *!=* for 'not equal to' logic.

SQL Properties

The second set of properties are the SQL properties that include SQL Type, SQL Length, SQL Precision, and SQL Scale. These properties specify how the fields in a flat file translate to the columns in a relational table for example.

The SQL properties you set here have the following implications for mapping design, validation, and generation:

- **External table:** If you create an external table based on a single flat file record type, the columns properties are based on the SQL properties you defined for the flat file. For more information about external tables, see "Using External Tables" on page 14-26.
- **Populating an Empty Mapping Table.** In a mapping, if you populate an empty relational table with the metadata, the table inherits the SQL properties you defined for the flat file source.
- **Flat file target:** If you use the flat file as a target in a mapping, the target does not inherit the SQL properties. Instead, all fields inherit the default SQL*Loader data

type. For more information about using a flat file operator as a target, see "Flat File Operator" on page 25-29.

SQL Type

Warehouse Builder supports many SQL data types such as CHAR, DATE, FLOAT, and BLOB.

The wizard assigns a default value for the SQL type based on the SQL*Loader properties you set. If you accept the default SQL type, Warehouse Builder updates that default in the event you later change the SQL*Loader properties. However, if you override the SQL type by selecting a new SQL type from the drop down list, it then becomes independent of the flat file SQL*Loader data type.

SQL Length

This property defines the length for the SQL column.

SQL Precision

When the SQL type is for example NUMBER and FLOAT, this property defines the precision for the SQL column.

SQL Scale

When the SQL type is for example NUMBER and FLOAT, this property defines the scale for the SQL column.

Updating a File Definition

You can update the definition of the file format by editing its property sheet.

To update a file definition:

1. Select the file definition in the Project Explorer.
2. Right-click the file name and select **Properties.**

   Warehouse Builder displays the Flat File property sheet with the following tabs:

   Name Tab: Use this tab to edit the name and descriptive for the file.

   General Tab: Use this tab change the global properties, such as the physical record size, the number of physical records for each logical record, and the delimiter and enclosure characters.

   Record Tab: This tab is available only for flat files with multiple record types. Use this tab to redefine fields or add, delete, or edit record types.

   Structure Tab: Use this tab to edit field level attributes, SQL Loader and SQL Properties.

Name Tab

Use this tab to edit the name, default physical file name, description, and character set for the file.

General Tab

Use this tab to change the global properties, such as the physical record size, the number of physical records for each logical record, and the delimiter and enclosure characters.

For delimited records, the General tab contains the following fields:

- **Records delimited by:** Select this option if the end of each record is designated by a delimiter. Then specify that record delimiter. You can accept the default record delimiter, carriage return (<CR>), or you can type in a new value.
- **Record length (in characters):** Select this option if each record in the file is the same length. Then specify the number of characters in each record. For files with multibyte characters, count a multibyte character as one character.

If the file contains logical records, click **File contains logical records.** Then select one of the following options to describe the file:

- **Number of physical records for each logical record:** The data file contains a fixed number of physical records for each logical record.
- **End character of the current physical record:** The data file contains a variable number of physical records with a continuation character at the end that signifies that the record is associated with the next physical record.
- **Start character of the next physical record:** The data file contains a variable number of physical records with a continuation character at the beginning of each physical record that signifies that the record is associated with the previous physical record.

Record Tab

If the file contains multiple record types, you can select each record type from the **Record Name** field. Warehouse Builder displays the Record sheet and you can edit the record type information.

Record type is in column number: This field displays the column that contains the record type indicator. You can change this value. For example, if you have a flat file with two record types that are distinguished from each other by the first character in the third column as shown in the following list, then the value in this field is 3:

- Record Type 1: `2002 0115 E 4564564`
- Record Type 2: `2003 1231 D 659871 Q HKLIH`

Record type values: This table displays each record type, the value that distinguishes it from the other record types, and the name you have given to the record type. Table 14–1 shows an example of what the record type values for the two sample records earlier might be:

Table 14–1 Example of Record Type Values

Type Value	Record Name
E	Employee
D	Department

- To add new record types, click **New** and enter a Type Value and a Record Name describing the record type.
- To delete record types, select the check box to the left of each record type you want to remove and click **Delete**.

After you have identified and defined the sources for our target system, you are ready to model your target schema.

Structure Tab

Use the Structure tab to edit a field name, data type, mask, SQL*Loader Properties and SQL Properties. You can add or delete a field. You can also add a field mask, NULLIF condition, or DEFAULTIF condition.

If the file contains multiple record types, you can select each record type from the **Record Name** field. Warehouse Builder displays the Record sheet and you can edit the record type information.

The wizard deactivates properties that do not apply to a given data type. For example, you can edit the length for a CHAR, but precision and scale are not available.

Using External Tables

External tables are database objects in the Oracle Database, versions 9*i* and higher. You cannot use external tables with any other database type or any Oracle Database previous to the 9*i* release.

External tables are tables that represent data from flat files in a relational format. They are read-only tables that behave similarly to regular source tables in Warehouse Builder. When you create and define an external table, the metadata for the external table is saved in the Warehouse Builder repository. You can use these external table definitions in mappings to design how you want to move and transform data from flat file sources to your targets.

The following sections provide information about external tables:

- Creating a New External Table Definition
- Synchronizing an External Table Definition with a Record in a File
- Editing External Table Definitions
- Configuring External Tables

Creating a New External Table Definition

Before you begin

Each external table you create corresponds to a single record type in an existing flat file. Before you begin, first define the file within the Warehouse Builder repository by one of two methods described in "Importing ASCII Files into the Repository" on page 14-1 and "Adding Existing Binary Files to the Repository" on page 14-2.

To create a new external table definition:

1. From the Warehouse Builder Project Explorer expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the external table.
3. Right-click the External Tables node and select **New.**

   Warehouse Builder displays the Welcome page of the Create External Table wizard. Use the wizard to complete the following pages:

   - Name Page
   - File Selection Page
   - Locations Page

Name Page

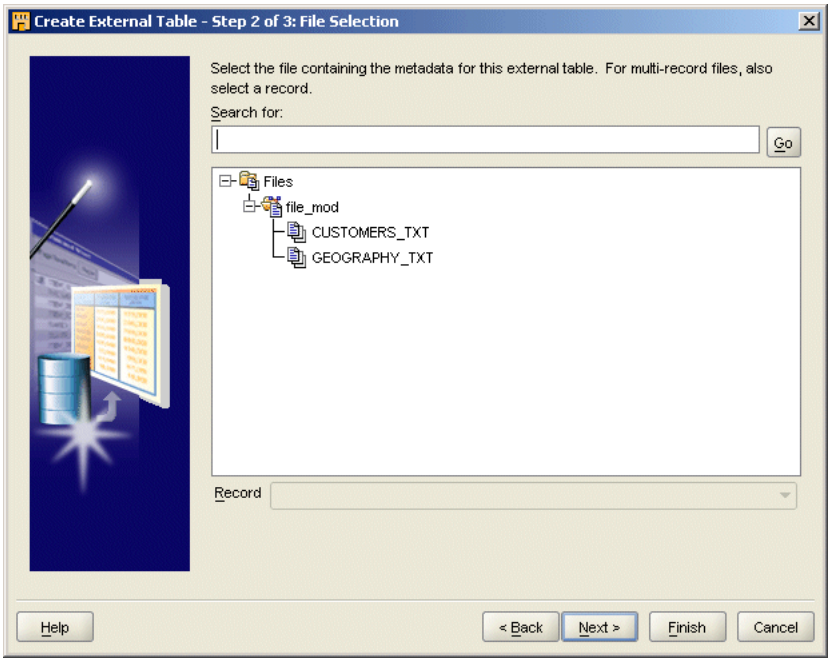
Use the Name page to define a name and an optional description for the external table. Enter the name in the Name field. In physical mode, you must type a name between 1 and 30 valid characters. Spaces are not allowed. In logical mode, you can type a unique name up to 4000 characters in length. The external table name must be unique within the table. Spaces are allowed.

Use the Description field to enter an optional description for the external table.

File Selection Page

The wizard displays the File Selection page as shown in Figure 14–15.

Figure 14–15 File Selection Page for External Table



The wizard lists all the flat files available in the repository. Select a file upon which to base the external table. To search through long lists of files, type the first few letters of the file name and click **Find.**

If you cannot find a file, make sure you either imported or defined the metadata for the file as described in "About Flat Files in Warehouse Builder" on page 14-1.

If you select a file that contains multiple record types, you must also select the record type name at the bottom of the File Selection page. An external table can represent only one record type.

Locations Page

You can select a location from the drop down box which lists the locations associated with flat files. Alternatively, you can leave the location unspecified. If you do not specify a location in the wizard, you can later specify a location on the external table properties sheet.

Tip: You must create and deploy a connector between the locations for the flat file and the Oracle module before you can deploy the external table.

Synchronizing an External Table Definition with a Record in a File

Warehouse Builder enables you to update the external table definition with the metadata changes made to the file associated with the external table. You do this by synchronizing the external table with the source file.

To synchronize an external table definition with a record in a file:

1. In the Project Explorer, right-click the external table that you want to synchronize and select Synchronize.

   Warehouse Builder displays the Reconcile dialog.

2. Use the Select the Object to synchronize drop-down list to specify the flat file with which the external table is to be synchronized.

   By default, Warehouse Builder displays the flat file that was used to create the external table in this drop-down list. Expand the list to see a list of flat file modules and the flat files they contain.

3. Click Advanced.

   Warehouse Builder displays the Advanced dialog.

4. Use the **Match Strategy drop-down list** to specify how Warehouse Builder searches for matches and updates the external table with the information from the flat file. The options for match strategy are:

   MATCH_BY_OBJECT_ID: This strategy compares the field IDs of that the external table columns references with the field IDs in the flat file.

   MATCH_BY_OBJECT_NAME: This strategy compares the physical names in the external table with the physical names in the flat file.

   MATCH_BY_OBJECT_POSITION: This strategy matches by position, regardless of physical names and IDs. The first external table attribute is reconciled with the first record in the file, the second with the second, and so on. Use this strategy when you want to reconcile the external table with a new record.

5. Use the **Synchronize Strategy** drop-down list to indicate how Warehouse Builder handles differences in metadata between the existing external table definition and the record you specified.

   Merge: Warehouse Builder combines the metadata from the existing external table definition and the record you specified.

   Replace: Warehouse Builder deletes metadata from the external table definition if it does not match the metadata from the record you specified. The resulting reconciled external table contains metadata that matches the file record metadata.

6. Click OK.

   The Advanced dialog is closed and you return to the Reconcile dialog.

7. Click OK to complete synchronizing the external table definition.

Editing External Table Definitions

You use the External Table editor to edit an external table definition. To open the editor, right-click the name of the external table from the Project Explorer and select **Open Editor**. The Edit External Table dialog is displayed. The tabs and properties that you can edit depend on how you defined the external table in the repository.

The External Table Properties window displays with the following tabs:

- Name Tab
- Columns Tab
- File Tab
- Locations Tab
- Data Rules Tab
- Access Parameters Tab (display only under certain circumstances)

Name Tab

Use the Name tab to rename the external table. The same rules for renaming tables apply to external tables. For more information, see "Naming Conventions for Data Objects" on page 4-4.

Columns Tab

Use the Columns tab to add or edit columns. The same rules for adding columns to tables apply to external tables. For more information, see "Editing Table Definitions" on page 12-16.

File Tab

Use the File tab to view the name of the flat file that provides the metadata for the external table. If the source flat file has multiple record types, the File tab also displays the record name associated with the external table. You can update this relationship or change it to a different file and record by reconciling the external table. For more information, see "Synchronizing an External Table Definition with a Record in a File" on page 14-28.

The File tab displays under the following conditions:

- You used the New External Table Wizard to create the external table and you specified a file name.
- You did not specify a file name in the New External Table Wizard, but you reconciled the external table definition with a file and record.

Locations Tab

Use the Location tab to view or change the flat file location. The Location drop-down list displays the available locations. Select a location from this list.

Data Rules Tab

Use the Data Rules tab to define data rules for the external table. For more information about using data rules, see "Using Data Rules" on page 20-35.

Access Parameters Tab

Access parameters define how to read from the flat file. In some cases, the External Table editor displays the Access Parameters tab instead of the File tab.

The tab for the access parameters displays under the following conditions:

- You imported an external table from another repository. In this case, you can view and edit the access parameters.
- You created an external table in an Oracle Database and imported its definition into Warehouse Builder. In this case, you can view and edit the access parameters.
- You used the Create External Table Wizard to create an external table and did not specify a reference file. The access parameters will be empty. Before generating the external table, you must reconcile the external table definition with a flat file record or manually enter the access parameters into the properties sheet.

The access parameters describe how fields in the source data file are represented in the external table as columns. For example, if the data file contained a field `emp_id` with a data type of INTEGER(2), the access parameters could indicate that field be converted to a character string column in the external table.

Although you can make changes to the access parameters that affect how Warehouse Builder generates and deploys the external table, it is not recommended. Warehouse Builder does not validate the changes. For more information on the access parameters clause, see *Oracle Database Utilities Manual.*

Configuring External Tables

Configure the following properties for an external table:

- Access Specification
- Reject
- Data Characteristics
- Parallel
- Field Editing
- Identification
- Data Files

> **Note:** When you import an external table into the repository and when you manually define access parameters for an external table, some external table configuration properties are overruled by settings on the Access Parameters tab in the External Table Properties window.

To configure the physical properties for an external table:

1. Select an external table from the Project Explorer.
2. From the Edit menu, select **Configure.** You can also click the Configure icon from the tool bar.

   The Configuration Property window displays.
3. To configure a property, click the white space and make a selection from the drop down box.

Access Specification

If you imported the external table into the repository or created the external table without specifying a source file, do not configure these properties. Access specification properties are overruled by settings on the Access Parameters tab in the External Table Properties window.

Under **Access Specification**, you can indicate the following file names and locations Warehouse Builder uses to load the external table through SQL*Loader.

- **Bad File:** If you specify a name and location for a bad file, Warehouse Builder directs the Oracle Database to write to that file all records that were not loaded due to errors. For example, records written to the bad file include those not loaded due to a data type error in converting a field into a column in the external table. If you specify a bad file that already exists, the existing file is overwritten.
- **Discard File:** If you specify a name and location for a discard file, Warehouse Builder directs the Oracle Database to write to that file all records that were not loaded based on a SQL *Loader load condition placed on the file. If you specify a discard file that already exists, the existing file is overwritten.
- **Log File:** If you specify a name and location for a log file, Warehouse Builder directs the Oracle Database to log messages related to the external table to that file. If you specify a log file that already exists, new messages are appended.

For each of these files, you can either specify a file name and location, select **Do not use**, or select **Use default location.**

Reject

Under **Reject,** you can indicate how many rejected rows to allow. By default, the number of rejected rows allowed is unlimited. If you set **Rejects are unlimited** to false, enter a number in **Number of rejects allowed.**

Parallel

Parallel: Enables parallel processing. If you are using a single system, set the value to `NONPARALLEL` to improve performance. If you are using multiple systems, accept the default `PARALLEL`. The access driver attempts to divide data files into chunks that can be processed separately. The following file, record, and data characteristics make it impossible for a file to be processed in parallel:

- Sequential data sources (such as a tape drive or pipe).
- Data in any multibyte character set whose character boundaries cannot be determined starting at an arbitrary byte in the middle of a string. This restriction does not apply to any data file with a fixed number of bytes for each record.
- Records with the VAR format

Data Characteristics

If you imported the external table into the repository or created the external table without specifying a source file, do not configure these properties. Data characteristics properties are overruled by settings on the Access Parameters tab in the External Table Properties window.

Under **Data Characteristics** you can set the following properties:

- **Endian:** The default for the Endian property is Platform. This indicates that Warehouse Builder assumes the endian of the flat file matches the endian of the platform on which it resides. If the file resides on a Windows platform, the data is

handled as little-endian data. If the file resides on Sun Solaris or IBM MVS, the data is handled as big-endian. If you know the endian value for the flat file, you can select big or little-endian. If the file is UTF16 and contains a mark at the beginning of the file indicating the endian, Warehouse Builder uses that endian.

- **String Sizes in:** This property indicates how Warehouse Builder handles data with multibyte character sets, such as UTF16. By default, Warehouse Builder assumes the lengths for character strings in the data file are in bytes. You can change the selection to indicate that strings sizes are specified in characters.

Field Editing

If you imported the external table into the repository or created the external table without specifying a source file, do not configure these properties. Field editing properties are overruled by settings on the Access Parameters tab in the External Table Properties window.

Under **Field Editing,** you can indicate the type of whitespace trimming to be performed on character fields in the data file. The default setting in Warehouse Builder is to perform no trim. All other trim options can reduce performance. You can also set the trim option to trim blanks to the left, right, or both sides of a character field.

Another option is set the trim to perform according to the SQL*Loader trim function. If you select SQL*Loader trim, fixed-length files are right trimmed and delimited files specified to have enclosures are left trimmed only when a field is missing an enclosure.

You can indicate how to handle missing fields in a record. If you set the option Trim Missing Values Null to true, fields with missing values are set to NULL. If you set the property to false, fields with missing values are rejected and sent to specified bad file.

Identification

See "Identification" on page 12-32 for details.

Data Files

You must add at least one data file to an external table to associate the external table with more than one flat file.

To add a data file:

1. Right-click the **Data Files** node and select **Create.**

   Type a name for the datafile such as *DATAFILE1.* Your entry displays as a new node in the right panel of the Configuration Properties dialog.

2. Type in the following values for each datafile you define:

   Data File Location: Location for the flat file.

   Data File Name: The name of the flat file including its extension. For example, type *myflatfile.dat*.

15

Defining Business Intelligence Objects

Warehouse Builder provides an end-to-end business intelligence solution by enabling you to integrate metadata from different data sources, designing and deploying it to a data warehouse, and making that information available to analytical tools for decision making and business reporting.

This section contains the following topics:

- Using Business Definitions
- Using the Data Object Editor with Business Intelligence Objects
- Using Business Presentations
- Configuring Business Intelligence Objects
- Accessing Business Intelligence Objects Using Oracle BI Discoverer and Oracle BI Beans

> **Note:** The use of the functionality described in this chapter requires the Warehouse Builder Enterprise ETL Option.

Using Business Definitions

Warehouse Builder introduces Business Intelligence objects that enable you to integrate seamlessly with Oracle Business Intelligence tools such as Discoverer and Business Intelligence (BI) Beans. You can define Business Intelligence objects in Warehouse Builder that enable you to store definitions of business views and presentation templates. You can then deploy these definitions to the Oracle Business Intelligence tools and extend the life-cycle of your data warehouse.

Business definitions are the equivalent of Discoverer EUL objects and enable you to integrate with Oracle BI Discoverer. Business definitions facilitate data analysis of data stored in Warehouse Builder. You can define and then deploy business objects to Oracle BI Discoverer. For more information about business definitions, see "About Business Definitions" on page 5-5.

You can create business definitions or derive them from existing schemas. For information on creating business definitions, see "Creating Business Definitions" on page 15-2. For information about deriving business definitions, see "Deriving Business Intelligence Objects" on page 15-20.

This section contains the following topics:

Creating Business Definitions

Before you derive business definitions to deploy to Discoverer, you must create a module to store your business definitions.

To create a Business Definition module in Warehouse Builder:

1. From the Warehouse Builder Project Explorer, expand the project node.
2. Expand the Business Intelligence node.
3. Right-click **Business Definitions** and select **New**.

   Warehouse Builder opens the Create Business Definition Module Wizard.
4. Follow the wizard steps by clicking **Next.**

Naming the Business Definition Module

In the Name and Description page, type a name and optional description for the Business Definition module. Also, indicate the type of module you are creating.

For more information about naming conventions, see "Naming Conventions for Data Objects" on page 4-4.

Setting the Connection Information

On the Connection Information page, you define the location where you want to deploy your business definitions. For example, this may be the system where you are currently running Oracle Discoverer.

If you want to use a deployment location you previously created, you can select it from the Location drop-down list. Then the connection information for this location displays on the wizard page.

You can also choose to create this location later and skip to the next page. Note that you cannot deploy the Business Definitions successfully until you provide the connection information for this target location.

The wizard initially creates a default target location for the module you are creating. For example, if your module is named DISCOVERER_OBJECTS, then the location will be called DISCOVERER_OBJECTS_LOCATION. You can choose to provide the connection information for this location by clicking **Edit.** Warehouse Builder displays the Edit Discoverer Location dialog. Provide the required information to connect with your target system and click **OK.** For more information on the Edit Discoverer Location dialog, see "Defining Discoverer Locations" on page 15-3.

Note: Deployment of Discoverer locations will fail if the EUL owner does not have the CREATE DATABASE LINK privilege.

Defining Discoverer Locations A Discoverer location provides details about the system to which the business definitions you create are deployed. This system should have Oracle Discoverer EUL version 10.1.2 or later installed.

To define a Discoverer location, enter the following details on the Edit Discoverer Location dialog:

- **Name:** The name of the Discoverer location. Warehouse Builder assigns a default name for the location. You can choose to change this name.
- **Description:** An optional description for the Discoverer location.
- **User Name:** The name of the EUL owner to which you want to deploy your business definitions. You can also specify a user who has administrator privileges.
- **Password:** The password for the user specified in **User Name**.
- **Type:** The type of connection used to connect to the Discoverer EUL. The options you can select are **Host:Port:Service** or **SQL*Net Connection**.

  When you choose SQL*Net Connection, specify the net service name in the **Net Service Name** field. When you choose Host:Port:Service, specify the following additional details.

  Host: The host name of the system on which the EUL exists.

  Port: The default port number is 1521.

  Service Name: The service name of the Oracle Database installation.
- **Version:** Represents the version of Discoverer to which the business definitions should be deployed. The list contains only one value, 10.1. Use this option to deploy to Oracle Discoverer 10*g* Release 2. This includes all Oracle Discoverer 10.1.x versions.

Note: You cannot directly deploy business intelligence objects to versions of Discoverer lower then Oracle Discoverer 10*g* Release 2. You can however use the work around described in "Deploying Business Definitions" on page 15-23.

After you specify these details, you may click **Test Connection** to verify the accuracy of the connection details. The **Test Results** displays the results. Click **OK** to close the dialog.

Reviewing the Summary Information

In the Summary page, review the name and location information for the Business Definition module. Click **Back** if you want to make any changes or click **Finish** to finish creating the Business Definitions module.

After Warehouse Builder creates the Business Definition module, you can locate it on the Project Explorer under the Business Definitions node. Expand the module to see that Warehouse Builder provides a representation for the different objects types that comprise the Discoverer End User Layer (EUL). You can define the following types of Discoverer EUL objects in Warehouse Builder:

- Item Folders
- Business Areas
- Drill Paths
- Lists of Values
- Alternative Sort Orders
- Drills to Detail
- Registered Functions

About Item Folders

Item Folders are equivalent to Folder objects in Oracle Discoverer that map to database tables, external tables or views. They represent a result set of data, similar to a database view. Item Folders also store information just like tables. For example, they can store details about employees or customers of an organization. An Item Folder contains Items that map to columns in a table. Each Item has a name and contains specific type of information. For example, the Item Folder containing details about employees may include Items such as, employee name, start date, and department.

There are two types of Item Folders in Warehouse Builder: Simple and Complex. Simple Item Folders contain items from exactly one table in your Warehouse Builder repository. Complex folders, like database views, provide a method to group Items from multiple Item Folders within the same Business Definition module. Thus, Item Folders also contain joins, calculated items, and conditions.

Note: Warehouse Builder does not support the Discoverer custom folders.

Warehouse Builder creates Item Folders when you derive business definitions from warehouse design objects in your repository, as described in "Deriving Business Intelligence Objects" on page 15-20. You can also manually create a customized Item Folder using the Create Item Folder Wizard or the Data Object Editor. The Data Object Editor is also used to edit item folders.

The following sections contain more information related to Item Folders in Warehouse Builder:

- Editing an Item Folder
- Creating an Item Folder
- Creating a Business Area

Editing an Item Folder

After you derive your design object definitions, Warehouse Builder creates an Item Folder as part of the derived business definitions.

Warehouse Builder provides the Data Object Editor that enables you to edit the name and description of an Item Folder, view its source design objects, edit the Items it contains, and specify or edit any joins or conditions.

To edit an Item Folder:

1. From the Project Explorer, expand your Business Definition module node, then expand the Item Folders node.
2. Right-click the Item Folder name and select **Open Editor**. Or double-click the Item Folder name.

   Warehouse Builder opens the Data Object Editor.
3. Click each of the tabs to edit the Item Folder using the following guidelines.

Name Tab

The Name tab enables you to edit the name and description for the Item Folder. It also lists the item folder type.

Source Items Tab

The Source Items tab displays the available source items for your Item Folder. The available items change depending on the type of Item Folder and the options that are currently selected in the editor.

For simple Item Folders, the Available column displays the relational objects in the current project. For complex Item Folders, the Available column displays item folders in that Business Definition module.

When you are editing an existing item folder, the Selected column displays the source items that were selected at the time of creating the item folder. To select different items as the source, use the reverse shuttle arrows to return the items from the Selected column to the Available column. You then use the shuttle arrows to move the new source item from the Available column to the Selected column.

When you create a simple item folder using the editor, the Selected column displays all the relational objects in the current project. For a complex item folder, the Selected column displays the selected items and their item folders.

Your Selected column can contain related items from multiple Item Folders.

If you want to change the Selected items, then use the reverse shuttle arrows to return the previously selected items. Now select an initial folder item from any of the available Item Folders within the same Business Definition module. You can then select additional folder items that have a relationship with the previously selected item. You cannot select items from unrelated Item Folders. The relationship between the Item Folders are defined by the Joins between them. If your Item Folders do not have any relationships, then use the Joins tab in this editor to specify relationships between two Items Folders.

Items Tab

The Items tab displays the details and properties of all Items in an Item Folder. You can view, create, and edit the following for an Item:

Item Details

- **Name:** Represents the name of an Item. If you want to change the current Item, double-click the name and retype the new name.
- **Visible to User:** Check this box if you want this Item to be visible to a Discoverer user.
- **Description:** Optionally type a description for this Item.

Item Properties

When you select an Item in the Item Details section, this field displays a list of properties for that Item. Each of these properties can be edited as follows:

- **Data Type:** Select the data type for the Item. All the data types are supported by Discoverer.
- **Formula:** You can provide a formula for any calculated items you want to specify. Click the Ellipsis button in this field to open the Formula dialog. This dialog contains a subset of the options in the Expression Builder. Use the Formula dialog to create your calculation. This field is populated after you close the Formula dialog. For more information on the Expression Builder, see "The Expression Builder User Interface" on page 26-4.
- **Database Column:** Displays the name of the database column that maps to this Item.
- **Item Class:** Assign an Item Class that enables you to define properties for the Item. The Item Class drop-down contains Lists of Values, Alternative Sort Orders, and Drills to Detail. You can also remove a reference to an Item Class.
- **Default Position:** Select the position of this Item on a Discoverer report.
- **Default Aggregate:** Indicate if the Item will default to an aggregate in the Discoverer report.
- **Heading:** The title for the Item in a Discoverer report.
- **Format Mask:** The format mask for this Item when it is used in a work sheet.
- **Alignment:** The default alignment used for this Item in a Discoverer report.
- **Word wrap:** The default word wrap setting used for this Item in a Discoverer report.
- **Case Storage:** Select the case storage method.
- **Display Case:** Select in what case the Item information will display in a Discoverer report.
- **Default width:** The default width of the Item when it is displayed in a Discoverer report. The width is in characters.
- **Replace NULL with:** The value to use instead of the Item value if the value is NULL.
- **Content Type:** Describes the content of multimedia data in this Item when used in drilling. If the column contains filenames, set this property to FILE. Else set it to the file extension (avi,wav,jpg) to define the application that should process the data.
- **Max char fetched:** The maximum amount of data that will be fetched from LONG, LONG RAW and BLOB datatypes.

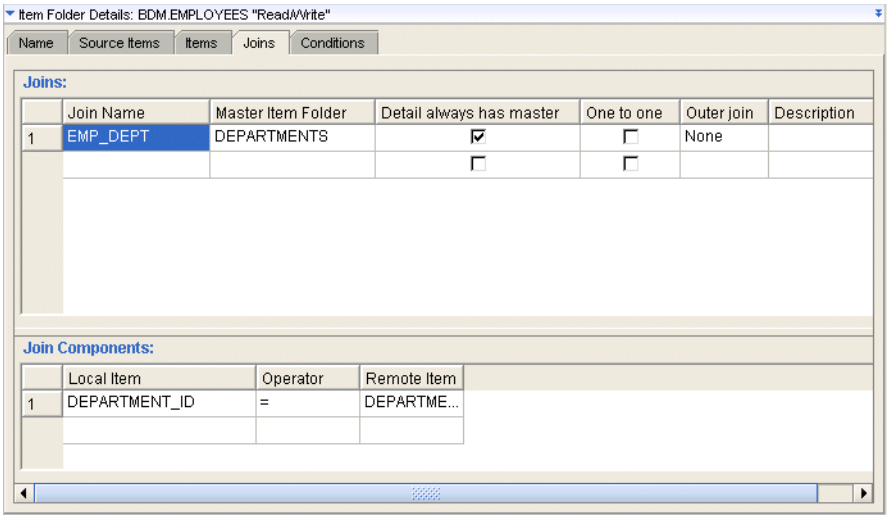
Joins Tab

Joins enable you to associate the data between two Item Folders. During data analysis, you may require information that resides in more than one folder. Joins enable end users to perform business analysis and run reports across multiple Item Folders. After you create joins between Item Folders in Warehouse Builder and deploy them to your Discoverer EUL, they are available for analysis in Discoverer Plus and Discoverer Viewer.

The Joins tab displays the relationships or joins between two Item Folders. You can define new joins by clicking on a new row and providing the required information. You can delete a join by right-clicking the box on the left of each join row and selecting **Delete.**

Figure 15–1 shows how the items in two different item folders SALES and SALES1 are related.

Figure 15–1 Creating and Editing Joins



On the Joins page, click a row in the Specify the Joins field. Provide the following information, as shown in Figure 15–1:

- **Join Name:** Type a name for the join you are creating.
- **Master Item Folder:** Select the Item Folder that will be the Master. In the above example, you select the item folder SALES as your master. This means that you will select an item from the SALES Item Folder to join with the two items you selected from the Item Folder SALES1.
- **Detail always has Master:** Check this box to indicate if your detail Item Folder will always have this master.
- **One to one:** Check this box to indicate a one to one relationship between the two Item Folders.
- **Outer join:** Indicate from the drop down list whether there is an outer join in this relationship and its type.
- **Description:** Optionally describe the join.

For each join, you can specify the Join Components by clicking in the field below and providing the following information:

- **Local Item:** Warehouse Builder automatically populates this drop-down list with the Items contained in the current Item Folder. Select an Item from this list. For example, you can select the Item AMOUNT from the Item Folder SALES1
- **Operator:** Select the relationship between the Local Item you selected and the Remote Item you will select from the Master Item Folder. For example, AMOUNT '=' PRODUCT_LIST_PRICE.
- **Remote Item:** Select an Item from your Master Item folder to join with the Local Item from your local Item Folder. For example, you select PRODUCT_LIST_PRICE from the Item Folder SALES.

  In Figure 15–1, JOIN1 forms a relationship between the two Item Folders, SALES and SALES 1. You can now include the related Items from these two different Item Folder in the same complex Item Folder.

Conditions Tab

The Conditions tab enables you to define or edit a condition that restricts selection on the chosen Item Folder. Use this tab to provide or edit a condition. This tab contains the following:

- **Condition Name:** The name of the condition.
- **Condition:** Click the Ellipsis button in this field to display the Expression Builder. Use this to create or edit a condition. For more information on the Expression Builder, see "The Expression Builder User Interface" on page 26-4.
- **Description:** Optionally describe the condition.
- **Mandatory:** Check this box to specify if the condition is mandatory. A mandatory condition is always applied to filter data that is being retrieved for this item folder. Non-mandatory conditions can be switched on and off by the user.
- **Case Sensitive:** For character data types, specifies if in the condition defined the case should match exactly.

Creating an Item Folder

When you derive intelligence objects, Warehouse Builder creates Item Folders as part of the derived business definitions. However, if you want to define a customized Item Folder, Warehouse Builder enables you to create an Item Folder using the Create Item Folder Wizard.

Item Folders are Discoverer objects that may be of Simple or Complex type. You must specify the type of folder you want to create. Each Item Folder contains items that you can delete or edit after the import, as described in "Editing an Item Folder" on page 15-5.

To create an Item Folder using the Data Object Editor:

1. Expand the Business Definition module in which you want to create an item folder.
2. Right-click **Item Folders** and select **New**, then **Using Editor**.

   The Data Object Editor containing the tabs needed to create an Item Folder is displayed. Use the following tabs to define the Item Folder:

   - Name tab
   - Source Items tab
   - Items tab

- Joins tab
- Conditions tab

For more information on how you specify the details on each tab, refer to the description of these tabs in the "Editing an Item Folder" section on page 15-5.

Alternately, if the Data Object Editor is open, you can use the editor menu or the editor canvas to create an item folder. To create an item folder using the editor menu, select **Add** and then **Item Folder** from the **Diagram** menu. The Add a New or Existing Item Folder dialog is displayed. Follow the steps listed in "Steps to Create an Item Folder" on page 15-9.

To create an Item Folder using the editor canvas, drag and drop an Item Folder icon from the Data Object Editor Palette onto the canvas. Or right-click a blank area on the canvas and select **Add a New Item Folder**. The Add a New or Existing Item Folder dialog is displayed. Follow the steps listed in "Steps to Create an Item Folder" on page 15-9.

Steps to Create an Item Folder Use the following steps to create an Item Folder:

1. Select the **Create a New Item Folder** option.
2. In the **Item Folder Name** field, specify a name for the Item Folder.
3. In the Oracle Module drop-down list, select the name of the Business Definition module to which the Item Folder should belong.
4. In the Item Folder Type drop-down, select the type of item folder to be created. The options are simple and complex.
5. Click **OK**.

The Item Folder is added to the editor canvas. Use the tabs in the Details panel of the Data Object Editor to define the Item Folder. For more information on the contents of these tabs, see "Editing an Item Folder" on page 15-5.

To create an Item Folder using the Create Item Folder Wizard:

1. Expand the Business Definition module in which you want to create an item folder.
2. Right-click **Item Folders** and select **New**, then **Using Wizard**.

   Warehouse Builder opens the Create Item Folder Wizard.
3. Follow the wizard steps by clicking **Next.**

Naming and Describing the Type of Item Folder

In the Name and Description page, type a name and optional description for the Item Folder.

Select whether you want to create a simple or complex folder. You cannot change the folder type after you create it.

Warehouse Builder distinguishes Simple Item Folders from Complex Item Folders in the same way as Discoverer. A simple Item Folder is directly based on columns from a single table in the Warehouse Builder repository and calculated items based on constants or items from that Item Folder. A complex Item Folder can contain items from multiple Item Folders within the same Business Definition module, as well as calculated items.

Selecting Source Items

Select items for your Item Folder.

For a simple Item Folder, you can select exactly one table, view, or external table from any module in the Warehouse Builder repository, to be referenced by the Item Folder. Expand the selected object and proceed to select columns within the selected object, to your selected items. You can multi-select these referenced items by pressing the CTRL key and use the shuttle arrows to move them to the list of selected Items.

A complex Item Folder can contain items from multiple Item Folders within the same Business Definition module. You can select the initial folder items from Item Folder A within a Business Definition module. You can then select additional folder items from another Item Folder B within the same module. However, the two Item Folders A and B must be related. You cannot select items from unrelated Item Folders. Thus, complex Item Folders combine multiple Item Folders that must be joined. You can define the joins in Warehouse Builder using the Data Object Editor for an Item Folder. For more information on creating joins, see "Joins Tab" on page 15-7.

Selecting the Join

When you create a complex item folder, if there is more than one join path between the item folders selected as the item sources, Warehouse Builder displays the Join Selection page. The drop-down list on this page displays all the joins between the item folders. Select the join to be used for the complex item folder being created.

Reviewing the Summary

In the Summary page, review the name and type of your Item Folder as well as items to be included in your Item Folder. Click **Back** if you want to make any changes or click **Finish** to create the Item Folder.

You can locate the Item Folder on the Project Explorer under the Item Folders node in your Business Definition module. This Item Folder contains all the selected items. You can edit the Item Folder properties, create joins and conditions, and edit item properties using the Data Object Editor, as described in "Editing an Item Folder" on page 15-5.

Creating a Business Area

Warehouse Builder enables you to create a Business Area to deploy to a Discoverer EUL. Business areas contain references to Item Folders stored in your Business Definition module and are used to group information about a common subject, for example, Sales Analysis, Human Resources, or Stock Control. The Discoverer end users use these Business Areas as their starting point for building a query.

Business areas only contain references to Item Folders not the actual Item Folder definitions. Thus, a Business Area can contain a collection of unrelated Item Folders and the same Item Folder can appear in multiple Business Areas. This enables you to set up multiple Business Areas with different levels of detail: Sales Analysis area containing one Item Folder, Sales Details area containing six Item Folders, and a Sales Transaction area with 30 Item Folders. When you delete an Item Folder, the reference to it from the Business Area is also deleted.

When you deploy a Business Area using the Design Center, the dependencies of the Business Area are not automatically deployed. For example, your Business Area BUSN_AREA contains two Item Folders, IF1 and IF2. When you deploy BUSN_AREA using the Design Center, IF1 and IF2 are not deployed.

You create a Business Area using either the Create Business Area Wizard or the Data Object Editor. You also use the editor to edit a business area.

To create a Business Area using the Data Object Editor:

1. Expand a Business Definition module.
2. Right-click **Business Areas** and select **New**, then **Using Editor**.

   Warehouse Builder opens the Data Object Editor for the business area.
3. Specify details on the following tabs of the Data Object Editor.
   - Name tab
   - Item Folders tab

   For more information on the contents of these tabs, refer to the description of these tabs in the "Editing a Business Area" section on page 15-12.

Alternately, if the Data Object Editor is open, you can use the editor menu or the editor canvas to create a business area.

To create a business area using the Data Object Editor menu, select **Add** and then **Business Area** from the **Diagram** menu. The Add Business Area dialog is displayed. Create a new business area by selecting **Create a new Business Area**, specifying the name of the business area, selecting the module to which it belongs, and clicking **OK**. The Data Object Editor displays the tabs needed to create a Business Area. These tabs are the same as the ones listed above. Specify values in these tabs.

To create a business area using the Data Object editor canvas, drag and drop a Business Area icon from the editor Palette on to the editor canvas. Or, right- click a blank area on the editor canvas and select **Add a New business Area**. The Add a New or Existing Business Area dialog is displayed. Select **Create a new Business Area** and specify the name of the business area and the module to which it belongs. Click **OK**. The Data Object Editor displays the Name tab and the Item Folders tab. Specify values on these tabs.

To create a Business Area using the Create Business Area Wizard:

1. Expand a Business Definition module.
2. Right-click **Business Areas** and select **New**, then **Using Wizard**.

   Warehouse Builder opens the Create Business Area Wizard.
3. Follow the wizard steps by clicking **Next.**

Naming the Business Area

In the Name and Description page, type a name and optional description for the Business Area.

Selecting the Item Folders

In the Item Folders page, Warehouse Builder displays all the Item Folders available within the Business Definition module. You can multi-select the Item Folders by pressing the CTRL key and use the shuttle arrows to move them to the list of Selected Item Folders.

Reviewing the Summary

In the summary page, review the Item Folders you selected. Click **Back** if you want to make any changes or click **Finish** to finish creating the Business Area.

After Warehouse Builder creates the Business Area, you can locate it on the Project Explorer under the Business Areas node with references to the selected Item Folders stored in it.

To make changes to your Business Area definitions after you create them, Warehouse Builder provides the Edit Business Area dialog. For details, see "Editing a Business Area" on page 15-12.

Editing a Business Area

Warehouse Builder enables you to edit the definitions for a Business Area using the Edit Business Area dialog.

To edit a Business Area:

1. From the Project Explorer, expand the Business Area node.
2. Right-click a Business Area name and select **Open Editor**.

   Warehouse Builder opens the Edit Business Area dialog containing two tabs: Name and Item Folders. Edit these tabs as follows:

Editing the Business Area Name

The Name tab enables you to edit the name and description of a Business Area.

Reviewing Item Folders in a Business Area

The Item Folders tab displays all the Item Folders within the Business Definition module under the Selected Item Folders column. The Item Folders that are not currently included in the Business Area are listed under the Available Item Folders column.

Use the shuttle arrows to include additional Item Folders to the Business Area from the Available Folders column or to remove included Item Folders from the Selected Folders column.

Creating a Drill Path

Warehouse Builder enables you to create a Drill Path to deploy to a Discoverer EUL. Drill Paths define a hierarchy relationship between the items in your Business Definition module. For example, Region, Sub-region, Country, State, and so on. Warehouse Builder creates these drill paths for derived dimensions. You can also create your own customized drill path definitions if you are familiar with your data.

To create a Drill Path:

1. Expand the Business Definition module.
2. Right-click **Drill Paths** and select **Create Drill Path**.

   Warehouse Builder opens the Create Drill Path Wizard.
3. Follow the wizard steps by clicking **Next.**

Naming the Drill Path

In the Name and Description page, type a name and optional description for the Drill Path.

Specifying Drill Levels

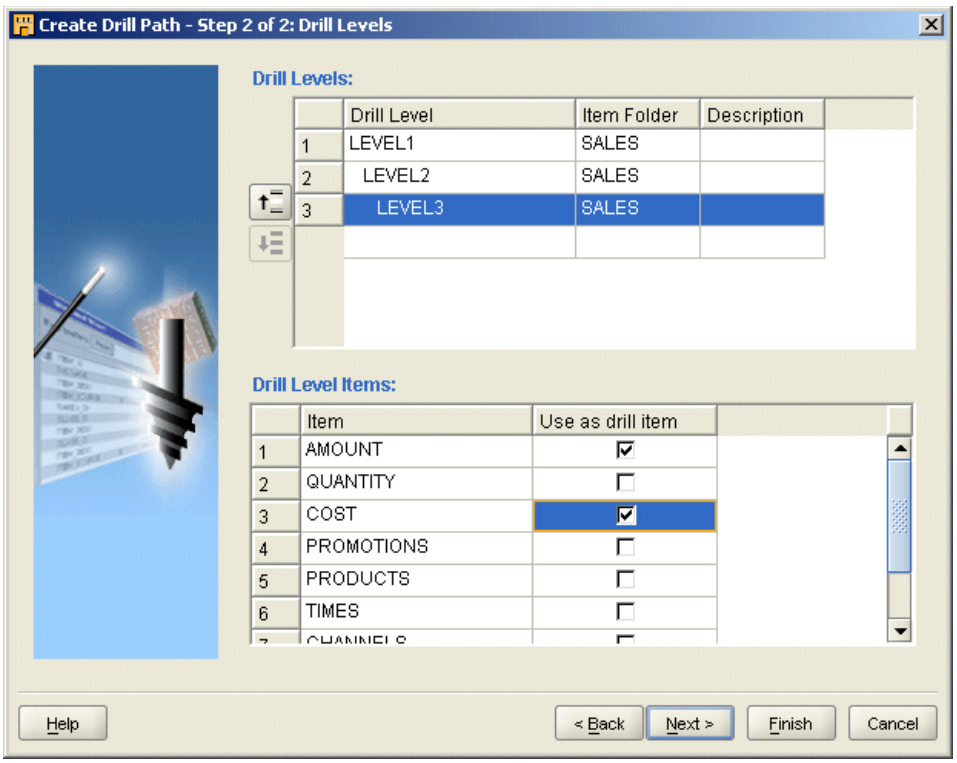
Use the Drill Levels page to define a drill level and specify the Item Folder it references. Optionally, you can provide a description for the Drill Levels. To define drill levels, click on a row and provide the following information:

- **Drill Level:** Type a name for the drill level.
- **Item Folder:** From the drop down field, select the Item Folder it references.
- **Description:** Provide an optional description for the drill level.

When you select a referencing Item Folder for the Drill Level, the wizard lists the available Items within that Item Folder under the Drill Level Items field at the bottom.

In this field, you can specify one or more items to act as drill items. Select the **Use as Drill Item** option for each Item you want to include as a drill item in the level, as shown in Figure 15–2.

Figure 15–2 Creating Drill Levels



Specifying the Join

If there are more than one join paths between the Item Folders referenced by the drill levels, Warehouse Builder displays the Join Selection page. The drop-down list displays a list of the existing joins between the selected Item Folder. Select the join that you want to use for the drill path.

Reviewing the Summary

In the summary page, review the drill levels you are creating. Click **Back** if you want to make any changes or click **Finish** to create the drill path.

You can locate the drill path on the Project Explorer under your Business Definition module. Warehouse Builder enables you to edit a drill path using the Edit Drill Path dialog.

Editing a Drill Path

Warehouse Builder enables you to edit drill paths using the Drill Path using the Edit Drill Path dialog.

To edit a drill path:

1. From the Project Explorer, expand the Drill Paths node.
2. Right-click the Drill Path and select **Open Editor**.

   Warehouse Builder displays the Edit Drill Path dialog containing two tabs: Name and Drill Levels.

Editing the Drill Path Name

The Name tab enables you to edit the name and the description of the drill path.

Reviewing the Drill Levels in the Drill Path

Use the Drill Levels tab to edit the drill levels that you defined. The **Drill Levels** section lists the drill levels along with the item folders that they reference. The Item Folder column displays the item folder that a drill path references. You can modify this by selecting the new item folder from the drop-down list.

The **Drill Level Items** section displays the items that act as drill items. You can modify this list by selecting more items that act as drill items.

Creating Lists of Values

In Discoverer, Lists of Values (LOVs) represents a set of valid values for an item. These are the values in the database column on which the item is based. LOVs enable end users to easily set conditions and parameter values for reports. An example of an LOV can be names of different countries that a user can pick from a drop down list to view a report on the quantities of a product sold in four specific countries.

In Warehouse Builder, you can create lists of values for Item Folders using the Create List of Values Wizard as described below.

To create a List of Values:

1. Expand the Business Definition module.
2. Right-click **Lists of Values** and select **New**.

   Warehouse Builder opens the Create List of Values Wizard.
3. Follow the wizard steps by clicking **Next.**

Naming the List of Values

In the Name and Description page, type a name and optional description for this list of values. Check the **Set as Drill to Detail** box if you also want to set this as a Drill to Detail. When you deploy these definitions to Discoverer, Warehouse Builder creates an Item Class that you can use both as a List of Values and as a Drill to Detail.

Defining Items in a List of Values

The Defining Items page enables you to select the item that will generate your LOV in Discoverer. This page displays all the Items available in your Warehouse Builder repository. Expand the nodes to select an item and click **Next.**

Referencing Items in a List of Values

The Referencing Item page enables you to associate your LOV with different items. The Available Items column displays all the Items available in your Warehouse Builder repository. Expand the nodes to select the items that will reference your list of values. Use the shuttle arrows to move your selections to the Selected Items column and click **Next.**

Reviewing the Summary

In the summary page, review the defining and referencing items selected for the LOV. Click **Back** if you want to make any changes or click **Finish** to finish creating the LOV.

You can locate the LOV on the Project Explorer in the Business Definition module under the Lists of Values node. Warehouse Builder enables you to edit the name, description, and defining and referencing items associated with an LOV using the Edit List of Values dialog.

Editing Lists of Values

Warehouse Builder enables you to edit a list of values using the Edit List of Values dialog.

To edit a list of values:

1. From the Project Explorer, expand the List of Values node.
2. Right-click the List of Values and select **Open Editor**.

   Warehouse Builder displays the Edit List of Values dialog containing the following tabs: Name, Defining Item, Referencing Items, and Options.

Editing the List of Values Name

Use the Name tab to edit the name and description of the list of values.

Editing Items in the List of Values

Use the Defining Item tab to edit the item that generates the list of values in Discoverer. The item that is the defining item is highlighted. To edit this and specify that another item should be used to generate the LOV, select the new item.

Editing Referencing Items

Use the Referencing Items tab to edit the items that reference the list of values. The Selected column lists the items that the list of values references. To add more items to which the list of values references, select the item in the Available column and use the shuttle arrows to move it to the Selected column. To remove items that the list of values currently references, select the item from the Selected column and use the shuttle arrows to move it to the Available column.

Advanced Options for List of Values

Use the Advanced tab to specify advanced options for the list of values. The advanced options are as follows:

- **Retrieve values in groups of:** Use this option to specify the number of values that are retrieved in group. The default value is 100 which means that the values are retrieved in groups of 100.
- **Sort the values and remove duplicates:** Select this option to remove duplicate values from the list of values and to order the values. This ensures that the LOV always shows unique, ordered values.
- **Show values in "Select Items" page of the Worksheet Wizard:** Select this option to enable users to expand the List of Values when selecting items to include in a query.
- **Require user to always search for values:** Select this option to display the Search dialog every time the List of Values is expanded.
- **Cache list of values during each connection:** Select this option to store the list of values when the List of Values is expanded for the first time. This improves performance because otherwise, every time the List of Values is expanded, the values are fetched from the database.

Creating Alternative Sort Orders

In Discoverer, alternate sorts enable end users to display values in a non-standard sequence.

For example, by default the values of the Description item are sorted alphabetically. If you want to sort the description according to the values of the Product Key item, you need to define an alternate sort item and link the two items together. One item defines the sort order and the other defines the item to be sorted.

Define how you want to order the information in your Discoverer EUL using the Create Alternative Sort Order Wizard.

To create an Alternative Sort:

1. Expand the Business Definition module.
2. Right-click **Alternative Sort Orders** and select **New.**

   Warehouse Builder opens the Create Alternative Sort Order Wizard.
3. Follow the wizard steps by clicking **Next.**

Naming the Alternative Sort Order

In the Name and Description page, type a name and optional description for the alternative sort order.

Check the **Set as Drill to Detail** box if you also want to set this as a Drill to Detail. When you deploy these definitions to Discoverer, Warehouse Builder creates an Item Class that can be used both as an Alternative Sort Order and as a Drill to Detail.

Defining Item for the Alternative Sort Order

The Defining Item page enables you to select the Item that contains the values to be sorted. Expand the nodes to select an item and click **Next.**

Defining Order Item for the Alternative Sort Order

Use the Defining Order Item page to select an Item, in the same Item Folder, that defines the order in which the values of the Item you selected on the Defining Item page are displayed. Expand the nodes to select the item and click **Next.**

Referencing Items for the Alternative Sort Order

The Referencing Items page enables you to associate your Alternative Sort Order with different items. The Available column lists all the Items in the Warehouse Builder repository. Expand the nodes to select the items that will reference your Alternative Sort Order. Use the shuttle arrows to move your selections to the Selected column and click **Next.**

Referencing Selection Panel for the Alternative Sort Order

This panel enables you to shuttle across an item that already references an item class. You can either change the reference or decide not to shuttle the item across.

Reviewing the Summary

In the summary page, review the alternative sort order definition. Click **Back** if you want to make any changes or click **Finish** to finish creating the alternative sort order.

You can locate the alternative sort order on the Project Explorer in the Business Definition module under the Alternative Sort Order node. Warehouse Builder enables you to edit the name, description, and defining and referencing items associated with an alternative sort order using the Edit dialog.

Editing Alternative Sort Orders

The Edit Alternative Sort Order dialog enables you to edit an alternative sort order.

To edit an alternative sort order:

1. Expand the Alternative Sort Order node in the Project Explorer.
2. Right-click the Alternative Sort Order and select **Open Editor**.

   The Edit Alternative Sort Order dialog containing the following tabs is displayed: Name, Defining Item, Defining Order Item, Referencing Order Items, and Options.

Editing the Alternative Sort Order Name

Use the Name tab to edit the name and description of the alternative sort order.

Editing the Defining Item

Use the Defining Item tab to edit the item that contains the values to be sorted. This tab displays the Item that currently defines the alternative sort order highlighted. To change this selection, click the item that you now want to use to define the alterative sort order.

Editing the Defining Order Item

The Defining Order Item tab displays the Item Folder with the item that currently defines the order in which the values of the Item selected on the Defining Item tab are displayed. You can change this selection by clicking a new item from the tree.

Editing the Referencing Order Items

The Referencing Order Items tab lists the items that will reference your Alternative Sort Order in the **Selected** column. To add more items to this list, select the item in the **Available** column and use the shuttle arrows to move the item to the Selected column. To remove an item that is already selected, move the item from the Selected column to the Available column using the shuttle arrows.

Advanced Options

Use the Options tab to specify advanced options for the alternative sort order. The options you can set are as follows:

- **Retrieve values in groups of:** Use this option to specify the number of values that are retrieved in group. The default value is 100 which means that the values are retrieved in groups of 100.
- **Sort the values and remove duplicates:** Select this option to remove duplicate values from the alternative sort order and to order the values. This ensures that the alternative sort order always shows unique, ordered values.
- **Show values in "Select Items" page of the Worksheet Wizard:** Select this option to enable users to expand the alternative sort order when selecting items to include in a query.
- **Require user to always search for values:** Select this option to display the Search dialog every time the Alternative Sort Order is expanded.
- **Cache list of values during each connection:** Select this option to store the Alternative Sort Order when it is expanded for the first time. This improves performance because otherwise, every time the Alternative Sort Order is expanded, the values are fetched from the database.

Creating Drills to Detail

In Discoverer, drills to detail enable you to analyze your data thoroughly by navigating through your data and performing drill down operations to obtain detailed information. When you define drills to detail in Warehouse Builder, you define relationships between items. These drills enable you to interactively drill up or down through your data to see a different level of detail. For example, you can move from actuals to budgets for the same department, then look at the department employee details, then drill down to their salary and training histories, then drill to their job grades structure, and so on.

You can define a drill to detail in Warehouse Builder using the Create Drill to Detail dialog.

To create a Drill to Detail:

1. Expand the Business Definition module.
2. Right-click **Drills to Detail** and select **New.**

   Warehouse Builder opens the Create Drill to Detail dialog.

Create Drill to Detail

Name: Type a name for the drill to detail definition.

Description: Provide an optional description for the drill to detail.

The **Available** column at the bottom of the dialog lists the Item Folders in the Business Definition Module. Select a referencing item from this set and use the shuttle arrow buttons to move it to the **Selected** column.

Editing Drills to Detail

Use the Edit Drill to Detail dialog to edit a Drills to Detail.

To edit a Drills to Detail:

1. Expand the Drills to Detail node in the Project Explorer.
2. Right-click the name of the Drill to Detail and select **Open Editor**.

   The Edit Drill to Detail dialog is displayed. The contents of this dialog are the same as the Create Drill to Detail dialog. In addition to modifying the name and description of the drill to detail, you can edit the referencing items. for more details on the contents of the Drill to Detail dialog, see "Create Drill to Detail" on page 15-18.

Creating Registered Functions

In Discoverer, you can use custom PL/SQL functions to perform operations or calculations on values in an Item. To access these functions in Discoverer, the user-defined functions are registered in the EUL. If you want to use any of those registered user-defined functions in Discoverer, you need to include that information in your Warehouse Builder definitions.

You can define a registered function using the Create Registered Function Wizard as described below.

To create a Registered Function:

1. Expand the Business Definition module.
2. Right-click **Registered Function** and select **New**.

   Warehouse Builder opens the Create Registered Function Wizard.
3. Follow the wizard steps using the following guidelines.

Naming the Registered Function

In the Name and Description page, type a name and optional description for the alternative sort order.

From the drop-down menu, select a return type for the function. Select the **Available to User** option to indicate if a Discoverer end-user can use this registered function in calculations.

Specifying the Function Parameters

Specify the function parameters by clicking on a row and typing a name for the parameter. From the drop-down list, select the data type for the parameter. Check the box to indicate if this parameter is required or not and type an optional description.

Reviewing the Summary

In the Summary page, review the function definition. Click **Back** if you want to make any changes or click **Finish** to finish creating the registered function.

You can locate the registered function on the Project Explorer in the Business Definition module under the Registered Functions node. Warehouse Builder enables you to edit the name, description, and parameters of the function using the Edit dialog.

Editing Registered Functions

Use the Edit Registered Function dialog to edit a registered function.

To edit a registered function:

1. Expand the Registered Functions node in the Project Explorer.

2. Right-click the registered function and select **Open Editor**.

   The Edit Registered Function dialog containing the following tabs is displayed: Name and Parameters.

Renaming a Registered Function

Use the Name tab to edit the name and the description of the registered function.

Modifying the Parameters of a Registered Function

Use the Parameters tab to edit the parameters of the registered function. You can edit the name, type, and description of a parameter. Add new parameters by clicking on an empty row and specifying the name of the parameter and its datatype. Delete a parameter by right-clicking the grey cell to the left of the parameter name and selecting **Delete**.

Deriving Business Intelligence Objects

Warehouse Builder enables you directly derive business intelligence objects from your data warehouse design definitions. When you run the Perform Derivation Wizard on a warehouse module, it automatically tailors the existing definitions to those required by an Oracle Discoverer End User Layer. For example, the Perform Derivation Wizard organizes the metadata into Item Folders and Drill Paths ready to be integrated with a Discoverer EUL.

To derive Business Intelligence objects in Warehouse Builder:

1. From the Warehouse Builder Project Explorer, select an Oracle module that you want to derive. This indicates that you are deriving all the objects contained in that module. Alternatively, you can also choose to derive one object definition at a time. For example, you can select an individual table or dimension to derive.
2. Right-click the name of the warehouse module or object and select **Derive**.

   Warehouse Builder opens the Perform Derivation Wizard.
3. Follow the wizard steps using the following guidelines.

You can also launch the Perform Derivation Wizard from the Data Object Editor using the following steps:

1. Open the Data Object Editor for the object to be derived.

   You can do this by double-clicking the object name in the Project Explorer. Alternately, you can right-click the object in the Project Explorer and select **Open Editor**.
2. From the **Object** menu, select **Derive**.
3. Follow the wizard steps using the following guidelines.

Selecting Source Objects

The Source Objects page enables you to select additional objects for derivation. The **Available** column displays all the objects available in your Warehouse Builder repository for deployment to Discoverer. These objects can belong to different warehouse modules. You can also select a collection for derivation. The Oracle module or object you selected before starting the wizard displays in the Selected Objects column.

Expand the nodes in the Available column and use the shuttle arrows to select the objects you want to derive. Select the **Automatically add the Dimensions** option to derive the dimension objects that are associated with the selected cube objects.

Selecting a Target for the Derived Objects

In the Target page, indicate the Business Definition module in which you want to store the definitions for the derived objects. For example, if you created a Business Definition module called DISCOVERER_OBJECTS, then the name of that module will display on this page. Select DISCOVERER_OBJECTS and click **Next.** You can also select a business area as the target. In this case, Warehouse Builder creates shortcuts to the item folders in the business areas. It is recommended that you deploy to a Business Area. Otherwise, when you deploy the objects, they will not belong to any Business Area and thus will not be shown to end-users of Discoverer tools.

When you select a collection for derivation, if the target is a business area, the individual objects contained in the collection are derived. Shortcuts are created to these item folders from the business area. If the target is a Business Definition module, Warehouse Builder creates a business area with the same name as the collection, stores the objects in the collection as item folders in the Business Definition module, and creates shortcuts to these item folders from the business area.

Specifying Derivation Rules

In the Rules page, specify the derivation rules and parameters. Warehouse Builder loads, configures, and executes these rules to derive the business intelligence definitions from the selected design object definitions. You can set parameters for different rule types by selecting the type of objects from the **Rules** drop-down menu. For example, you can set global rules, rules for relational objects, rules for dimension objects, or rules for cube objects. The rules and parameters that you can set are displayed on the page.

Select the **Show advanced parameters** option to display certain advanced rules for an object. You can also set parameters for more than one rule type.

Setting Global Rules

You can specify the following parameters for creating Discoverer EUL:

- **Preserve user changes:** Select to preserve any manual changes to the display properties name and description.
- **Log level:** Specify the level of detail you want to see in the log file by selecting one of the options from the drop-down menu. You can choose to record only errors, warnings, information, or trace debug information.
- **Log file location:** Provide a path on your local system to store your log file. For example, *..\..\iobuilder\derive.log*.
- **Validate before derive:** Check the box if you want Warehouse Builder to validate the selected objects before deriving them.
- **Abort on error:** Check the box if you want Warehouse Builder to stop the derivation if it encounters an error.
- **Capitalize:** Check the box if you want to capitalize the names of the derived objects.
- **Replace underscores with spaces:** Check the box if you want to replace the underscores in the names with spaces after derivation.

You can specify the following rule for Relational objects:

- **Bound Table Suffix:** Specify a suffix for the bound tables you want to derive.
- **Default Aggregate:** Specify the default aggregate function to be applied to numeric measures.
- **Remove Column name prefixes:** Check the box if you want to remove the text immediately before an underscore in the column name. The prefix is removed provided the same prefix is used for all columns.
- **Sort items by name:** Check this option to sort the items alphabetically.

You can specify the following rules for Dimensions:

- **Always build item folders for the dimension:** Check this option to force the Perform Derivation Wizard to create Item Folders for the derived dimension definitions.
- **Build Item Folders for the levels:** Check this option to force the Perform Derivation Wizard to create Item Folders for the derived dimension levels.
- **Drill Paths on Item Folders for the levels:** Check this option if you want the Perform Derivation Wizard to create Drill Paths on Item Folders being created for each dimension level. This option applies only if item folders are created for each level.
- **Prefix Items with Level Name:** Check this option if you want to prefix the item names with the dimension level names.
- **Prefix separator:** If you choose to prefix the item names with the dimension level names, then indicate a prefix separator. The default is an underscore.
- **Sort items by name:** Check this option if you want to sort the items alphabetically.
- **Derive Dimension Roles:** Check this option if you want the Perform Derivation Wizard to derive additional item folders for each role.

You can specify the following rules for Cubes:

- **Sort items by name:** Check this option if you want to sort the items alphabetically.

Reviewing the Pre Derivation Rules

The Pre Derivation page displays the objects to be derived and the target or Business Definition module for storing the derived definitions.

Review this information and click **Next** to perform the derivation.

Reviewing Derivation Progress

The Derivation page displays a progress bar indicating the status of the derivation. When the progress bar displays 100%, the Message Log field displays any errors or warnings. At the end, the log indicates if the derivation was completed successfully.

Click **Next** to view the list of derived objects.

Finishing the Derivation

The Finish page displays the list of derived objects. Click **Finish** to accept the derivation. If you are not satisfied and you want to perform the derivation again, click **Back** to repeat the process.

Warehouse Builder displays the derived definitions in your Business Definition module. You can edit the Item Folder definitions or create additional definitions for deployment to Discoverer.

Deploying Business Definitions

After you create your business definitions, you can deploy them using the Control Center. The business definitions are deployed to the Discoverer location associated with the Business Definition module that contains these business definitions. Before you deploy business definitions, ensure that a valid Discoverer location is associated with the Business Definition module. For information about how to associate a Discoverer location with a Business Definition module, see "Setting the Connection Information" on page 15-2.

Note that you can directly deploy business definitions only to Discoverer 10*g* Release 2. When you deploy business definitions to Discoverer 10*g* Release 2, Warehouse Builder creates an .eex file that contains the objects, connects to the EUL specified using the Discoverer location, and imports the .eex file into Oracle Discoverer. During the import, any new business definitions are appended on top of the existing definitions. You must validate the EUL and remove redundant definitions. For example, you deploy an item folder that contains four items. Subsequently, you delete one item from the item folder. When you redeploy the item folder, it still contains four items. This is because only new definitions are being appended, but old definitions are not removed.

Deploying Business Definitions to Versions Lower than of Discoverer 10*g* Release 2

You cannot directly deploy business definitions to versions of Discoverer lower than Discoverer 10*g* Release 2. However, you can still transfer your business definitions to Discoverer using the following work around.

When you deploy business definitions to a location that points to a version of Discoverer lower than Discoverer 10g Release 2, the deployment will fail. But Warehouse Builder still creates an .eex file that contains the business definitions. This .eex file is assigned a default name, for example, *2022.eex*, and is stored in the `<OWB_ORACLE_HOME>\owb\deployed_scripts` directory. You can connect to the EUL using Discoverer and import this .eex file.

Using the Data Object Editor with Business Intelligence Objects

Apart from using the Data Object Editor to create business areas and item folders, you can perform the following tasks:

- Create Business Areas, see "Creating Business Areas Using the Data Object Editor" on page 15-23.
- Add Item Folders to a Business Area, see Create "Adding Item Folders to a Business Area" on page 15-24.
- Create Item Folders, see "Creating Item Folder Using the Data Object Editor" on page 15-24.
- Add Items to an Item Folder, see "Adding Items to An Item Folder" on page 15-25.
- Synchronize Item Folders with the underlying database table, see "Synchronizing Item Folders" on page 15-26.
- Create joins, see "Creating Joins Using the Data Object Editor" on page 15-27.

Creating Business Areas Using the Data Object Editor

To create a Business Area using the Data Object Editor:

1. On the Data Object Editor canvas, navigate to the Business Definition tab.

2. Right-click a blank area on the canvas and select **Add a Business Area**.

   Warehouse Builder displays the Add a New or Existing Business Area dialog.

3. Select **Create a New Business Area** and specify a name for the Business Area. Also select the Business Definition module to which the Business Area belongs using the **Business Definition Module** drop-down list.

4. Click **OK**.

   Warehouse Builder creates the Business Area and adds an icon representing the Business Area to the canvas.

5. To add Item Folders to a Business Area, follow steps 3 to 7 in the section "Adding Item Folders to a Business Area" on page 15-24.

Adding Item Folders to a Business Area

You can use the Data Object Editor canvas to add item folders to a Business Area. Use the following steps:

1. Open the Data Object Editor for the Business Area to which you want to add Item Folders.

   To do this, right-click the Business Area name in the Project Explorer and select **Open Editor.** Alternately, double-click the name of the Business Area in the Project Explorer.

2. Navigate to the Business Definition tab of the canvas.

3. Drag and drop an Item Folder icon from the Palette onto the canvas.

   The Add a New of Existing Item Folder dialog is displayed.

4. Choose **Select an existing Item Folder**.

5. From the selection tree, choose the Item Folder that you want to add to the Business Area.

6. Click **OK**.

   Warehouse Builder adds an icon that represents the Item Folder on the canvas.

7. Hold down your mouse on the Items group of the Item Folder, drag and then release on the Item Folders group of the Business Area.

   The Item Folder is added to the list of item folders in the Item Folders group of the Business Area.

You can delete an Item Folder from a Business Area by right-clicking the Item Folder name in the Business Area and selecting Delete.

Creating Item Folder Using the Data Object Editor

To create an Item Folder using the Data Object Editor:

1. Open the Data Object Editor and navigate to the Business Definition tab.

2. Right-click a blank area on the canvas and select **Add an Item Folder**.

   The Add a New or Existing Item Folder dialog is displayed.

3. Select **Create a New Item Folder**.

4. Specify the following details for the Item Folder:

- Specify a name for the Item Folder using the **New Item Folder Name** field. A default name is assigned initially. You can choose to use this name or edit it.
- Specify the Business Definition module to which the Item Folder belongs. The **Business Definition Module** drop-down list displays a list of the available business definition modules. Select a module from this list.
- Specify the type of Item Folder to be created using the **Item Folder Type** drop-down list.

5. Click **OK**.

   Warehouse Builder displays a node that representing the Item Folder on the canvas.

6. Right-click the Item Folder node and select **Detail View**.

   The Details tab that contains a node for the Item Folder is displayed.

7. From the Palette, drag and drop the icon representing the type of object on which the Item Folder is based on to the canvas. For example, if your Item Folder is based on a table, drag and drop a Table icon from the Palette on to the canvas.

   The Add a New of Existing <Object> dialog is displayed.

8. Use this dialog to select the object on which the Item Folder is based.

   Warehouse Builder adds a node for this object on the canvas.

9. Map the required columns from the database object to the Items group of the Item Folder.

Adding Items to An Item Folder

You can use the Data Object Editor to add items to an Item Folder. Follow these steps:

1. Open the Data Object Editor for the Item Folder to which you want to add Items.

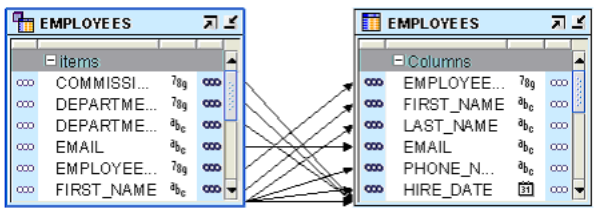
   You can do this by right-clicking the Item Folder name in the Project Explorer and selecting **Open Editor**. Alternately, double-click the Item Folder name in the Project Explorer.

2. On the Business Definition tab of the canvas, right-click the Item Folder and select Detail View.

   Warehouse Builder displays an additional tab that has the same name as the Item Folder. This tab contains the Item Folder and the source object that is used to create the items in the Item Folder as shown in the Figure 15–3.

   In the case of simple item folders, the source object is a table or view. In the case of a complex item folder, the source object is an item folder.

Figure 15–3 Item Folder and its Source Object



3. From the editor Palette, drag and drop the icon that represents the source item onto the canvas. The source item can be a table or a view. Alternately, you can right-click a blank area on the canvas and select **Add a Table** or **Add a View**.

   Warehouse Builderdisplays the Add a New or Existing <Object> dialog.

4. Select the Select an existing <Object> option.

5. From the selection tree, choose the name of the object that contains the source data for the items.

6. Click **OK**.

   Warehouse Builder adds the source object to the canvas.

7. From the attribute that stores the source data, drag the Item that you want to add and drop it on the Items group of the Item Folder.

   Warehouse Builder maps the source attributes to the target object.

You can also use the Data Object Editor canvas to delete items from an Item Folder. Right-click the item from the Item Folder and select **Delete**.

Synchronizing Item Folders

Item Folders are defined based on existing tables, views, or external tables. When the definition of the underlying object changes, you can update the Item Folder definition by synchronizing it with the object on which it is based.

To synchronize an Item Folder:

1. Expand the Item Folders node in the Project Explorer.

2. Right-click the Item Folder and select **Open Editor**.

   The Data Object Editor for the Item Folder is displayed.

3. On the canvas, right-click the node that represents the Item Folder and select **Synchronize**.

   The Synchronize Item Folder dialog is displayed.

4. Review the details displayed on this dialog and click **OK**.

   Warehouse Builder synchronizes the item folder with the data object on which the item is based.

Synchronize Item Folder Dialog

The Synchronize Item Folder dialog enables you to update the Item Folder with any changes made to the data types used in the database object on which the Item Folder is based. This dialog displays the details of the changes to be made to the Item Folder.

The Synchronize Item Folder dialog contains three columns: Object, Reason, and Action. The Object column lists the component in the underlying database object that has changed. The Reason column displays a brief description of the reason for the synchronization. The Action column displays the action that will be taken by Warehouse Builder to synchronize the Item Folder. The available actions are Update and None. If you choose None for a component, no synchronization is performed for that object. Only definitions that have an Action set to Update are synchronized.

For example, the Item Folder DEPT_ITMF is derived from the DEPT table. After the Item Folder is created, you modify the DEPT table and change the data type of the column LOCATION from VARCHAR2 to NUMBER. When you synchronize the Item Folder DEPT_ITMF, the Synchronize Item Folder dialog displays LOCATION in the

Object column. The Reason column displays "Datatype mismatch". The Action column displays Update.

Click **OK** to perform the actions listed on the Synchronize Item Folder dialog and update the Item Folder definition. If you do not wish to perform the actions listed on this dialog, click **Cancel**.

Creating Joins Using the Data Object Editor

To create a join, ensure that the Data Object Editor canvas is displaying both the Item Folders between which a joins is being created. To do this, you open the Data Object Editor for one of the Item Folders. Next use the following steps to add the second item folder:

1. Drag and drop an Item Folder icon onto the canvas. Alternately, you can right-click the canvas and select Add an Item folder.

   Warehouse Builder displays the Add a New or Existing Item Folder dialog.

2. Select the **Select an Existing Item Folder** option.

3. Select the Item folder from the selection tree.

   Warehouse Builder adds the item folder to the Data Object Editor canvas.

Once you have both item folders on the Data Object Editor canvas, you can create a join between them by mapping the item from the source Item Folder to the corresponding item in the target Item Folder. The default join condition used is '='. You can change this by editing the join.

Alternately, you can use the following steps:

1. Right-click the Joins group of the Item Folder, either in the Business Definition tab or the detail view, and select **Add a Join**.

   Warehouse Builder displays the Folder Join dialog.

2. Specify a name for the join and click **OK**.

   Warehouse Builder adds this join to the Joins group of the Item Folder.

3. Select an Item from the owning Item Folder and drag it on to the Join to create the local item.

4. Select an Item from the other Item Folder and drag it on to the Join to create the remote item.

Using Business Presentations

Business presentations enable you to integrate with Oracle BI Beans. You can define business presentations using Warehouse Builder metadata and then deploy these definitions to Oracle BI Beans. To define business presentations, you first define a business presentation module that acts as the container for your presentation objects. For more information about business presentations, see "About Business Presentations" on page 5-5.

The following sections describe these topics:

- Editing Presentation Templates on page 15-30

Creating Business Presentation Modules

Business Presentation modules store definitions for presentation templates in Warehouse Builder. You can later deploy these presentation templates to BI Beans.

Define a Business Presentation module using the Create Business Presentation Module Wizard as described below.

To create a Business Presentation Module:

1. Expand the Business Intelligence node.
2. Right-click **Business Presentations** and select **New.**

   Warehouse Builder opens the Create Business Presentation Module Wizard.
3. Follow the wizard steps using the following guidelines.

Naming the Business Presentation Module

In the Name and Description page, type a name and optional description for the business presentation module. From the drop-down menu, select the module status.

For more information about naming conventions, see "Naming Conventions for Data Objects" on page 4-4.

Specifying the Deployment Location

Specify the target location where you want to deploy your business presentation definitions. This is the BI Beans catalog in which you want to define the business presentations.

If you want to use a deployment location that you created previously, select the location using the **Location** drop-down list.

By default, the wizard creates a location for the module you are creating. For example, if your module is called BUSN_PRES, the wizard creates a location called BUSN_PRES_LOCN. To provide the connection information for this location, click **Edit**. The Edit BI Beans Location dialog is displayed. For more information about this dialog, see "Defining BI Beans Locations" on page 15-28. You can skip this step at this time and create a deployment location later. Note that you cannot successfully deploy the definitions until you create and specify a target location.

Defining BI Beans Locations A BI Beans location provides details about the system to which the business presentations that you create are deployed. This system should have Oracle BI Beans installed.

To define a BI Beans location, enter the following details on the Edit Discoverer Location dialog:

- **User Name:** The name of the BI Beans catalog owner to which you want to deploy your business definitions.
- **Password:** The password for the user specified in **User Name**.
- **Host:** The host name of the system on which the BI Beans catalog exists.
- **Port:** The default port number is 1521.
- **Service Name:** The service name of the Oracle Database installation.

After you specify these details, you may click **Test Connection** to verify the accuracy of the connection details. The **Test Results** displays the results. Click **OK** to close the dialog.

Reviewing the Summary

In the summary page, review the name and deployment location information for the new module. Click **Back** if you want to make any changes or click **Finish** to finish creating the module.

You can locate the Business Presentations module on the Project Explorer under the Business Intelligence node. Warehouse Builder enables you to edit the name, description, and location information of this module using the Edit dialog.

Expand your Business Presentation module node to display the Presentation Templates node. This is where you define your presentation templates, as described in "Creating Presentation Templates" on page 15-29.

Editing Business Presentation Modules

Use the Edit Business Presentation Module dialog to edit a Business Presentation Module definition.

To edit a Business Presentation Module:

1. Expand the Business Presentation Module node in the Project Explorer.
2. Right-click the name of the Business Presentation Module and select **Open Editor**.

   The Edit Business Presentation Module dialog containing the Name tab and the Data Location tab is displayed.

Renaming the Business Presentation Module

Use the Name tab to edit the name and the description of the Business Presentation Module.

Modifying the Data Location

The Data Locations tab displays the data location that is selected for the Business Presentation Module in the Selected Locations column. The Available Locations column lists the available BI Beans locations. To specify a new location for the Business Presentation Module, move the current selection from the Selected Locations column to the Available Locations column using the shuttle arrows. Then select the new location from the Available Location column and move it to the Selected Locations column.

You can create a new location by clicking **New** and specifying the details for the new location.

Creating Presentation Templates

Warehouse Builder provides a wizard to help you create presentation templates to be configured and deployed to BI Beans. You can create simple cross-tab and graphical presentation templates that can be deployed any number of times.

Follow these steps to create a Business Presentation Template in Warehouse Builder.

To create a Business Presentation Template:

1. Expand your Business Presentations module.

2. Right-click **Presentation Templates** and select **New.**

   Warehouse Builder opens the Create Presentation Template Wizard.

3. Follow the wizard steps using the following guidelines.

Naming the Presentation Template

In the Name and Description page, type a name and optional description for the Presentation Template.

Selecting the Type

Select the type of report structure or presentation template you want to define for display in BI Beans. You can choose a Crosstab or Graph structure type.

Selecting the Items to Include

Select the Items you want to display in your presentation template when it is deployed to BI Beans. The **Available** column lists all the cubes available in your Warehouse Builder repository, along with their measures and dimensions. Select the Items you want to include and use the shuttle arrows to move the selected Items to the **Selected** column.

Select **Automatically add the dimensions and dimension roles** to include dimensions and dimension roles that are related to the selected items.

Defining the Layout

You can choose the layout of your presentation template by selecting what items you want to see on what axis. For example, you can drag and drop the Page Edge Items to the x-axis or y-axis to customize your layout.

Reviewing the Summary

In the Summary page, review the settings for your Presentation Template. Click **Back** if you want to make any changes or click **Finish** to finish creating the Presentation Template.

You can locate the Presentation Template on the Project Explorer under the Presentation Templates node. Use the Edit dialog to modify the name, type, and layout of your Presentation Template.

Editing Presentation Templates

Use the Edit Presentation Template dialog to edit a Presentation Template.

To edit a Presentation Template:

1. Expand the Presentation Templates node from the Project Explorer.
2. Right-click the Presentation Template and select **Open Editor**.

   The Edit Presentation Template dialog containing the following tabs is displayed: Name, Type, Items, Layout.

Renaming the Presentation Template

Use the Name tab to edit the name and description of the Presentation Template.

Modifying the Type of Report Template

Use the Type tab to modify the type of presentation template that is used to display the report in BI Beans. The available options are Crosstab and Graph.

Modifying the Report Items

Use the Items tab to modify the items displayed in the presentation template when it is displayed in BI Beans. The items that are currently displayed in BI Beans are listed in the Selected column. To display more items, move these items from the Available column to the Selected column. To remove certain items when the presentation template is displayed, move these items from the Selected column to the Available column using the shuttle arrows.

Modifying the Report Layout

Use the Layout tab to modify the layout of the report. You can change the page edge items for the report.

Configuring Business Intelligence Objects

During the design phase, you create definitions for the business intelligence objects using Warehouse Builder design objects. After you design objects, you can assign physical properties to these design objects by setting configuration parameters.

To configure a business intelligence object, right-click the object in the Project Explorer and select Configure. The Configuration Properties dialog is displayed. Click the object name on the left side of this dialog to display the configuration parameters on the right.

All business intelligence objects have a configuration parameter called **Deployable**. Select Deployable if you want to generate scripts and deploy the business object. Warehouse Builder only generates scripts for objects marked deployable.

The following sections describe additional configuration parameters for different types of business intelligence objects.

Configuration Parameters for Business Definition Modules

You can set the following configuration parameters for a Business Definition module.

Object Matching: Indicates how Warehouse Builder should perform object matching during deployment to Discoverer. When you deploy business definitions, Warehouse Builder first creates an .eex file and then imports this file into the Discoverer EUL.

The options you can select for Object Matching are **By Identifier** or **By Name**. Warehouse Builder uses this setting to check if an object similar to one that is being deployed already exists in the EUL. If a similar object is found, in Create mode the objects are not deployed and in Upgrade mode the objects are refreshed.

MLS Deployment Language: Represents the language used for deployment to Discoverer.

Location: Represents the Discoverer location to which the Business Definition module is deployed.

Configuration Parameters for Item Folders

You can set the following configuration parameters for item folders.

Optimizer Hint: Represents the optimizer hint to be added when the item folder is used in a query. Click the Ellipsis button on this field to specify the optimizer hint.

Location: Represents the location of the database object that the item folder references.

Configuration Parameters for Registered Functions

For registered functions, you can set the following configuration parameters.

Package: Represents the name of the package that contains the registered function.

AUTHID: Specifies the AUTHID option to be used while generating the registered function. The options you can select are None, Current_User, or Definer. The function will be executed with the permissions defined by the AUTHID clause rather than the permissions of the function owner.

Configuration Parameters for Business Presentation Modules

Set the following configuration parameters for business presentation modules.

Default Catalog Folder. Represents the default catalog folder in the BI catalog to which all presentations in the module are deployed. Warehouse Builder uses this catalog folder for presentations that do not specify a catalog folder in their configuration options.

For example, `busn_pres/my_reports`. If you do not specify a folder, the default is the root folder in the BI catalog. Note that the folder you specify must exist before you deploy business presentation objects in this module. If the folder does not exist, deployment fails.

Location: Represents the BI Beans location to which business presentations are deployed.

Configuration Parameters for Presentation Templates

Set the following configuration parameters for presentation templates.

Catalog Folder: Represents the folder in the BI Beans catalog to which the presentation template is deployed. If no value is specified for this parameter, Warehouse Builder deploys this presentation to the folder specified in the Default Catalog Folder configuration property of the presentation module to which the presentation template belongs.

Location: Represents the location of the referenced cube.

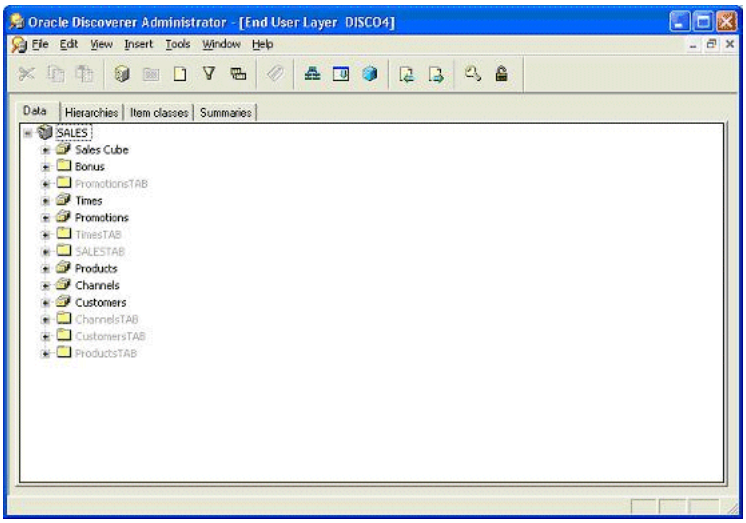
Accessing Business Intelligence Objects Using Oracle BI Discoverer and Oracle BI Beans

Once you successfully deploy the business intelligence objects that you create, these objects are available in Oracle BI Discoverer and Oracle BI Beans. You can use these objects to perform analysis on your warehouse data.

Using Business Definitions in Oracle BI Discoverer

After you deploy the business definitions that you create using Warehouse Builder, these objects are available in the EUL to which they were deployed. You log in to Oracle BI Discoverer Administrator using the user name that you used to deploy the business definitions. The business definitions that you deployed are displayed as shown in Figure 15–4.

Figure 15–4 Discoverer Administrator Showing Business Intelligence Objects

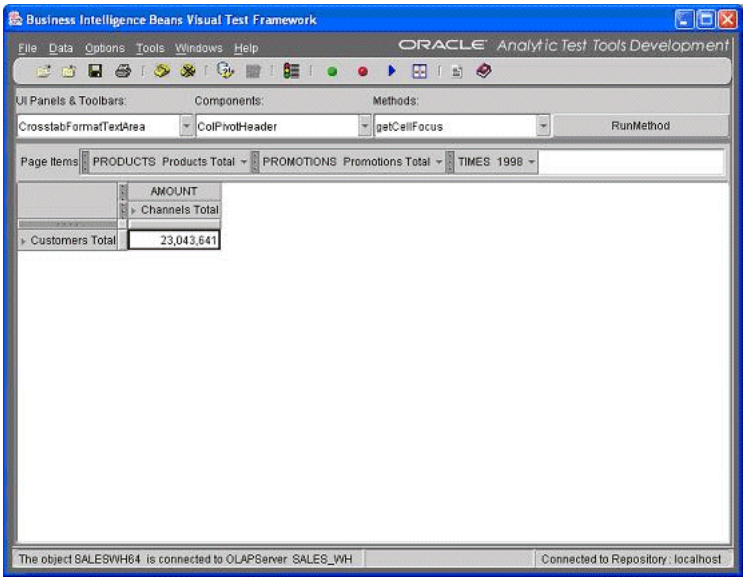


You can now use Oracle BI Discoverer to create reports based on the business intelligence objects that you display.

Using Business Presentations in Oracle BI Beans

After you deploy the business presentations created using Warehouse Builder, you can access them from the BI Beans catalog using applications such as BI Beans VTF (Visual Test Framework) or JDeveloper. The results you see is the presentation you deployed operating on the warehouse data that Warehouse Builder created.

Figure 15–5 Business Presentation Deployed to BI Beans



16

Importing Data Definitions

This chapter shows you how to import definitions from different data sources for use in Warehouse Builder. You first import these definitions into the source modules.

This chapter includes the following topics:

Using the Import Metadata Wizard

Importing is also known as reverse engineering. It saves design time by bringing metadata definitions of existing database objects into Warehouse Builder. You use the Import Metadata Wizard to import metadata definitions into target modules.

The Welcome page of the Import Metadata Wizard lists the steps you follow to import metadata from source applications into the appropriate module. You can also use this wizard to import PL/SQL transformations into a warehouse module.

The Import Metadata Wizard supports importing tables, views, dimensions, cubes, external tables, sequences, queues, user defined types, and PL/SQL transformations directly or through object lookups using synonyms.

Importing a table includes importing its columns, primary keys, unique keys, and foreign keys, which enable import of secondary tables. When you import an external table, Warehouse Builder also imports the associated location and directory information for the associated flat file.

These definitions can be imported from either the Oracle Database catalog or Designer/2000 (Oracle Designer).

Importing Definitions from a Database

Use the Import Metadata Wizard to import metadata from a database into a module. You can import metadata from an Oracle Database, a non-Oracle Database, or a Designer repository.

To import definitions from an Oracle Data Dictionary:

1. Right-click a data source module name and select **Import.**

   The Welcome page of the Import Metadata Wizard is displayed. This page lists the steps to import object metadata. Click **Next** to proceed with the import.

   If you did not specify the location details for the Oracle module, Warehouse Builder displays the Warehouse Builder Warning dialog. This dialog informs you that you must first specify the location details. Click **OK**. The Edit Oracle Database

Location dialog for the Oracle module is displayed. Use this dialog to specify the location information. Clicking **OK** on this dialog displays the Welcome page of Import Metadata Wizard.

2. Complete the following pages:

   Filter Information Page on page 16-2

   Object Selection Page on page 16-2

   Summary and Import Page on page 16-3

   Import Results Page on page 16-3

Filter Information Page

Use the Filter Information page to limit the search of the data dictionary. Use one of the following methods to limit the search:

Selecting the Object Type The Object Type section displays the types of database objects that you can import. This include tables, dimensions, external tables, sequences, queues, cubes, views, PL/SQL transformations, and user-defined types. Select the types of object you want to import. For example, to import three tables and one view, select the **Tables** and **Views** options in the Object Type section.

Search Based on the Object Name Use the **Only select objects that match the pattern** option to type a search pattern. Warehouse Builder searches for objects whose names match the pattern specified. Use % as a wild card match for multiple characters and _ as a wild card match for a single character. For example, you can type a warehouse project name followed by a % to import objects that begin with that project name.

Click **Next** and Warehouse Builder retrieves names that meet the filter conditions from the data dictionary and displays the Object Selection page.

Object Selection Page

Select items to import from the Available list and click the arrow to move them to the Selected list.

To search for specific items by name, click the Find Objects icon that displays as a flashlight.

To move all items to the Selected Objects list, click the double arrow.

Importing Dependent Objects The Import Metadata wizard enables you import the dependent objects of the object being imported. If you are re-importing definitions, previously imported objects appear in bold.

To specify if dependent objects should be included in the import, select one of the following options:

- **None**: Moves only the selected object to the Selected list. No dependencies are imported when you select this option.
- **One Level**: Select this option to move the selected object and the objects it references to the Selected list. This is the default selection.
- **All Levels**: Select this option to move the selected object and all its references directly or indirectly to the Selected list.

Click **Next** and the Summary and Import page is displayed.

Importing Dimensions When you import a dimension that uses a star schema relational implementation, the implementation table that stores the dimension data is not imported. You must explicitly import this table by moving the table from the Available list to the Selected list on the Object Selection page. Also, after the import, you will need to bind the dimension to its implementation table. For more information on how to perform binding, see "Binding" on page 4-27.

Summary and Import Page

This page summarizes your selections in a spreadsheet listing the name, type of object, and whether the object will be reimported or created. Verify the contents of this page and add descriptions for each of the objects.

If the objects you selected on the Object Selection page already exist in the module into which you are attempting to import them, you can specify additional properties related to the reimport. Click **Advanced Import Options** to specify options related to reimporting objects. The Advanced Import Options dialog is displayed. For more information on the contents of this dialog, see "Advanced Import Options" on page 16-5.

Click **Finish** to import the selected objects. The Importing Progress Dialog shows the progress of the import activity. After the import completes, the Import Results page is displayed.

Import Results Page

This page summarizes the import and lists the objects and details about whether the object was created or synchronized.

Click **OK** to accept the changes. To save an MDL file associated with this import, click **Save.** Click **Undo** to cancel the import.

Warehouse Builder stores the definitions in the database module from which you performed the import.

Re-Importing Definitions from an Oracle Database

Re-importing your source database definitions enables you to import changes made to your source metadata since your previous import. You do not have to remove the original definitions from the repository. Warehouse Builder provides you with options that also enable you to preserve any changes you may have made to the definitions since the previous import. This includes any new objects, foreign keys, relationships, and descriptions you may have created in Warehouse Builder.

To re-import definitions:

1. Right-click a data source module name and select **Import.**

   The Welcome page for the Import Metadata Wizard is displayed.

2. Click **Next.**

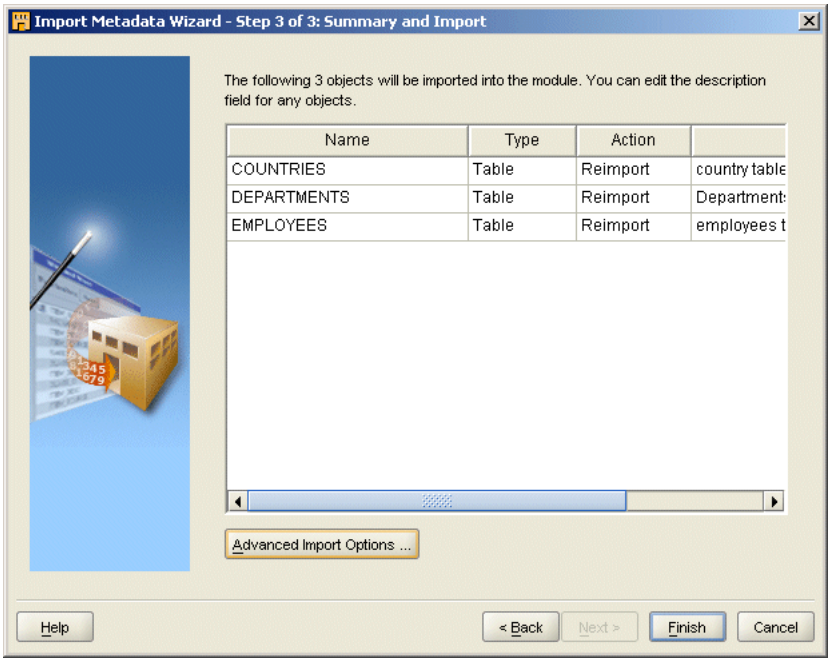
   The Filter Information page is displayed.

3. Complete the Filter Information Page and Object Selection Page, selecting the same settings used in the original import to ensure that the same objects are re-imported.

4. The Summary and Import page displays as shown in Figure 16–1. The Reimport action is displayed for the objects that already exist in the repository or that you are re-importing.

If the source contains new objects related to the object you are re-importing, the wizard requires that you import the new objects at the same time. The Create action displays for these objects.

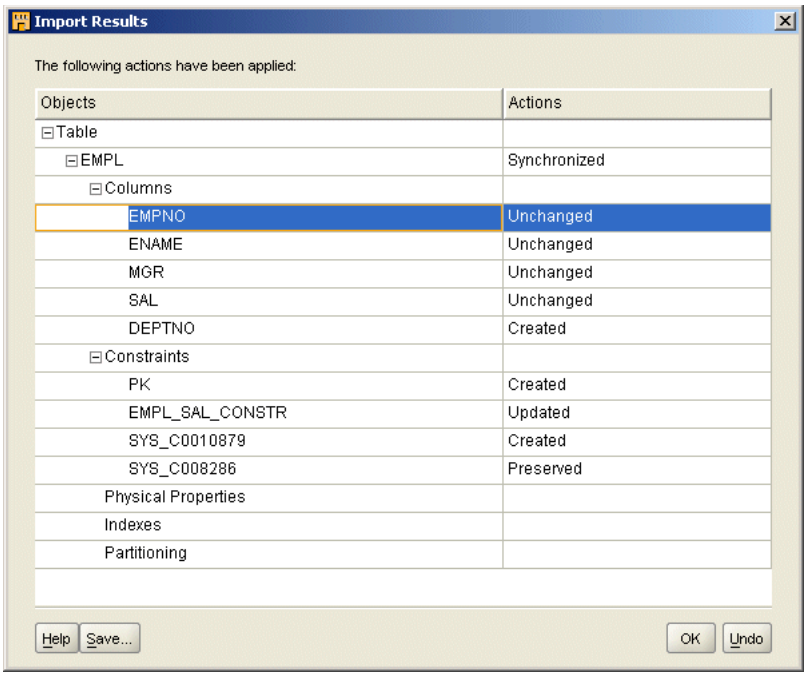
Figure 16–1 Summary and Import Page Showing Reconcile Action



5. Click Advanced Import Options and make selections. (Optional)
6. Click **Finish.**

   Warehouse Builder reconciles and creates objects. When this is complete, the Import Results dialog displays, as shown in Figure 16–2.

Figure 16–2 Import Results Dialog



The report lists the actions performed by Warehouse Builder for each object.

Click **Save** to save the report. You should use a naming convention that is specific to the re-import.

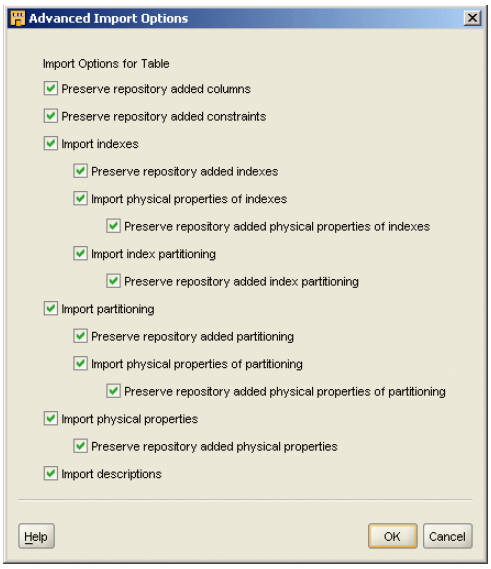
7. Click **OK** to proceed.

   Click **Undo** to undo all changes to your repository.

Advanced Import Options

The Advanced Import Options dialog displays the options that you can configure while importing objects. This dialog enables you to preserve any edits and additions made to the object definitions in the Warehouse Builder repository. Figure 16–3 displays the Advanced Import Options dialog for tables.

Figure 16–3 Advanced Import Options Dialog for Tables



By default, all options on this dialog are checked. Clear boxes to have these repository objects replaced and not preserved.

For example, after importing tables or views for the first time, you manually add descriptions to the table or view definitions. If you want to make sure that these descriptions are not overwritten while reimporting the table or view definitions, you must select the Preserve Existing Definitions option. This ensures that your descriptions are not overwritten.

The contents of this dialog depend on the type of objects being imported. For more information about the advanced import options for each type of objects, refer to the following sections:

- Advanced Import Options for Views and External Tables
- Advanced Import Options for Tables
- Advanced Import Options for Object Types
- Advanced Import Options for Queue Tables, Advanced Queues, Streams Queues, and SQL Collections (Nested Tables and Varrays)

Advanced Import Options for Views and External Tables Select these options for reconciling views or external tables:

- **Import descriptions:** The descriptions of the view or external table are imported. Existing descriptions in the repository are not preserved.
- **Preserve repository added columns:** The columns you added to the object in the repository are preserved.

Advanced Import Options for Tables Select these options for reconciling tables:

- **Preserve repository added columns:** Select this option to retain any columns added the table in the repository.
- **Preserve repository added constraints:** The constraints you added to the table in Warehouse Builder are preserved.
- **Import indexes:** Select this option to specify additional details about how indexes should be imported. Importing indexes consist of the following options:

  - **Preserve repository added indexes:** Select this option to retain any indexes added to the repository table.
  - **Import physical properties of indexes:** Select this option to indicate how indexes should be imported. Select the **Preserve repository added physical properties of indexes** option below this option to specify that any physical properties added to the indexes should be preserved.
  - **Import index partitioning:** Select this option to indicate how index partitions should be imported. Select the **Preserve repository added index partitioning** option to specify that any index partitions added to the repository table must be preserved.

- **Import Partitioning:** Select this option to specify additional details about how partitions should be imported. Importing partitions contains the following options:
  - **Preserve repository added partitioning:** Select this option to retain all partitions added to the repository table.
  - **Import physical properties of partitioning:** Use this option to indicate how the physical properties of partitions should be imported. Select **Preserve repository added physical properties of partitioning** to indicate that all physical properties of the partitions in the repository table should be retained.
- **Import physical properties:** Select this option to indicate how the physical properties of the table should be imported. Select the **Preserve repository added physical properties** option to specify that all physical properties added to the repository table must be preserved.
- **Import descriptions:** Select this option to import the descriptions of the table.

Advanced Import Options for Object Types

- **Import descriptions:** Select this option to import the descriptions of the object type.
- **Preserve repository added attributes:** Select this option to retain the attributes added to the object type in the repository.

Advanced Import Options for Queue Tables, Advanced Queues, Streams Queues, and SQL Collections (Nested Tables and Varrays)

- **Import descriptions:** Select this option to import the descriptions of the queue table, advanced queue, streams queue, nested table, or Varray.

Updating Oracle Database Source Definitions

The Edit Module dialog enables you to edit the name, metadata location, and the data location of a source module.

To update the database definitions:

1. Double-click any Oracle module.

   The Edit Module dialog displays. You can edit the metadata location as well as the data location of the database.

2. To edit the metadata location, click the Metadata Location tab and specify the following:
   - **Source Type**: The source type identifies the location of the data and the metadata. It can be either Oracle Data Dictionary or Oracle Designer

Repository. Select Oracle Data Dictionary if the metadata is stored in the default repository of the Oracle Database. Select Oracle Designer Repository if the metadata is stored in an Oracle Designer repository.

- **Location**: Identifies the location of the module. You can select a location from the drop-down list.

3. To edit the data location, click the Data Location tab. You can either select from the existing locations or create a new location. To create a new location, click **New**. The Edit Oracle Database Location window displays. You can specify the data location here.

Using Oracle Designer 6*i*/9*i* Sources

In Warehouse Builder, you can create a source module that connects with an Oracle Designer repository. When the definitions for an application are stored and managed in an Oracle Designer repository, the time required to connect to the application is reduced.

Designer 6*i*/9*i* repositories use workareas to control versions of an object. By selecting a workarea, you can specify a version of a repository object. With Designer 6*i*/9*i*, you can also group objects into container elements within workareas. Container Element contains definitions for namespace and ownership of objects and enables you to view objects even though they are owned by a different user. Because Designer 6*i*/9*i* container elements are controlled by workareas, they have version control. See the Designer 6*i*/9*i* documentation for more information about workareas and container elements.

All visible objects of a workarea or a container element in Designer 6*i*/9*i* are available for use as data sources in Warehouse Builder. To select Designer 6*i*/9*i* objects as Warehouse Builder sources:

- Specify a workarea and
- Specify the container element in the workarea

The Module Editor detects the Designer version. If it finds Designer 6*i*/9*i*, the Metadata Location tab shows two drop-down lists, Workarea and Container Element. When you select a workarea, the Container Element list will show the container elements in that workarea.

The list of repository objects available for import is determined by the following criteria:

- The object type must be supported by Warehouse Builder (Table, View, Sequence, and Synonyms).
- The object must be accessible in the specified workarea. This determines the version of objects accessed.
- The object must be visible in the specified container element. The list displays objects owned by the specified container element and other objects shared by the specified container element, but not owned by it.

To import definitions from a Designer 6*i*/9*i* source, you must follow the steps outlined in Importing Definitions from a Database on page 16-1.

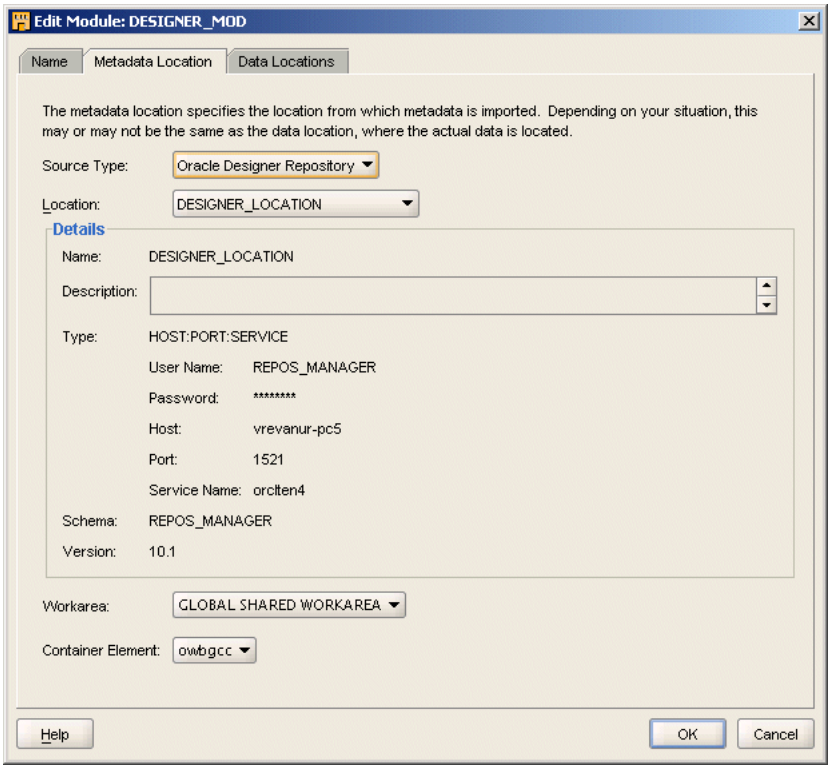
Using Designer 6*i*/9*i* as a Metadata Source

To create a Designer 6i/9i source module:

1. Create a database source module.
2. Double-click the name of the newly created module to open the Module Editor.
3. In the Metadata Location tab, select the source type as Oracle Designer Repository. Also select the database location containing the Designer object.

   When you select the source type as Oracle Designer Repository, two new drop-down lists, **Workarea** and **Container Element**, are visible in the Metadata Location tab as shown in Figure 16–4.
4. Select the Designer *6i/9i* object from the workarea and select the specific container element.

Figure 16–4 The Metadata Location Tab



> **Note:** The database you specify as source must contain a Designer *6i/9i* object. If not, then the Workarea and Element Container lists will be empty.

5. Click **OK**.

For related information, see the following sections:

- Importing Definitions from a Database on page 16-1
- Re-Importing Definitions from an Oracle Database on page 16-3
- Updating Oracle Database Source Definitions on page 16-7

17

Integrating Metadata Through the Warehouse Builder Transfer Wizard

Oracle Warehouse Builder provides several utilities for sharing its metadata with other tools and systems. This chapter first shows you how to create collections that store definitions for a group of Warehouse Builder objects. It also discusses importing metadata stored in Object Management Group (OMG) Common Warehouse Model (CWM) formats using the Oracle Warehouse Builder Transfer Wizard.

This chapter includes the following topics:

Using the Oracle Warehouse Builder Transfer Wizard

The Oracle Warehouse Builder Transfer Wizard enables you to synchronize, integrate, and use metadata stored in OMG CWM formats. Use the Oracle Warehouse Builder Transfer Wizard to import selected metadata from OMG CWM compliant applications version 1.0 into the Warehouse Builder repository. You can also import metadata that is stored in a proprietary format by integrating with MIMB (Meta Integration Model Bridges). For more information on the formats that MIMB can translate, see "Integrating with the Meta Integration Model Bridges (MIMB)" on page 17-1.

The Oracle Warehouse Builder Transfer Wizard can be used only in single-user mode. This means that no other users can be connected to the repository into which you are importing OMG CWM metadata. Before you use the Oracle Warehouse Builder Transfer Wizard to import OMG CWM metadata into a repository, ensure that you are the only user connected to that repository.

Integrating with the Meta Integration Model Bridges (MIMB)

Warehouse Builder enables you to integrate with Meta Integration Model Bridges (MIMB) that translate metadata from a proprietary metadata file or repository to the standard CWM format. You can then import this metadata into Warehouse Builder using the Oracle Warehouse Builder Transfer Wizard. After integrating with MIMB, you can also import metadata from CA ERwin, Sybase PowerDesigner, and many other sources through these bridges.

MIMB integration enables you to import metadata into Warehouse Builder from the following sources:

- OMG CWM 1.0: Database Schema using JDBC 1.0/2.0
- OMG CWM 1.0: Meta Integration Repository 3.0

- OMG CWM 1.0: Meta Integration Works 3.0
- OMG CWM 1.0: XML DTD 1.0 (W3C)
- OMG CWM 1.0: XML DTD for HL7 3.0
- Acta Works 5.x
- Adaptive Repository
- ArgoUML
- Business Objects Crystal Reports
- Business Objects Data Integrator
- Business Objects Designer
- CA COOL
- CA ERwin
- CA ParadigmPlus
- Cognos Impromptu 7.1
- Cognos ReportNet Framework Manager
- Hyperion Application Builder
- IBM
- Informatica PowerCenter
- Merant App Master Designer 4.0
- Microsoft SQL Server
- Microsoft Visio
- Microsoft Visual Studio
- Microsoft Yukon Database Source View Designer
- MicroStrategy
- NCR Teradata
- Oracle Designer
- Popkin System Architect
- ProActivity 3.x & 4.0
- Rational Rose
- SAS ETL Studio
- Select SE 7.0
- Silverrun
- Sybase Power Designer
- Unisys Rose
- Visible IE Advantage 6.1
- Various XML/XMI versions

Note: The list of sources is not exhaustive and does not include the product versions. For a list of all the supported sources, refer to the MIMB documentation at *http://www.metaintegration.net/Products/MIMB/SupportedTools.html*.

Follow these steps to integrate Warehouse Builder with MIMB.

Download the Meta Integration Model Bridge

After you download the MIMB on your system, Warehouse Builder automatically recognizes the installation and displays the new import options on the Metadata Source and Target Identification Page of the Oracle Warehouse Builder Transfer Wizard.

To download MIMB:

1. Download the Model Bridge (personal) product from the following Web site: *http://www.metaintegration.net/Products/Downloads/*
2. Install the MIMB by running the setup on your system.
3. During installation, choose **Typical with Java Extensions** as the installation type from the Setup Type page.

   If the set up program is not able to find a JDK on your machine, you must provide the JNI library directory path name. Your path environment variable must contain the metaintegration directory. If not, you need to add it to the path: *c:\program files\metaintegration\win32.*
4. Enable MIMB through your Warehouse Builder client by starting the Oracle Warehouse Builder Transfer Wizard. For more information on importing metadata using this wizard, see "Importing Metadata into Warehouse Builder" on page 17-3.

Importing Metadata into Warehouse Builder

The Transfer Wizard enables you to import metadata from OMG CWM metadata into Warehouse Builder. Perform the following steps to import metadata:

1. Run the Oracle Warehouse Builder Transfer Wizard.

   The Oracle Warehouse Builder Transfer Wizard converts the OMG CWM compliant metadata and stores it in an MDL file. For more details on using the Oracle Warehouse Builder Transfer Wizard, see "Running the Oracle Warehouse Builder Transfer Wizard" on page 17-3.
2. Import the MDL file created in Step 1 into Warehouse Builder using the Metadata Loader Import Utility.

   For more information importing the MDL file, see "Importing the MDL File into Warehouse Builder" on page 17-6.

Running the Oracle Warehouse Builder Transfer Wizard

From the **Design** menu, select **Import**, and then **Bridges**. Warehouse Builder displays the Welcome page of the Oracle Warehouse Builder Transfer Wizard. This page lists the steps you perform while using the Oracle Warehouse Builder Transfer Wizard. If you want to display version information about the wizard, click **About Oracle OWB Transfer Tool**. For version information about the individual bridges, press the **Bridge Versions** button from the About Oracle OWB Transfer Tool dialog. Click **Next** to proceed.

Provide details on the following pages of the Oracle Warehouse Builder Transfer Wizard:

- Metadata Source and Target Identification Page
- Transfer Parameter Identification Page
- Summary Page

Metadata Source and Target Identification Page

Use this page to specify the format of the source data.

The **From** field automatically identifies your metadata source as OMG CWM 1.0. You can import metadata from many other sources if you choose to integrate with the Meta Integration Model Bridge (MIMB). When you integrate with MIMB, the From field displays a list of other sources from which you can import metadata. For details, see "Integrating with the Meta Integration Model Bridges (MIMB)" on page 17-1.

The **To** field displays OWB mdl. You cannot change this value. The Oracle Warehouse Builder Transfer Wizard converts the source metadata into an mdl file that can processed by the Metadata Loader Utility.

Use the **Description** field to enter an optional description of the metadata to be transferred. This description displays in the progress bar during the transfer process.

Transfer Parameter Identification Page

This page lists the parameters used during the transfer process. Table 17–1 describes the transfer parameters used by the Oracle Warehouse Builder Transfer Wizard.

Table 17–1 Transfer Parameters for OMG CWM File Import

Transfer Parameter Name	Description
OMG CWM Input File	Specify the file containing the OMG CWM model you want to import. Click the Ellipsis button to browse for the file location.
OWB Project	Specify the name of the Warehouse Builder project into which the model will be imported.
Warehouse Builder MDL	Specify the name of the MDL file that will store the converted source metadata.
Default Module Type	Choose the type of module into which you want to import the metadata. The options are **Source warehouse module** and **Target warehouse module**. The Oracle Warehouse Builder Transfer Wizard will create the correct module and import the metadata into it. When you choose the Source warehouse module option, the names are case-sensitive. When you choose Target warehouse module, all the object names are converted to uppercase.
Process OLAP Physical Representation	If you are importing metadata from a star schema, then you can choose True if you want to maintain its physical representation or False to only import its logical definitions. If you are importing a snowflake schema, its physical implementation is lost and only the logical definitions are imported into Warehouse Builder.

Table 17–1 (Cont.) Transfer Parameters for OMG CWM File Import

Transfer Parameter Name	Description
Log Level	Log level to be assigned to the transfer: Errors, Information, and Trace. **Errors:** lists the errors, if any, generated during the transfer. **Information:** lists the transfer details about the metadata objects, including any errors. **Trace:** produces a trace for debugging purposes, which also includes the errors and information log details. If you need assistance with the trace, contact Oracle World Wide Support. Trace type provides the most detailed log information.

When you integrate with MIMB, the Transfer Parameter Identification page displays additional parameters. The parameters displayed depend on the source you choose in From field of the Metadata Source and Target Identification page. The descriptions of these parameters display in the box at the bottom of the wizard page.

Summary Page

The Summary page displays the values that you selected for each field. Review your entries. If any entries are incorrect, click **Back** to return to the previous screen and make the necessary changes. Click **Finish** to transfer the metadata. The Oracle WB Transfer dialog is displayed. For details about this dialog, see "Oracle WB Transfer Dialog" on page 17-5.

Oracle WB Transfer Dialog This dialog indicates the progress of the metadata transfer. The description you provide on the Metadata Source and Target Identification Page is appended to the title of the transfer window. For example, if you entered the description as My Metadata Transfer, the title of this dialog will be Oracle WB Transfer - My Metadata Transfer. If you did not provide a description, a title does not display.

The transfer can require several minutes or an hour or more to complete, depending on the amount of metadata you transfer.

Do one of the following:

- If the transfer completes successfully (100% displays), click **OK.** The Oracle Warehouse Builder Transfer Wizard displays a reminder informing you that you must import the MDL file into the repository.
- If a transfer failure message displays, click **View Log File** and review the log. (You can also view the log of a successful transfer.)

  To save the log for reference, click **Save As** to open the Save dialog. Select the folder where you want to store the log and click **Save.**
- If you determine the cause of the failure from the Information Log, note the information requiring update. Close the log by clicking **OK.** On the Oracle WB Transfer dialog, click **Return to Wizard** and update the incorrect information on the Transfer Parameter Identification Page. Then transfer the data again.
- If you cannot determine the cause of the failure from the Information Log, you can create a Trace log. Close the current log by clicking **OK.** On the Oracle WB Transfer dialog, click **Return to Wizard** and change the Log Level to **Trace** on the Transfer Parameter Identification Page. Then transfer the data again.

If the transfer is successful, the Oracle Warehouse Builder Transfer Wizard creates an MDL file and stores it in the location you specified in the **Warehouse Builder MDL**

parameter on the Transfer Parameter Identification Page. The version of the MDL file created by the Oracle Warehouse Builder Transfer Wizard is 10.1.0.3.

Importing the MDL File into Warehouse Builder

After the Oracle Warehouse Builder Transfer Wizard converts the source metadata and stores it in an MDL file, you must import this MDL file into the Warehouse Builder repository. The Metadata Import Utility enables you to import an MDL file into Warehouse Builder. For more information about the Metadata Import Utility, see Chapter 33, "Importing and Exporting with the Metadata Loader (MDL)".

Use the following steps:

1. From the **Design** menu, select **Import**, and then **Warehouse Builder Metadata**.

   The Metadata Import dialog is displayed.

2. In the **File Name** field, specify the name of the MDL file created using the Oracle Warehouse Builder Transfer Wizard. You can also use the **Browse** button to select the MDL file.

3. Click **Import**.

   The Metadata Upgrade dialog is displayed.

4. Click **Upgrade**.

   The Metadata Import Progress dialog is displayed. The text above the progress bar displays the status of the import.

   If you want to display a detailed log of the import process, click **Show Details**. The Message Log panel displays the details. Click **Show Statistics** for information about the number and type of objects imported.

5. Click **Close**.

Transfer Considerations

Before you transfer metadata into the Warehouse Builder repository, you need to perform tasks within the source and target tools to ensure that the metadata can be transferred successfully.

This section lists the objects that are extracted from the source tool during the transfer and their corresponding Warehouse Builder objects.

Importing Metadata from an Object Management Group CWM Standard System

An OMG CWM standard system could have more than one implementations of a cube. In such cases, the Oracle Warehouse Builder Transfer Wizard imports the first implementation of the cube. The log file of the import details the version of the cube implementation that is imported.

Table 17–2 lists the object conversion from an OMG CWM file system to the Warehouse Builder repository.

Table 17–2 OMG CWM Object Conversion

Object in OMG CWM	Object Imported to Warehouse Builder
Package	Project
Schema	Module
Dimension	Dimension

Table 17–2 (Cont.) OMG CWM Object Conversion

Object in OMG CWM	Object Imported to Warehouse Builder
Level	Level
Attribute	Level Attributes
Hierarchy	Hierarchy
Cube	Cube
Measure	Cube Measure
Table	Table
Column	Column
Foreign Key	Foreign Key
Unique Constraint/Primary Key	Unique Key
View	View
Column	Column

Table 17–3 lists the data type conversion from an OMG CWM system to a Warehouse Builder repository.

Table 17–3 OMG CWM Data Type Conversions

Data type in OMG CWM	Data type Imported to Warehouse Builder
VARCHAR	VARCHAR2
NCHAR VARYING	NVARCHAR2
LONG, ROWID, UROWID	VARCHAR2
NUMERIC, SMALLINT	NUMBER
BFILE	BLOB
LONG RAW	RAW
TIME	TIMESTAMP
TIME WITH TIMEZONE, TIMESTAMP WITH TIMEZONE	TIMESTAMP WITH LOCAL TIMEZONE
BOOLEAN	CHAR
REAL, DOUCLE PRECISION, BINARY FLOAT, BINARY DOUBLE	FLOAT
XMLTYPE, SYS.ANYDATA	UNKNOWN
INTERVAL	INTERVAL DAY TO SECOND

18

Importing Data From Third Party Applications

Warehouse Builder enables you to interpret and extract metadata from custom and packaged applications and databases into its repository.

This chapter contains the following topics:

Integrating with E-Business Suite

Before You Begin

Contact the database administrator for the E-Business Suite database and request a user name and password for accessing the APPS schema. The DBA may have previously created a user by running the script `owbebs.sql` as described in the *Oracle Warehouse Builder Installation and Administration Guide*. If not, you will need to provide the DBA with a list of the tables, views, sequences, and keys from which you plan to extract data.

Depending on the preference of the DBA, there may be a single user who extracts both, the metadata as well as the data. Or, there may be two separate users: one able to access metadata and another able to access data.

Creating an E-Business Suite Module

You can create a module in Warehouse Builder to store data from the E-Business Suite database. The Create Module wizard enables you to create a new module.

To create a new E-Business Suite source module:

1. From the Project Explorer, expand the Applications node.
2. Right-click the ORACLE_EBUSINESS_SUITE node and select **New.**

   Warehouse Builder displays the Welcome page for the Create Module Wizard.
3. Click **Next.**

   The wizard displays the Name and Description page.
4. Provide the following information in this page:

Name of the module: Type a unique name for the module between 1 and 30 alphanumeric characters. Spaces are not allowed.

Status of the module: Select a status for the module from the drop-down list: Development, Quality Assurance, Production.

Selecting one of these options can help you document the warehouse design version.

Description: Type a description of the module you are creating (Optional).

5. Click **Next.**

   The wizard displays the Connection Information page, which contains the following fields:

   Source Type: Specify the data source. The data source can either be an Oracle Database or it can be an external source accessed using a gateway.

   Location: Specify the location of the data source. Click **Edit** to open the Edit Location dialog. Use this dialog to specify the connection details of the E-Business Suite database. For more information, see Connecting to an E-Business Suite Database.

   Directory: Specify the directory where Oracle Gateway can access the source data, if the source is not an Oracle Database.

   Import after finish: Select this option if you wish to import the data immediately after creating the module.

   > **Note:** If you do not wish to import after finish, then you do not have to necessarily provide the connection details while creating an E-Business Suite database module. In this case, you will have to specify the connection details when you attempt to import data into the module.

6. Click **Next**.

   The wizard displays the Summary page, which provides details of the values you entered in the previous pages.

7. Click **Finish**.

   The wizard creates and inserts the new module under the ORACLE_EBUSINESS_SUITE node in the Project Explorer.

Connecting to an E-Business Suite Database

1. Specify the connection type. Based on the connection type, you will need to provide the connection details. The connection type can be one of the following:

   HOST:PORT:SERVICE: Makes a connection using the Easy Connect Naming method, which requires no prior setup. For more information on the Easy Connect Naming method, refer the *Oracle Database Net Services Administrator's Guide*. Provide the following connection details:

   - **User Name**: The database user credential that has permission to access the schema location. When connecting to a database that does not have user names, enter any text as a mock user name and password.

- **Host**: The name of the system where the database is installed. If Warehouse Builder client is installed on the same system as an Oracle Database, you can enter *localhost* instead of the system name.
- **Port**: The SQL port number for the database.
- **Service Name**: The service name of the database.
- **Use Global Name**: The unique name of the database, which is composed of the database name and the domain in the form *database_name.database_domain*. Select this option if the database is accessed through a network connection.

Database Link: A database link is a schema object that contains information for connecting to a remote database. Database links are used in distributed database environments and enable a client, such as Warehouse Builder, to access two physical databases as one logical database. Select this method only when you do not have privileges that enable you to make a direct connection. You cannot deploy to a location that uses a database link. A database link is not available for BI Beans or Discoverer locations. Provide the following connection details:

- **From Location**: An existing location where the database link is defined.
- **Database Link**: The object name of the database link.

SQL*Net Connection: Makes a connection using a net service name previously defined using a tool, such as Oracle Net Configuration Assistant. The net service name provides a convenient alias for the connection information. This method of connecting is ideal for RAC installations. Provide the following connection details:

- **User Name**: The database user credential that has permission to access the schema location. When connecting to a database that does not have user names, enter any text as a mock user name and password.
- **Net Service Name**: The name of the predefined connection.
- **Use Global Name**: The unique name of the database, which is composed of the database name and the domain in the form *database_name.database_domain*. Select this option if the database is accessed through a network connection.

2. Specify the schema where the source data is stored or the target objects will be deployed. The schema must be registered with Warehouse Builder. By default, it is the *User Name* schema.
3. Specify the version number of the Oracle Database. This is not required for non-Oracle Database locations.
4. Click **Test Connection** to verify that the connection information you provided are correct.
5. Click **OK** to go back to the Connection Information page of the Create Module wizard.

Importing E-Business Suite Metadata Definitions

After creating the E-Business Suite source module, you can import metadata definitions from E-Business Suite objects using the Import Metadata Wizard. This wizard enables you to filter the E-Business Suite objects you want to import and verify those objects. You can import metadata for tables, views, and sequences.

Perform the following steps to import E-Business Suite metadata:

1. From the Warehouse Builder Project Explorer, expand the **Applications** node and then the **ORACLE_EBUSINESS_SUITE** node.

2. Right-click the E-Business Suite source module into which you want to import metadata and select **Import** from the pop-up menu.

   Warehouse Builder displays the welcome page for the Import Metadata Wizard.

3. Click **Next.**

4. Complete the following tasks:

   Filtering E-Business Suite Metadata

   Selecting the Objects

   Reviewing Import Summary

Filtering E-Business Suite Metadata

The Import Metadata Wizard includes a Filter Information page that enables you to select the metadata. Warehouse Builder provides two filtering methods:

- Business Domain

  This filter enables you to browse E-Business Suite business domains to locate the metadata you want to import. You can view a list of objects contained in the business domain and the names of the objects in the E-Business Suite application. For more information, see Filtering E-Business Suite Metadata by Business Domain.

- Text String Matching

  This filter enables you to search tables, views, and sequences by typing text string information in the field provided in the Filter Information page. This is a more specific search method if you are familiar with the contents of your E-Business Suite application database. For more information, see Filtering E-Business Suite Metadata by Text String.

Choose a filtering method and click **Next** to proceed with the importing of metadata.

Filtering E-Business Suite Metadata by Business Domain

1. Select **Business Domain** and click **Browse** to open the Business Component Hierarchy dialog.

2. The Business Component Hierarchy dialog lists the available E-Business Suite business domains.

   Note: It may take two to ten minutes to list the business domains depending on the network location of the E-Business Suite application server, the type of LAN used, or the size of the E-Business Suite application database.

   Use the Business Component Hierarchy dialog to select the E-Business Suite business domains that contain the metadata objects you want to import.

3. Select a business domain and click **Show Entities**.

   The Folder dialog displays a list of objects available in the selected business domain.

4. Review this dialog to ensure that you are selecting an appropriate number of objects and click **OK** to go back to the Business Component Hierarchy dialog.

Some business domains can contain more than 1000 objects. Importing such a large amount of metadata can take from one to three hours or more, depending on the network connection speed and the processing power of the source and target systems.

5. Click **OK.**

   The wizard displays the Filter Information page with the E-Business Suite business domain displayed in the Business Domain field.

Filtering E-Business Suite Metadata by Text String

1. Select **Text String, where object**.
2. Select the objects you wish to import. You can select Tables, Views, and Sequences.

   If you wish to select specific objects, type the object name in the text field. Create a filter for object selection by using the wildcard characters (%) for zero or more matching characters, and (_) for a single matching character.

   For example, if you want to search the business domain for tables whose names contain the word CURRENCY, then type `%CURRENCY%`. If you want to refine the search to include only tables named CURRENCY and followed by a single digit, then type `%CURRENCY_`.

Selecting the Objects

The Object Selection page contains a description of the objects and enables you to select the objects you want to import into the E-Business Suite module. To select the objects:

1. Move the objects from the available list to the selected list.

   The Import Wizard also enables you to choose whether you want to import tables with foreign key relationships for each object that you choose to import. You can select one of the following:

   None: Import only the objects in the Selected list.

   One Level: Import the objects in the Selected list and any tables linked to it directly through a foreign key relationship.

   All Levels: Import the objects in the Selected list and all tables linked to it through foreign key relationships.

   The foreign key level you select is the same for all tables selected for importing.

   > **Note:** Selecting All Levels increases the time it takes to import the metadata because you are directing the wizard to import tables that are related to each other through foreign key constraints. Select this option only if it is necessary.

2. Click **Next.**

   If you select One Level or All Levels, the Confirm Import Selection dialog appears.

   Review this dialog to ensure that you are selecting an appropriate number of tables.

3. Click **OK.**

   The selected objects appear in the right pane of the Object Selection page.

4. Click **Next.**

   The wizard displays the Summary and Import page.

Reviewing Import Summary

The wizard imports definitions for the selected objects from the E-Business Suite Application Server, stores them in the E-Business Suite source module, and then displays the Summary and Import page.

You can edit the descriptions for each object by selecting the description field and typing a new description.

Review the information on the Summary and Import page and click **Finish.**

The E-Business Suite integrator reads the table definitions from the E-Business Suite application server and creates the metadata objects in the Warehouse Builder repository.

The time it takes to import the E-Business Suite metadata to the Warehouse Builder repository depends on the size and number of tables and the connection between the E-Business Suite application server and the repository. Importing 500 or more objects could take one to three hours or more, especially if you are connecting servers in separate LANs.

When the Import completes, the Import Results dialog displays. Click **OK** to finish importing.

Integrating with PeopleSoft Data

PeopleSoft applications provide ERP solutions. A PeopleSoft application consists of numerous modules, each pertaining to a specific area in an enterprise, such as Human Resource Management System (HRMS), Financials, and Material Management.

Creating a PeopleSoft Module

You can create a module in Warehouse Builder to store data from a PeopleSoft database. The Create Module wizard enables you to create a new module.

To create a new PeopleSoft source module:

1. From the Project Explorer, expand the Applications node.
2. Right-click the PEOPLESOFT8_9 node and select **New.**

   Warehouse Builder displays the Welcome page for the Create Module wizard.
3. Click **Next.**

   The wizard displays the Name and Description page.
4. Provide the following information in this page:

   Name of the module: Type a unique name for the module between 1 and 30 alphanumeric characters. Spaces are not allowed.

   Status of the module: Select a status for the module from the drop-down list: Development, Quality Assurance, Production.

   Selecting one of these options can help you document the warehouse design version.

   Description: Type a description of the module you are creating (Optional).

5. Click **Next.**

   The wizard displays the Connection Information page, which contains the following fields:

   Source Type: Specify the data source. The data source can either be an Oracle Database or it can be an external source accessed using a gateway.

   Location: Specify the location of the data source. Click **Edit** to open the Edit Location dialog. Use this dialog to specify the connection details of the PeopleSoft database. For more information, see Connecting to PeopleSoft Database.

   Directory: Specify the directory where Oracle Gateway can access the source data, if the source is not an Oracle Database.

   Import after finish: Select this option if you wish to import the data immediately after creating the module.

   > **Note:** If you do not wish to import after finish, then you do not have to necessarily provide the connection details while creating a PeopleSoft database module. In this case, you will have to specify the connection details when you attempt to import data into the module.

6. Click **Next**.

   The wizard displays the Summary page, which provides details of the values you entered in the previous pages.

7. Click **Finish**.

   The wizard creates and inserts the new module under the PEOPLESOFT8_9 node in the Project Explorer.

Connecting to PeopleSoft Database

1. Specify the connection type. Based on the connection type, you will need to provide the connection details. The connection type can be one of the following:

   HOST:PORT:SERVICE: Makes a connection using the Easy Connect Naming method, which requires no prior setup. For more information on the Easy Connect Naming method, refer the *Oracle Database Net Services Administrator's Guide*. Provide the following connection details:

   - **User Name**: The database user credential that has permission to access the schema location. When connecting to a database that does not have user names, enter any text as a mock user name and password.
   - **Host**: The name of the system where the database is installed. If Warehouse Builder client is installed on the same system as an Oracle Database, you can enter *localhost* instead of the system name.
   - **Port**: The SQL port number for the database.
   - **Service Name**: The service name of the database.
   - **Use Global Name**: The unique name of the database, which is composed of the database name and the domain in the form *database_name.database_domain*. Select this option if the database is accessed through a network connection.

   Database Link: A database link is a schema object that contains information for connecting to a remote database. Database links are used in distributed database environments and enable a client, such as Warehouse Builder, to access two

physical databases as one logical database. Select this method only when you do not have privileges that enable you to make a direct connection. You cannot deploy to a location that uses a database link. A database link is not available for BI Beans or Discoverer locations. Provide the following connection details:

- **From Location**: An existing location where the database link is defined.
- **Database Link**: The object name of the database link.

SQL*Net Connection: Makes a connection using a net service name previously defined using a tool, such as Oracle Net Configuration Assistant. The net service name provides a convenient alias for the connection information. This method of connecting is ideal for RAC installations. Provide the following connection details:

- **User Name**: The database user credential that has permission to access the schema location. When connecting to a database that does not have user names, enter any text as a mock user name and password.
- **Net Service Name**: The name of the predefined connection.
- **Use Global Name**: The unique name of the database, which is composed of the database name and the domain in the form *database_name.database_domain*. Select this option if the database is accessed through a network connection.

2. Specify the schema where the source data is stored or the target objects will be deployed. The schema must be registered with Warehouse Builder. By default, it is the *User Name* schema.
3. Specify the version number of the Oracle Database. This is not required for non-Oracle Database locations.
4. Click **Test Connection** to verify that the connection information you provided are correct.
5. Click **OK** to go back to the Connection Information page of the Create Module wizard.

Importing PeopleSoft Metadata Definitions

After creating the PeopleSoft source module, you can import metadata definitions from PeopleSoft objects using the Import Metadata Wizard. This wizard enables you to filter the PeopleSoft objects you want to import and verify those objects. You can import metadata for tables, views, and sequences.

Perform the following steps to import PeopleSoft metadata:

1. From the Warehouse Builder Project Explorer, expand the **Applications** node and then the **PEOPLESOFT8_9** node.
2. Right-click the PeopleSoft source module into which you want to import metadata and select **Import** from the pop-up menu.

   Warehouse Builder displays the welcome page for the Import Metadata Wizard.
3. Click **Next.**
4. Complete the following tasks:

   Filtering PeopleSoft Metadata

   Selecting the Objects

   Reviewing Import Summary

Filtering PeopleSoft Metadata

The Import Metadata Wizard includes a Filter Information page that enables you to select the metadata. Warehouse Builder provides two filtering methods:

- Business Domain

  This filter enables you to browse PeopleSoft business domains to locate the metadata you want to import. You can view a list of objects contained in the business domain. For more information, see Filtering PeopleSoft Metadata by Business Domain.

- Text String Matching

  This filter enables you to search tables, views, and sequences by typing text string information in the field provided in the Filter Information page. This is a more specific search method if you are familiar with the contents of your PeopleSoft application database. For more information, see Filtering PeopleSoft Metadata by Text String.

Choose a filtering method and click **Next** to proceed with the importing of metadata.

Filtering PeopleSoft Metadata by Business Domain

1. Select **Business Domain** and click **Browse** to open the Business Component Hierarchy dialog.

   The Import Metadata Wizard displays Loading Progress Dialog while it is retrieving the business domains.

2. The Business Component Hierarchy dialog lists the available PeopleSoft business domains.

   Note: It may take two to ten minutes to list the business domains depending on the network location of the PeopleSoft application server, the type of LAN used, or the size of the PeopleSoft application database.

   Use the Business Component Hierarchy dialog to select the PeopleSoft business domains that contain the metadata objects you want to import.

3. Select a folder and click **Show Entities**.

   The Import Wizard displays a list of objects in the selected business domain in the Folder dialog.

4. Review this dialog to ensure that you are selecting an appropriate number of objects.

   Some business domains can contain more than 1000 objects. Importing such a large amount of metadata can take from one to three hours or more, depending on the network connection speed and the processing power of the source and target systems.

5. Click **OK.**

   The wizard displays the Filter Information page with the PeopleSoft business domain displayed in the Business Domain field.

Filtering PeopleSoft Metadata by Text String

1. Select **Text String, where object**.

2. Select the objects you wish to import. You can select Tables, Views, and Sequences.

   If you wish to select specific objects, type the object name in the text field. Create a filter for object selection by using the wildcard characters (%) for zero or more matching characters, and (_) for a single matching character.

   For example, if you want to search the business domain for tables whose names contain the word CURRENCY, then type `%CURRENCY%`. If you want to refine the search to include only tables named CURRENCY and followed by a single digit, then type `%CURRENCY_`.

Selecting the Objects

The Object Selection page contains a description of the objects and enables you to select the objects you want to import into the PeopleSoft module. To select the objects:

1. Move the objects from the available list to the selected list.

   The Import Wizard also enables you to choose whether you want to import tables with foreign key relationships for each object that you choose to import. You can select one of the following:

   None: Import only the objects in the Selected list.

   One Level: Import the objects in the Selected list and any tables linked to it directly through a foreign key relationship.

   All Levels: Import the objects in the Selected list and all tables linked to it through foreign key relationships.

   The foreign key level you select is the same for all tables selected for importing.

   > **Note:** Selecting All Levels increases the time it takes to import the metadata because you are directing the wizard to import tables that are related to each other through foreign key constraints. Select this option only if it is necessary.

2. Click **Next.**

   If you select One Level or All Levels, the Confirm Import Selection dialog appears.

   Review this dialog to ensure that you are selecting an appropriate number of tables.

3. Click **OK.**

   The selected objects appear in the right pane of the Object Selection page.

4. Click **Next.**

   The wizard displays the Summary and Import page.

Reviewing Import Summary

The wizard imports definitions for the selected tables from the PeopleSoft Application Server, stores them in the PeopleSoft source module, and then displays the Summary and Import page.

You can edit the descriptions for each object by selecting the description field and typing a new description.

Review the information on the Summary and Import page and click **Finish.**

The PeopleSoft integrator reads the table definitions from the PeopleSoft application server and creates the metadata objects in the Warehouse Builder repository.

The time it takes to import the PeopleSoft metadata to the Warehouse Builder repository depends on the size and number of tables and the connection between the PeopleSoft application server and the repository. Importing 500 or more objects could take one to three hours or more, especially if you are connecting servers in separate LANs.

When the Import completes, the Import Results dialog displays. Click **OK** to finish importing.

Extracting Data From SAP Applications

Many companies implement SAP ERP system and Warehouse Builder enables easy access to the data in these SAP systems.

This section describes how you can extract data from an SAP system. It describes why you require an SAP connector, how to import metadata from SAP tables, use them in a mapping, generate ABAP code for the mappings, and deploy them to a SAP system. The chapter also describes the various methods by which you can extract data from the SAP system and load this data into a target table on the Warehouse Builder system.

This section contains the following topics:

Why SAP Connector

An SAP R/3 system operates differently compared to SQL based systems like EBusiness Suite and PeopleSoft.

The major differences include:

- The native data manipulation language is ABAP, which is a proprietary SAP language.
- Table names are cryptic compared to those in SQL based ERP systems.
- In addition to database tables, SAP contains logical tables called pool tables and cluster tables. These tables contain multiple physical tables and must be managed differently.

The SAP connector assists you in managing all these issues. Furthermore, the SAP connector allows you to comply with the administrative and security processes of the SAP environment.

Supported SAP Versions

For information about the SAP R/3 versions supported by Oracle Warehouse Builder 10g Release 2, log in to `https://metalink.oracle.com`, and navigate to the **Certify** link.

Overview of SAP Objects

This section provides a brief overview of the different types of tables in SAP, and how data is organized within an SAP system. The section consists of the following topics:

- SAP Object Types
- SAP Business Domains

SAP Object Types

With the SAP Connector, you can import metadata definitions for the following SAP table types:

- **Transparent**: A transparent table is a database table that stores data. You can access the table from non-SAP systems as well, for example, using SQL statements. However, Warehouse Builder uses ABAP code to access transparent tables.
- **Cluster**: A cluster table is usually used to store control data. It can also be used to store temporary data or documentation. Because cluster tables are data dictionary tables and not database tables, you can only access these tables using ABAP.
- **Pooled**: This is a logical table that must be assigned to a table pool in the database. A table pool consists of multiple pooled tables. A pooled table is used to store control data such as program parameters. You require ABAP code to access pooled tables.

SAP Business Domains

SAP application systems logically group tables under different business domains. In SAP, a business domain is an organizational unit in an enterprise that groups product and market areas. For example, the Financial Accounting (FI) business domain represents data describing financial accounting transactions. These transactions might include General Ledger Accounting, Accounts Payable, Accounts Receivable, and Closing and Reporting.

When you import SAP definitions, you can use a graphical navigation tree in the Business Domain Hierarchy dialog box to search the business domain structure in the SAP source application. This navigation tree enables you to select SAP tables from the SAP application server.

Overview of the Warehouse Builder-SAP Interaction

Moving data from an SAP system to an Oracle database using Warehouse Builder consists of the following tasks:

1. Connecting to the SAP system.
2. Importing metadata from SAP data objects.
3. Creating an extraction mapping in Warehouse Builder that defines:
   - The SAP source tables from which data is to be imported
   - The transformation operators that operate on the source tables to import data based on certain criteria

- The target table in Warehouse Builder to store the data imported from the SAP source tables

4. Deploying the mapping.

   This creates the ABAP code for the mapping.

5. Starting the mapping.

   This results in the following sequence of tasks, all of which are performed automatically by Warehouse Builder:

   - Transfer of the ABAP code to the SAP server.
   - Compiling of the ABAP code.
   - Execution of the ABAP code, which results in the generation of a data file (this file has a `.dat` extension).
   - Transfer of the data file to the Warehouse Builder server using FTP.
   - Loading data from the data file into the target table in Warehouse Builder. The loading takes place using SQL*Loader.

SAP Function Modules

To access SAP data from non-SAP systems, you typically use a function module to execute an ABAP program that extracts the data. A function module in SAP is a procedure that is defined in a special ABAP program known as function group. Once defined, the function module can then be called from any ABAP program.

SAP contains a predefined function module called RFC_ABAP_INSTALL_AND_RUN to execute ABAP code. To upload the Warehouse Builder generated ABAP code and execute it in SAP, you need access rights to this function module.

Alternately, you can ask the SAP administrator to create a customized function module that executes a specific ABAP program. You can then use this function module to execute the ABAP code generated by Warehouse Builder.

Data Rerieval Mechanisms

Data retrieval from the SAP system can be Completely Managed By Warehouse Builder, Managed By Warehouse Builder With SAP Verification, or Manual. This depends on whether the SAP administrator provides the Warehouse Builder user with access rights to the predefined function module RFC_ABAP_INSTALL_AND_RUN or whether the SAP administrator creates a customized function module to execute the ABAP code.

Completely Managed By Warehouse Builder

In this mechanism, Warehouse Builder has access to upload and execute the generated ABAP using the default function module RFC_ABAP_INSTALL_AND_RUN, and to use FTP to import the generated data file from the SAP system.

Thus the entire process of retrieving data from the SAP system and creating a target table is managed by the Warehouse Builder and can be completely automated. It is therefore the simplest method of data retrieval. See "Automated System" on page 18-25 for more details on implementing this data retrieval mechanism.

Managed By Warehouse Builder With SAP Verification

In this mechanism, as a Warehouse Builder user, you do not have access rights to the default function module RFC_ABAP_INSTALL_AND_RUN that executes the ABAP

code in the SAP system. Instead the SAP administrator first verifies the ABAP code that you generate using Warehouse Builder, and then creates a customized function module to execute this ABAP code. You can then run the ABAP code on the SAP system using this customized function module.

See "Semi Automated System" on page 18-26 for more details on implementing this data retrieval mechanism.

Manual

In this method, as a Warehouse Builder user, you cannot directly run the ABAP code on the SAP system. Instead, you generate the ABAP code for the mapping, and send it to the SAP administrator, who runs the code on the SAP system. You then import the generated data file using FTP and load the target table.

The tasks involved in retrieving data using FTP and creating the Oracle table are implemented using a Process Flow. See "Manual System" on page 18-28 for more details on implementing this system.

Implementing an SAP Data Retrieval Mechanism

As a Warehouse Builder user, you need to be aware of certain restrictions while trying to import data from an SAP system.

Since the SAP and Oracle Warehouse Builder systems are totally independent systems, as a Warehouse Builder user, you may only have restricted access rights to the SAP data (especially in the production environment). You will therefore have to interact with the SAP administrator to extract data from the system.

Access rights to the SAP system is most often determined by whether it is the development, test, or the production environment. Each of the data retrieval mechanisms can be implemented in the development, test, or production environment depending on the privileges granted by the SAP system administrator.

Development Environment

Typically, in the development environment, the SAP administrator gives you access rights to use the predefined function module RFC_ABAP_INSTALL_AND_RUN. Therefore, in this environment, you can implement a completely Automated System for data retrieval.

Test and Production Environment

Typically, in the test and production environments, the SAP administrator may not give you access rights to use the predefined function module RFC_ABAP_INSTALL_AND_RUN. Instead, the SAP administrator verifies the ABAP code, and either creates a customized function module that you can use, or runs the ABAP code on the SAP system, and allows you to extract the resultant data. You can therefore implement either a Semi Automated System or a Manual System for data retrieval.

A typical data retrieval system may therefore consist of any of the three mechanisms implemented in the different environments.

Scenario 1

You run the automated system in the SAP development environment. Once you verify the ABAP code in this environment, you then move the ABAP code to the SAP test environment and test the code using a customized function module. You then finally move this to the SAP production environment.

This implementation is recommended by Oracle, as it automates and simplifies the data retrieval task.

Scenario 2

Depending on the access rights to the development, test, and production environments, you implement any one of the data retrieval mechanisms in each of the environments.

The following sections provide details of the tasks involved in retrieving data from an SAP system:

1. Connecting to an SAP System
2. Importing Metadata from SAP Tables
3. Creating SAP Extraction Mappings
4. Retrieving Data from the SAP System

Connecting to an SAP System

To connect to an SAP system from Warehouse Builder, you require certain SAP-specific DLL files. Once you establish connection, you can then import metadata from SAP tables into SAP modules in Warehouse Builder.

This section contains the following topics:

- Required Files For SAP Connector
- Creating SAP Module Definitions
- Troubleshooting Connection Errors
- Creating SAP Module Definitions

Required Files For SAP Connector

Different sets of files are required depending on whether you are working on a Windows or a Unix system.

Files Required In Windows

The SAP Connector requires a dynamic link library file named `librfc32.dll` to use remote function calls on the client computer. You must copy `librfc32.dll` to the location specified in `java.library.path` on your client system.

To find this location, click **MyComputer**, **Properties**, and then click **Advanced**. Next click **Environment Variables**, and under System variables, check the locations specified for the variable *Path*.

You can copy the `librfc32.dll` file to any one of the multiple locations specified in Path. One of the locations will correspond to OWB_ORACLE_HOME, and is therefore the preferred location. This location is usually *OWB_ORACLE_HOME*`\owb\bin`.

See Table 18–1 for the list of files required in Windows.

Table 18–1 Required Files for Windows

Required Files	Path	Description
librfc32.dll	*OWB_ORACLE_HOME*`\owb\bin`	This file is available on the SAP Application Installation CD.

Table 18–1 (Cont.) Required Files for Windows

Required Files	Path	Description
sapjcorfc.dll	*OWB_ORACLE_HOME*\owb\bin	Copy this file to the same location where you placed librfc32.dll
sapjco.jar	*OWB_ORACLE_HOME*\owb\lib\int	

Restart the client after copying these files.

Files Required In Unix

The SAP Connector requires a dynamic link library file named librfccm.so to use remote function calls on the client computer. You need to copy this file to the location specified by the Unix environment variable path LD_LIBRARY_PATH on your client system.

By default, *OWB_ORACLE_HOME*/owb/bin/admin is the location specified in LD_LIBRARY_PATH. If it is not, then ensure that you add *OWB_ORACLE_HOME*\owb\bin\admin to LD_LIBRARY_PATH.

See Table 18–2 for the list of files required in Unix.

Table 18–2 Required Files for Unix

Required Files	Path	Description
librfcccm.so	*OWB_ORACLE_HOME*\owb\bin\admin	This file is available on the SAP Application Installation CD.
libsapjcorfc.so	*OWB_ORACLE_HOME*\owb\bin\admin	Copy this file to the same location where you placed librfcccm.so
sapjco.jar	*OWB_ORACLE_HOME*\owb\lib\int	

Restart the client after copying these files.

> **Note:** Different versions of SAP R/3 might require different versions of the DLL, SO, and JAR files. The correct versions are available in the SAP installation CD. The files can also be downloaded from:
>
> http://service.sap.com/patches

Troubleshooting Connection Errors

The most common errors while connecting to an SAP system are listed in Table 18–3:

Table 18–3 SAP Connection Errors

Error Message	Possible Reason
Connection failed. You are not authorized to logon to the target system (error code 1).	Incorrect User Name or Password to connect to the SAP server.
Connection failed. Connect to SAP gateway failed.	Incorrect Application Server, System Number, or Client details.
Some Location Details are missing. Please verify the location information is completely specified.	Missing DLL files, or DLL files placed in the wrong location.

Table 18–3 (Cont.) SAP Connection Errors

Error Message	Possible Reason
Missing `saprfc32.dll`	Missing `saprfc32.dll` file, or file placed in the wrong location.

Note: If you create an SAP source module and import SAP tables but cannot see the columns in the tables, then you have an incompatible `librfc32.dll` file. Download the correct version of the DLL file from the SAP Website.

Creating SAP Module Definitions

Use the Create Module Wizard to create an SAP source module that stores data from an SAP source.

To create a SAP Module:

1. Right-click **SAP** and select **New SAP**.

   The Create Module Wizard is displayed.

2. On the Name and Description page, provide a name for the SAP module. Select the module status and optionally also provide a description. Click **Next**.

3. On the Connection Information page, either select from an existing location or click **Edit** to open the Edit SAP Location dialog box. Specify the details as described in "Connecting to an SAP System" on page 18-15. Click **Next**.

4. On the Summary page, click **Finish**.

   A new sap module is now available on the Projects Navigator.

Note: Before you create a SAP location, ensure that you have all the necessary information. You can provide the location information either while creating the module or before importing metadata into the module. You need the following information to create the location: server name, user name, password, system number, and client number. Obtain these details from your system administrator.

When you set the connection information, you can choose one the following connection types:

Remote Function Call (RFC)

A remote function call enables you to call a function module on a remote system. This method requires specific IP Address information for the SAP application server.

SAP Remote Function Call (SAPRFC.INI)

You can also specify the connection information in a file called `SAPRFC.INI`, and copy this file to the following location: *`OWB_ORACLE_HOME`*`\owb\bin\admin`.

Using the `SAPRFC.INI` file requires prior knowledge of ABAP parameters, as you need to specify the values for certain parameters to make a SAP connection, and is not the recommended connection method if you are not familiar with ABAP.

Note: The `SAPRFC.INI` file comes with the SAP installation CD.

The Create Module Wizard creates the module for you based on the metadata contained in the SAP application server.

Connecting to an SAP System **1.** Select one of the following connection types:

- Remote Function Call (RFC)

  This is the recommended connection type, and is selected by default in Warehouse Builder.

- SAP Remote Function Call (SAPRFC.INI)

For more information about these connection types, see "Creating SAP Module Definitions" on page 18-17.

2. Type the connection information in the appropriate fields. The fields displayed on this page depend on the connection type you choose.

Note: Ensure that you have copied the DLL files to the right location. For more information, see "Required Files For SAP Connector" on page 18-15.

You must obtain the connection information to your SAP Application server from your system administrator before you can complete this step.

RFC Connection type requires the following connection information:

Application Server: The alias name or the IP address of the SAP application server.

System Number: The SAP system number. This must be provided by the SAP system administrator.

Client: The SAP client number. This must be provided by the SAP system administrator.

User Name: The user name with access rights to the SAP system. This name is supplied by the SAP system administrator.

Language: EN for English or DE for German. If you select DE, the description text displays in German and all other text displays in English.

SAPRFC connection type requires the following connection information:

RFC Destination: Type the alias for the SAP connection information.

In addition, both the connection types require the following connection information if the ABAP code is to be executed in SAP using a function module and the data file is to be transferred by FTP to Warehouse Builder:

Host Login User Name: A valid user name on the system that hosts the SAP application server. This user must have access rights to copy the data file using FTP.

FTP Directory: The directory in the SAP server that stores the data file generated when the ABAP report is executed. For systems where the FTP directory structure is identical to the operating system directory structure, this field can be left blank. For systems where the file system directory structure is mapped to the FTP

directory structure, enter the FTP directory path that is mapped to staging file directory in the file system directory structure. For example, on a computer that runs Windows, the staging file directory `"C:\temp"` is mapped to `"/"` in the FTP directory structure, then enter `"/"` in this field.

Execution Function Module: In a SAP instance, if a remote function module other than the SAP delivered function module: RFC_ABAP_INSTALL_AND_RUN is used to remotely execute ABAP reports through RFC connections, then enter the remote function module name here.

3. Click **Test Connection** to verify that the connection information you provided are correct.
4. Click **OK** to go back to the Connection Information page of the Create Module wizard.

Importing Metadata from SAP Tables

Once you establish a connection with the SAP server, you can import metadata from SAP tables.

This section contains the following topics:

- Importing SAP Metadata Definitions
- Analyzing Metadata Details

Importing SAP Metadata Definitions

After creating the SAP source module, you can import metadata definitions from SAP tables using the Import Metadata Wizard. This wizard enables you to filter the SAP tables to import, verify those tables, and reimport them. You can import metadata for transparent tables, cluster tables, or pool tables.

Perform the following steps to import SAP metadata:

1. From the Project Explorer, expand the **Applications** node.
2. Right-click the SAP source module into which you want to import metadata and select **Import**.

   Warehouse Builder displays the Welcome page for the Import Metadata Wizard.
3. Click **Next.**
4. Complete the following tasks:
   - Filtering SAP Metadata
   - Selecting Objects for Metadata Import
   - Reviewing Import Summary

Filtering SAP Metadata You can filter objects to import by business domain or by text strings. Select a filtering method and click **Next**.

Filtering SAP Metadata by Business Domain

1. Select **Business Domain** and click **Browse** to display the SAP R/3 Business Domain Hierarchy dialog box.

   The Import Metadata wizard displays the Loading Progress dialog box while it is retrieving the business domains.

2. The Business Domain Hierarchy dialog box lists the available SAP business domains.

   Note: It may take a few minutes to list the SAP business domains depending on factors such as the network location of the SAP application server, the type of LAN used, and the size of the SAP application database.

   Use the Business Domain Hierarchy dialog box to select the SAP business domains that contain the metadata tables you want to import.

3. Select a folder and click **Show Tables** to view the tables available in a business domain.

   The Import Wizard displays a list of tables in the selected business domain in the Folder dialog box.

4. Review this dialog box to ensure that you are selecting the required tables.

   Some business domains can contain more than 1000 tables. Importing such a large amount of metadata can take time, depending on the network connection speed and the processing power of the source and target systems.

5. Click **OK.**

   The wizard displays the Filter Information page with the SAP business domain displayed in the Business Domain field.

Filtering SAP Metadata by Text String

1. Select **Text String, where object** and use the **Name matches** or **Description matches** entry field to type a string and obtain matching tables from the SAP data source.

   The **Description matches** field is case sensitive, the **Name matches** field is not.

   Create a filter for object selection by using the wildcard characters `%` for zero or more matching characters, and _ for a single matching character.

   For example, if you want to search the business domain for tables whose descriptions contain the word CURRENCY, then select **Description matches** and type `%CURRENCY%`. You can also search for tables by their names.

2. Specify the number of tables you want to import in the Maximum number of objects displayed field.

Selecting Objects for Metadata Import The Object Selection page contains a description of the tables and enables you to select the tables you want to import into the SAP module. To select the tables:

1. Move the tables from the available list to the selected list.

   The Import Metadata Wizard also enables you to choose whether you want to import tables with foreign key relationships for each table that you choose to import. You can select one of the following:

   None: Import only the tables in the Selected list.

   One Level: Import the tables in the Selected list and any tables linked to them directly through a foreign key relationship.

All Levels: Import the tables in the Selected list and all tables linked to them hrough foreign key relationships.

2. Click **Next.**

   If you select One Level or All Levels, the Confirm Import Selection dialog box is displayed.

   Review this dialog box to ensure that you are selecting the required tables.

3. Click **OK.**

   The selected tables appear in the Selected list of the Table Selection page.

4. Click **Next.**

   The wizard displays the Summary and Import page.

Reviewing Import Summary The wizard imports the definitions for the selected tables from the SAP Application Server, stores them in the SAP source module, and then displays the Summary and Import page.

You can edit the descriptions for each table by selecting the Description field and typing a new description.

Review the information about the Summary and Import page and click **Finish.**

The SAP Connector reads the table definitions from the SAP application server and creates the metadata objects in the workspace.

The time it takes to import the SAP metadata into the workspace depends on the size and number of tables and the connection between the SAP application server and the workspace. It is a best practice to import small batches of tables to allow better performance.

When the import completes, the Import Results dialog box displays. Click **OK** to finish importing metadata.

Reimporting SAP Tables To reimport SAP tables, follow the importing procedure using the Import Metadata Wizard. Prior to starting the import, the wizard checks the source for tables with the same name as those you are importing. The tables that have already been imported appear in bold in the Object Selection page. On the Summary and Import page, the Action column indicates that these tables will be reimported. The wizard then activates the **Advanced Synchronize Options** button so that you can control the reimport options.

Note: If you wish to undo the reimport, click **Undo**. This ensures that no changes are made to the existing metadata.

Analyzing Metadata Details

With SAP tables, you cannot view the data after you import the metadata from these tables. However, you can get a good insight about the data that is stored in the tables by viewing the Column Descriptions and the Constraints Details.

Column Descriptions

You can view the column description of each of the columns in a table. This is valuable because the column names in SAP can be non-descriptive, and difficult to interpret if you have not previously seen the data in the table.

To view the descriptions, double-click the table to open the object editor for the table, and then click the **Columns** editor.

Constraints Details

The other benefit of data object editor is that you can get information about the primary and foreign keys within the table. To view the key constraints, click the **Constraints** editor.

> **Note:** It is also a useful practice to display the business names of the SAP tables in the Projects Navigator. Business names provide a description of the tables and are therefore more intuitive than the physical names. To view the business names for tables in Warehouse Builder, from the main menu, click **Tools**, **Preferences**, **OWB, Naming**, and then select **Business Names** in the Naming Mode field.

Creating SAP Extraction Mappings

After importing metadata from SAP tables, you must define the extraction mapping to extract data from the SAP system.

Defining an SAP Extraction Mapping

You can use the Mapping Editor to create a mapping containing SAP tables. Creating a mapping with SAP tables is similar to creating mappings with other database objects. However, there are restrictions on the operators that can be used in the mapping. You can only use Table, Filter, Joiner, and Mapping Input Parameter mapping operators in a mapping containing SAP tables.

A typical SAP extraction mapping consists of one or more SAP source tables (transparent, cluster, or pooled), one or more filter or joiner operators, and a non-SAP target table (typically an Oracle table) to store the imported data.

> **Note:** The source table is always an SAP table. Note that you cannot have both SAP and non-SAP (Oracle) source tables in a mapping, but the staging table is an Oracle table.

This section contains the following topics:

- Adding SAP Tables to the Mapping
- Setting the Loading Type
- Setting Configuration Properties for the Mapping

Adding SAP Tables to the Mapping To add an SAP table to a mapping:

On the Mapping Editor drag and drop the required SAP table onto the Mapping Editor canvas.

The editor places a Table operator on the mapping canvas to represent the SAP table.

Setting the Loading Type Use the Operator properties panel of the Mapping Editor to set the SQL*Loader properties for the tables in the mapping.

To set the loading type for an SAP Source Table:

1. On the Mapping Editor, select the SAP source table. The Table Operator Properties panel displays the properties of the SAP table operator.
2. Select a loading type from the Loading Type list. With ABAP code as the language for the mapping, the SQL*Loader code is generated as indicated in Table 18–4.

Table 18–4 SQL*Loader Code Generated in ABAP

Loading Type	Resulting Load Type in SQL*Loader
INSERT	APPEND
CHECK/INSERT	INSERT
TRUNCATE/INSERT	TRUNCATE
DELETE/INSERT	REPLACE
All other types	APPEND

Setting Configuration Properties for the Mapping ■Use the Configuration Properties dialog box to define the code generation language as described in Setting the Language Parameter.

- Set ABAP specific parameters, and the directory and initialization file settings in the Configuration Properties dialog box as described in Setting the Runtime Parameters.

Setting the Language Parameter

This parameter enables you to choose the type of code you want to generate for a mapping. For mappings containing SAP source tables, Warehouse Builder enables you to select either PL/SQL or ABAP.

If the SAP system uses a non-Oracle database to store data, then you must select ABAP to generate code. If the SAP data is stored on an Oracle database, then you can specify PL/SQL. However, with PL/SQL, you cannot extract pool or cluster tables. Therefore, in all instances it is desirable to set the language to ABAP.

Setting the Runtime Parameters

With the language set to ABAP, you can expand the Runtime Parameters node in the Configuration Properties dialog box to display settings specific to ABAP code generation.

Some of these settings come with preset properties that optimize code generation. It is recommended that these settings be retained, as altering them may slow down the code generation process.

The following Runtime parameters are available for SAP mappings:

- **Background Job**: Select this option if you wish to run the ABAP report as a background job in the SAP system. Enable this option for the longer running jobs. Foreground batch jobs that run for a long duration are considered hanging in SAP after a certain time. Therefore it is ideal to have background job running for such extracts.
- **File Delimiter for Staging File**: Specifies the column separator in a SQL data file.
- **Data File Name**: Specifies the name of the data file that is generated when the ABAP code for the mapping is run in the SAP system.
- **SQL Join Collapsing**: Specifies the following hint, if possible, to generate ABAP code.

```
SELECT < > INTO < > FROM (T1 as T1 inner join T2 as T2) ON <condition >
```

The default setting is TRUE.

- **Primary Foreign Key for Join**: Specifies the primary key to be used for a join.
- **ABAP Report Name**: Specifies the name of the ABAP report generated by the mapping. This is required only when you are running a custom function module to execute the ABAP code.
- **SAP System Version**: Specifies the SAP system version number to which you want to deploy the ABAP code. The characteristics of the generated ABAP code depends on the version number. For MySAP ERP and all other versions, select SAP R/3 4.7. Note that different ABAP code is generated for versions prior to 4.7.
- **Staging File Directory**: Specifies the location of the directory in the SAP system where the data file generated by ABAP code resides.
- **SAP Location**: The location of the SAP instance from where the data can be extracted.
- **Use Select Single**: Indicates whether Select Single is generated, if possible.
- **Nested Loop**: Specifies a hint to generate nested loop code for a join, if possible.

SQL*Loader Settings Specify a name for the control file. By default, the control file name is the same as the data file name specified in the Data File Name field. This means that the ABAP code will generate a single control file containing both the SQL*Loader control information and the data (since the log files are the same). The control file and the data file can be given different names, in which case different files are generated for the control information and data, and both the files are transferrred by FTP.

Setting the Join Rank You need to set this parameter only if the mapping contains the Joiner operator, and you wish to explicitly specify the driving table. Unlike SQL, ABAP code generation is rule based. Therefore, you must design the mapping in such a way that the tables are loaded in the right order. Or you can explicitly specify the order in which the tables have to be joined. To do this, from the Configuration Properties dialog box, expand **Table Operators**, and then for each table, specify the Join Rank. The driving table must have the Join Rank value set to 1, with increasing values for the subsequent tables.

You can also let Warehouse Builder decide the driving table, as well as the order of joining the other tables. In such cases, do not enter values for Join Rank.

Retrieving Data from the SAP System

After designing the extraction mapping, you must validate, generate, and deploy the mapping, as you do with all mappings in Warehouse Builder.

To generate the script for the SAP mapping:

1. Right-click the SAP mapping and select **Generate.**

   The Generation Results window is displayed.

2. On the Script tab, select the script name and select **View Code.**

   The generated code is displayed in the Code Viewer.

   You can edit, print, or save the file using the code editor. Close the Code Viewer to return to the Generation Results window.

3. To save the file, click **Save as File** and save the ABAP program to your hard drive.

After you generate the SAP mapping, you must deploy the mapping to create the logical objects in the target location. To deploy an SAP mapping, right-click the mapping and select **Deploy**. You can also deploy the mapping from Control Center Manager.

When an SAP mapping is deployed, an ABAP mapping is created and stored in the Warehouse Builder runtime schema. Warehouse Builder also saves the `ABAP` file under *`OWB_ORACLE_HOME`*`\owb\deployed_files`, where *`OWB_ORACLE_HOME`* is the location of the Oracle home directory of your Warehouse Builder installation. Note that if you are using the Warehouse Builder installation that comes with Oracle Database, then this is the same as the database home.

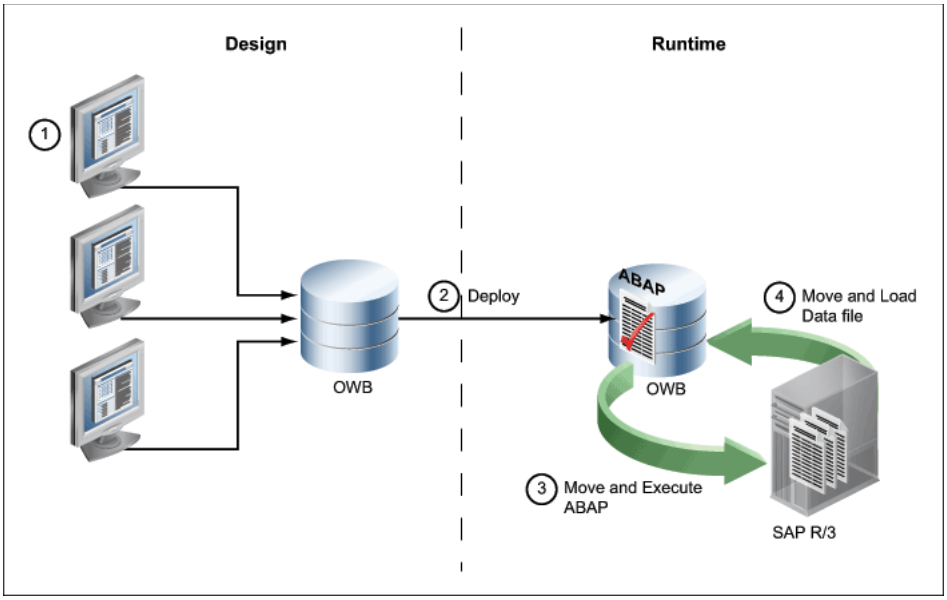
Depending on whether data retrieval from the SAP system is fully automated, semi-automated, or manual, you need to carry out the subsequent tasks. This section consists of the following topics:

- Automated System
- Semi Automated System
- Manual System

Automated System

In a completely automated system, as a Warehouse Builder user you have access to the predefined function module in the SAP system. This allows you to execute any ABAP code and extract data directly from the SAP system without being dependent on the SAP administrator, as shown in Figure 18–1.

Figure 18–1 Automated Data Retrieval



Because there is no dependence, you can automate the process of sending the ABAP code to the SAP system and importing the data file from the SAP system. Warehouse Builder will then use FTP to transfer the data file to the Warehouse Builder system, and load the target file with the imported data using SQL*Loader.

An automated system works as follows:

1. You design the extraction mapping and generate the ABAP code for this mapping.

2. Before deploying the mapping, ensure that you have set the following configuration properties for the mapping:
   - **ABAP Report Name**: The file that stores the ABAP code generated for the mapping.
   - **SAP Location**: The location on the SAP system from where data is extracted.
   - **Data File Name**: Name of the data file to store the data generated by the execution of ABAP code.

   Also ensure that you have provided the following additional connection details for the SAP location:
   - **Execution Function Module**: Provide the name of the predefined SAP function module. Upon execution, this function module will take the ABAP report name as the parameter, and execute the ABAP code.
   - **FTP Directory**: The directory on the Warehouse Builder system. The data file generated upon the execution of the function module will be transferred to this directory using FTP. Note that this requires an FTP server to be located in the SAP system.
   - Also provide a username who has write permissions on the FTP directory.
3. You then start the mapping, following which the following tasks are automatically performed:
   - Warehouse Builder deploys the ABAP and uses RFC_ABAP_INSTALL_AND_RUN to both load the ABAP and execute it in SAP.

     The ABAP code is sent to the SAP system using a Remote Function Call (RFC).
4. In the SAP system, the ABAP code extracts data from the source tables and creates a data file.

   This data file is stored in the location specified by Staging File Directory.
5. Warehouse Builder uses FTP to transfer this data file back to the Warehouse Builder system.

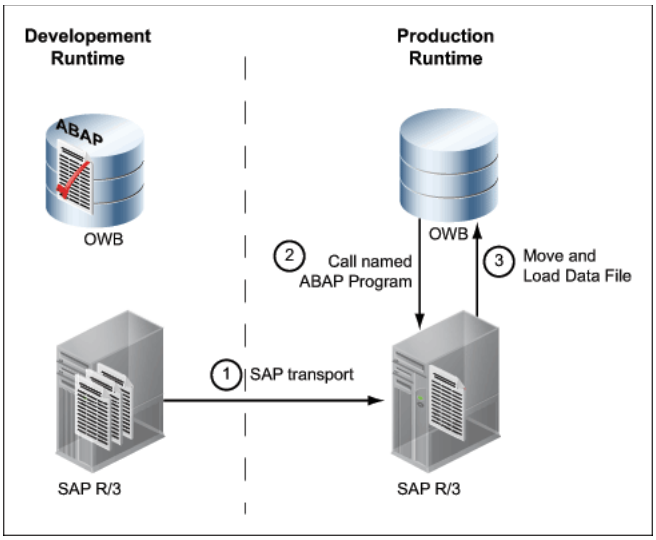
   The file is stored in the location specified in the FTP Directory field.
6. Using SQL*Loader, Warehouse Builder loads the target table in the mapping with the data from the data file.

The advantage of this system is that you can create a fully automated end-to-end solution to extract SAP data. As a user, you just create the extraction mapping and run it from Warehouse Builder, which then creates the ABAP code, sends it to the SAP system, extracts the resultant data file, and loads the target table with the extracted data.

Semi Automated System

In a semi automated system, as a Warehouse Builder user, you do not have access to the predefined function module RFC_ABAP_INSTALL_AND_RUN, and therefore cannot use this function module to execute ABAP code. Instead, you must create an extraction mapping, deploy it, and then send the ABAP code to the SAP administrator who verifies the code before allowing you to run it in the SAP system, as shown in Figure 18–2.

Figure 18–2 Semi Automated Implementation



A semi automated system works as follows:

1. You design the extraction mapping and generate the ABAP report for this mapping.

   You can then test this report in the development environment.

2. You then send the ABAP report to the SAP administrator, who tests the report, and loads it to the SAP repository in the production environment.

3. The SAP administrator can create a new report or use the same report that you send.

4. If the SAP administrator creates a new report, then obtain the name of the new report and use it in your mapping to extract data from the production environment.

5. Before you run the mapping in the production environment, ensure that you have set the following configuration properties for the mapping:

   - **ABAP Report Name**: The SAP administrator will provide the name of the ABAP report after verifying the ABAP code. You will then use this report name to extract data.
   - **SAP Location**: The location on the SAP system from where data is extracted.
   - **Data File Name**: Name of the data file to store the data generated during execution of ABAP code.

   Also ensure that you have provided the following additional connection details for the SAP location:

   - **Execution Function Module**: Provide the name of the custom function module created by the SAP administrator. On execution, this function module takes the ABAP report name as the parameter, and executes the ABAP code. You must obtain the function module name from the SAP administrator.
   - **FTP Directory**: A directory on the SAP system. The data file generated by the execution of the ABAP code is saved to this directory. Warehouse Builder will import the data file using FTP. Note that the FTP server resides on the SAP system.

- Also provide a username who has Read permissions on the FTP directory.

6. In the production environment, when you run the mapping, Warehouse Builder sends the ABAP report name and the custom function module to the SAP system using a Remote Function Call (RFC).
7. In the SAP system, the ABAP code gets executed and a data file is generated. Note that the ABAP code gets executed only if the ABAP report name and the function module is available.

   This data file is stored in the location specified by Staging File Directory.
8. Warehouse Builder imports the data file using FTP. Note that an FTP server must be available on the SAP server.
9. Warehouse Builder uses SQL*Loader to load the target table with data from the data file.

Manual System

In a manual system, your role as a Warehouse Builder user is restricted to generating the ABAP code for the mapping, and sending the ABAP code to the SAP administrator. The tasks involved in this system are:

1. You create an extraction mapping, and generate the ABAP code for the mapping.
2. While designing the mapping, make sure that you specify the Data File Name to store the data file.
3. You send the ABAP code to the SAP administrator.
4. The SAP administrator executes the ABAP code in the SAP system.
5. On execution of the code, a data file is generated.

You can then create a Process Flow to import the data file. The Process Flow may typically consist of the following activities:

1. A File Exists activity checks for the availability of the data file.
2. If the file exists, then an FTP activity transfers the file to the Warehouse Builder system.
3. If the file does not exist, then it must wait till the file is made available, and then perform an FTP.
4. Using SQL*Loader, the target table is loaded with data from the data file.

In most production environments, the SAP administrator may not allow any other user to access the SAP system. In such cases, implementing the manual system may be the only viable option.

Using SQL*Loader in the Process Flow

To use SQL*Loader in the process flow, insert an SQL*Plus activity. To use the SQL*Loader, use the HOST command. Once you insert the SQL*Plus activity, insert the following value for SCRIPT:

```
HOST sqlldr ${Target.User}/${Target.Password} CONTROL=${Working.RootPath}\C.CTL
quit
```

Insert the relevant value for the control (`.ctl`) file name.

Then configure the path settings for the SQL*Plus activity. To do this, right-click the process flow and select **Configure**.

Under SQL*Plus Activities, expand the SQLPLUS node and provide the required values under Path Settings.

Deployed Location refers to the location of the target table. Working location refers to the location of the control file.

19

Validating Data Objects

Validation verifies the definitions of both data objects and ETL objects and identifies any problems or possible errors that could occur during deployment. If objects are invalid, generation and deployment is not possible. You can validate objects and generate scripts for objects at any point in the design process. These functions are also available from within the deployment process; however, you can run them as standalone functions as you define your objects to ensure that the definitions you are providing are complete and valid. In addition, you can generate and view scripts prior to deployment to ensure that there are no problems or issues.

This chapter includes the following topics:

Validating Objects

You can manually select objects for validation at anytime. Select the **Validate** operation from the Object menu or from the right-click menu for an object.

To validate an object or set of objects:

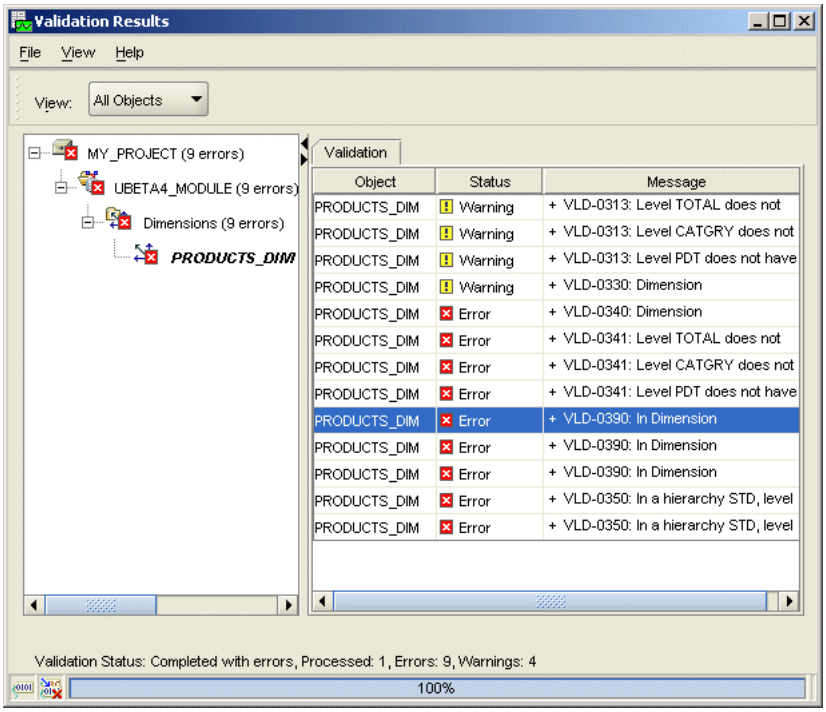
1. Select an object or set of objects from the Project Explorer.

   Use the **Control** key to select multiple objects. You can also select objects that contain objects, such as modules and projects.

2. From the **Object** menu, select **Validate** or right-click the object and select **Validate** from the pop-up menu.

   Warehouse Builder validates the selected object definitions and displays the results in the Validation Results dialog, as shown in Figure 19–1.

Figure 19–1 Validation Results Dialog



Viewing the Validation Results

The Validation Results dialog enables you to view validation results and correct any invalid definitions. When the Validation Results dialog is opened, a navigation tree displays on the left and the validation messages display on the right. You can use these areas to find and view validation messages for specific objects.

The **View** drop-down list at the top of the Validation Results dialog enables you to filter the type of validation messages displayed. The options you can select are as follows:

- All Objects

  Displays all types of validation messages for all the objects that were validated.

- Warnings

  Displays only the warnings. Warnings are not critical errors, but indicate potential problems during deployment.

- Errors

  Displays only the errors generated as a result of the validation. Errors indicate that the object definition is invalid.

The Validation Results dialog consists of the following:

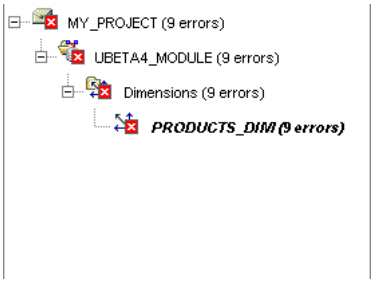
- Validation Results Navigation Tree
- Validation Messages

Note: A Validation Results window displays when you validate a definition. To view these results at a later time, select **Validation Messages** from the **View** menu.

Validation Results Navigation Tree

The object or set of objects that were validated display in a navigation tree on the left side, as shown in Figure 19–2. The navigation tree is useful for locating specific objects when validating a large set of objects.

Figure 19–2 Validation Navigation Tree



Using this tree, you can select an object or type of object and view the specific validation messages on the right.

Validation Messages

The validation messages display on the right side of the Validation Results window. If the object you selected contains other objects, all objects are included in the results. For example, if you select a module for validation, the module definition and the definitions of all the objects within that module are validated. Validation messages display the object name, the validation status, and the validation message.

> **Note:** There can be more than one validation message for a given object. If multiple errors or warnings occur, they display separately.

There are three types of validation status:

- **Success:** Indicates that the validation tests for this object were successfully passed. You can continue.
- **Warning:** Indicates that there may be problems during deployment. This is not a critical error. Warnings are shown to make you aware of potential issues.
- **Error:** Indicates that the object definition is invalid. The object definition cannot be generated.

You can launch the editor that enables you to edit invalid objects by double-clicking the object name in the Validation Messages section or in the navigation tree. For example, if you double-click the object name, PRODUCTS_DIM, either in the navigation tree or the Validation tab shown in Figure 19–1, the Data Object Editor for this dimension is launched. Use this to edit the dimension definition.

> **Note:** When you edit objects from the Validation Results window, the objects are not automatically re-validated. You must re-validate any previously invalid objects.

Double-click a message in the Validation tab open the validation messages in a text editor. You can save this message as a file using the **File** menu of the text editor.

Editing Invalid Objects

When you validate objects, the Validation Results window is displayed. From here you can view the invalid objects and access the editors for these objects directly.

To edit invalid definitions:

1. From the Validation Results window, double-click an invalid object from the tree or from the validation messages grid

   An editor for the selected object opens.

2. Edit the object to correct problems.
3. Close the editor when you are finished and re-validate.

Viewing Generated Scripts

In addition to validating object definitions prior to deployment, you can also generate and view the scripts. During this process, Warehouse Builder validates the object definitions and generates the scripts required to create and populate the objects. You can then view the generation results and the code generated for each script. You can also save these scripts to a file system.

> **Note:** The scripts generated using the method described in this section may not be deployable. This is for viewing and verification purposes only. If you choose, you may also save these scripts to a file system.

Generating Scripts

To generate scripts for an object or set of objects:

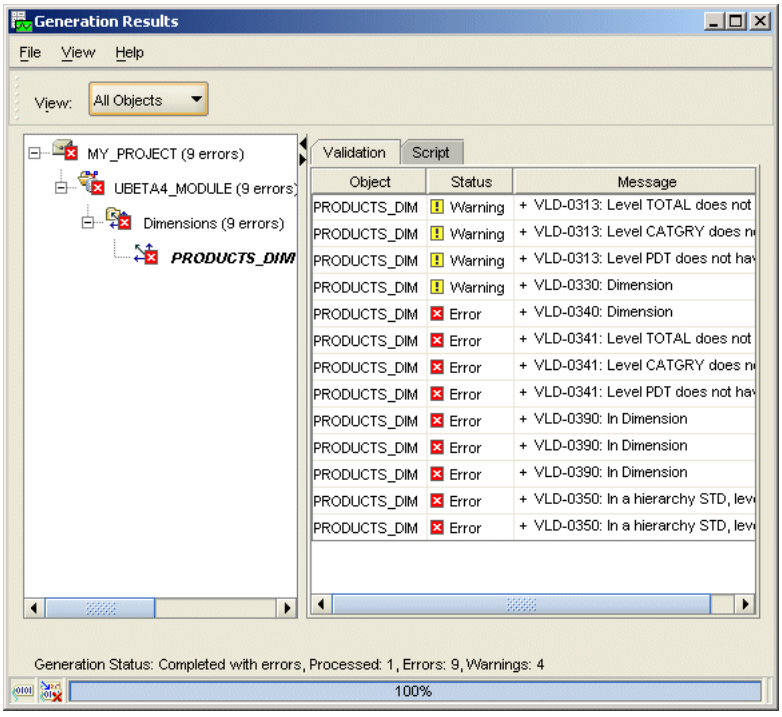
1. From the Project Explorer, select an object or set of objects.

   Select multiple objects using the **Control** key. You can also select modules, however, it may take a long time to generate scripts if it contains a large number of objects.

2. From the **Object** menu, select **Generate.** You can also right-click the object and select **Generate.**

   Warehouse Builder generates the scripts for the selected objects. The Generation Results window, as shown in Figure 19–3, displays when the operation is complete.

Figure 19–3 Generation Results Window



Viewing the Generation Results

The Generation Results dialog is similar to the validation results. You view the results of generating scripts for data objects in this dialog. The left side of this dialog displays a navigation tree. The generation messages display on the right.

You can expand or collapse object types displayed.

The **View** drop-down list at the top of the Generation Results dialog enables you to limit the generation messages displayed. The options you can select are as follows:

- All Objects

  Displays all types of validation messages for all the objects that were validated.

- Warnings

  Displays only the warnings. Warnings are not critical errors, but indicate potential problems during deployment.

- Errors

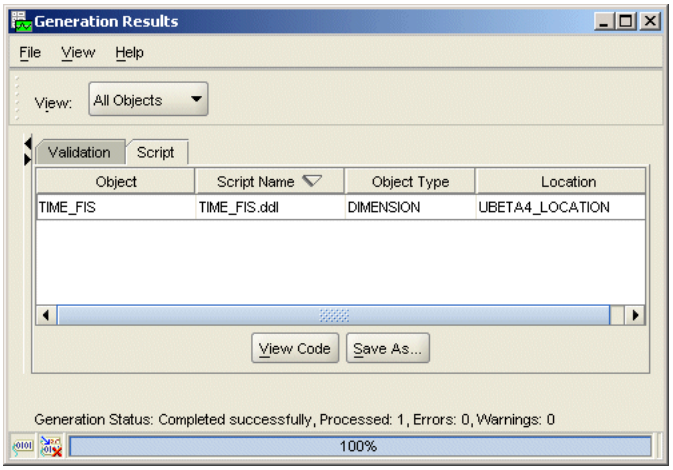
  Displays only the errors generated as a result of the validation. Errors indicate that the object definition is invalid.

> **Note:** The Generation Results window displays when you generate a script to create an object. To view these results at a later time select **Generated Scripts** from the **View** menu.

The Generation Results dialog displays two tabs on the right: Validation and Script. The Validation tab displays the validation messages for the selected objects This is for viewing purposes only. For details on validation, see "Validating Objects" on page 19-1.

The Script tab, as shown in Figure 19–4, displays a list of the scripts that will be generated to create the object you have selected. The list contains the name of the object, the name of the scripts, and the type of object that each script will generate.

Figure 19–4 Script Tab



Viewing the Scripts

After you have generated scripts for your target objects, you can open the scripts and view the code. Warehouse Builder generates the following types of scripts:

- **DDL scripts:** Creates or drops database objects.
- **SQL*Loader control files:** Extracts and transports data from file sources.
- **ABAP scripts:** Extracts and loads data from SAP systems.

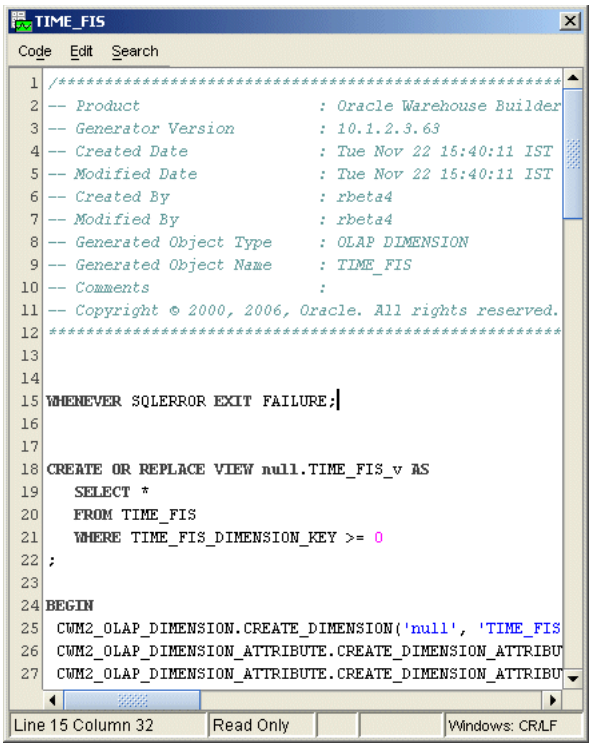
To view the generated scripts:

1. From the Generation Results window, select an object in the navigation tree on the left of the Generation Results dialog.
2. Select the Scripts tab on the right of this dialog.

   The Scripts tab contains a list of the generated scripts for the object you selected.
3. Select a specific script and click the **View Code** button.

   The selected script displays in a code viewer, as shown in Figure 19–5. This is read-only.

Figure 19–5 Generated Script



Saving the Scripts

When you view the generated scripts, you also have to option of saving them to a file system.

To save generated scripts:

1. From the Generation Results window, select an object from the navigation tree on the left.
2. Select the Scripts tab from the bottom section of the window.

   The Scripts tab contains a list of the generated scripts for the object you selected.
3. Select a specific script and click the **Save As** button.

   The Save dialog opens and you can select a location where you want to save the script file.

Part III

Data Quality Reference

This part contains the following chapters:

20

Ensuring Data Quality

Oracle Warehouse Builder provides a set of features that enable you to ensure the quality of data that is moved from source systems to your data warehouse. Data profiling is a feature that enables you to discover inconsistencies and anomalies in your source data and then correct them. Before you transform data, you can define data rules and apply data profiling and data auditing techniques.

This chapter contains the following topics:

Steps to Perform Data Profiling

The section "How to Perform Data Profiling" on page 10-7 describes the steps in the data profiling process. This section lists the steps and provides references to the sections that describe how to perform each step.

Use the following steps to perform data profiling:

1. Load the metadata

   See "Import or Select the Metadata" on page 10-8.

2. Create a data profile

   See "Using Data Profiles" on page 20-2.

3. Profile the data

   See "Profiling the Data" on page 20-13.

4. View the profiling results

   See "Viewing the Results" on page 20-15.

5. Derive data rules based on the results of data profiling

You can create data rules based on the data profiling results. For more information, see "Deriving Data Rules" on page 20-27.

6. Generate corrections

   See "Correcting Schemas and Cleansing Data" on page 20-29.

7. Define and edit data rules manually

   See "Using Data Rules" on page 20-35.

8. Generate, deploy and execute

   See "Generate, Deploy, and Execute" on page 10-9.

Using Data Profiles

Data profiling is a process that enables you to analyze the content and structure of your data to determine inconsistencies, anomalies, and redundancies in the data. To begin the process of data profiling, you must first create a data profile using the Design Center. You can then profile the objects contained in the data profile and then create correction tables and mappings.

This section contains the following topics:

Creating Data Profiles

Use the following steps to create a data profile:

1. From the Warehouse Builder Project Explorer, expand the project node in which you want to create the data profile.
2. Right-click **Data Profiles** and select **New**.

   The Create Data Profile Wizard opens and displays the Welcome page. Click **Next** to proceed. The wizard guides you through the following steps:

   Note: You cannot profile a source table that contains complex data types if the source module is located on a different database instance from the data profile.

Naming the Data Profile

Use the **Name** field to specify a name for your data profile. The name must be unique within a project. The Name tab of the Edit Data Profile dialog enables you to rename the data profile. You can select the name and enter the new name.

Use the **Description** field to provide an optional description for the data profile. On the Name tab, you can modify the description by selecting the current description and typing the new description.

Selecting Objects

Use the Select Objects page to specify the objects that you want to profile. The Select Objects tab of the Edit Data Profile dialog enables you to modify object selections that you made.

The Available section displays a list of objects available for profiling. Select the objects you want to profile and use the shuttle buttons to move them to the Selected list. You can select multiple objects by holding down the **Ctrl** key and selecting the objects. The Selected list displays the objects selected for profiling. You can modify this list using the shuttle buttons.

The objects available for profiling include tables, external tables, views, materialized views, dimensions, and cubes. The objects are grouped by module. When you select a dimensional object in the Available section, Warehouse Builder displays a warning informing you that the relational objects that are bound to these dimensional objects will also be added to the profile. Click **Yes** to proceed.

Note: You cannot profile a source table that contains complex data types if the source module is located on a different database instance than the data profile.

Reviewing the Summary

The Summary page displays the objects that you have selected to profile. Review the information on this page. Click **Back** to make changes or click **Finish** to create the data profile.

The data profile is created and added to the Data Profiles node in the navigation tree. If this is the first data profile you have created in the current project, the Connection Information dialog for the selected control center is displayed. Enter the control center manager connection information and click **OK**. The Data Profile Editor opens.

Editing Data Profiles

Once you create a data profile, you can use the Data Profile Editor to modify its definition. You can also add data objects to an existing data profile. To add objects, you can use either the menu bar options or the Select Objects tab of the Edit Data Profile dialog. For information about using the menu bar options, see "Adding Data Objects to a Data Profile" on page 20-4. The following instructions describe how to access the Edit Data Profile dialog.

To edit a data profile:

1. In the Project Explorer, right-click the data profile and select **Open Editor**.

   The Data Profile Editor is displayed.

2. From the **Edit** menu, select **Properties**.

   The Edit Data Profile dialog is displayed.

3. Use the following tabs on this dialog to modify the definition of the data profile:

   - Name tab, see "Naming the Data Profile" on page 20-2
   - Select Objects tab, see "Selecting Objects" on page 20-3

- Data Locations tab, see "Data Locations Tab" on page 20-4

Data Locations Tab

The Data Locations tab specifies the location that is used as the data profile workspace. This tab contains two sections: Available Locations and Selected Locations.

The Available Locations section displays all the Oracle locations in the current repository. The Selected Locations section displays the locations that are associated with the data profile. This section can contain more than one location. The location to which the data profile is deployed is the location for which the **New Configuration Default** option is selected.

To modify the data location associated with a data profile, use the shuttle arrow to move the new location to the Selected Locations section and select the **New Configuration Default** option for this location. When you change the location for a data profile, all objects that were created in the data profile workspace schema as a result of profiling this data profile are deleted.

You can also create a new location and associate it with the data profile. Click **New** below the Selected Locations section to display the Edit Database Location dialog. Specify the details of the new location and click **OK**. The new location is added to the Selected Locations section.

You cannot configure a data profile and change the location to which it is deployed. Note that this behavior is different from other objects. For example, consider Oracle modules. The Selected Locations section on the Data Locations tab of the Edit Module dialog contains a list of possible deployment locations. The objects in the module are deployed to the location that is set using Location configuration parameter of the Oracle module.

Adding Data Objects to a Data Profile

To add data objects to a data profile, use the following steps:

1. Right-click the data profile in the Project Explorer and select **Open Editor**.

   The Data Profile Editor is displayed.

2. From the **Profile** menu, select **Add Objects**.

   The Add Profile Tables dialog is displayed. Use this dialog to add data objects to the data profile.

Add Profile Tables Dialog

The Add Profile Tables dialog contains two sections: Available and Selected. The Available section displays the data objects that are available for profiling. To add an object displayed in this list to the data profile, select the object and move it to the Selected list using the shuttle arrows. Select multiple objects by holding down the **Ctrl** key and selecting the objects.

Click **OK**. The selected data objects are added to the data profile. You can see the objects on the Profile Objects tab of the Object Tree.

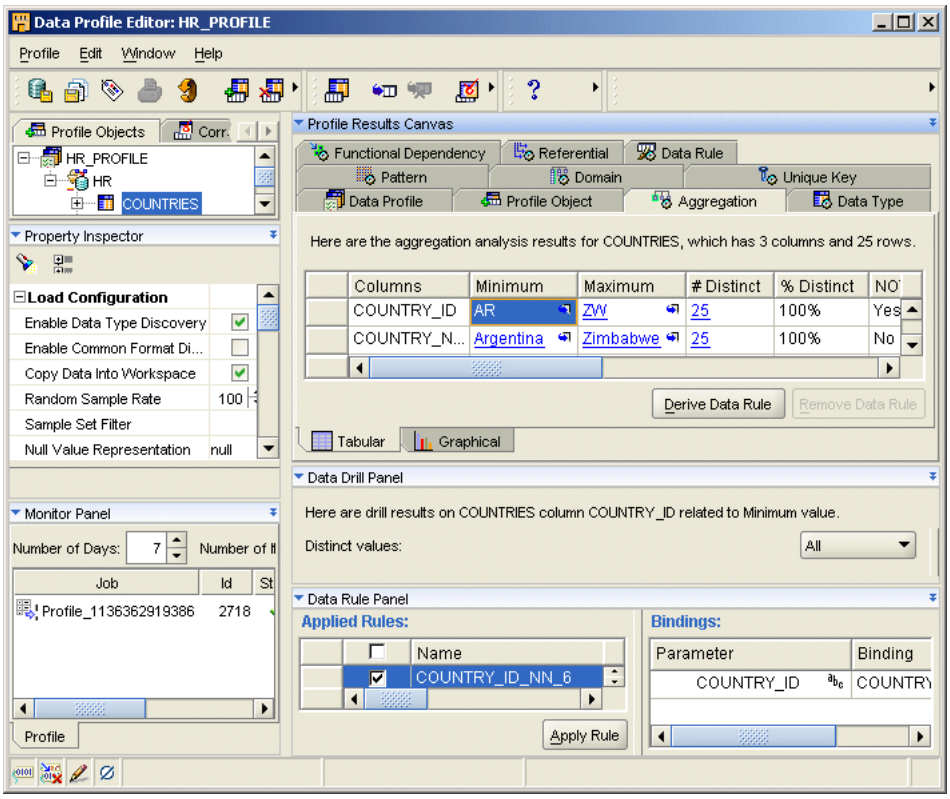
Using the Data Profile Editor

The Data Profile Editor provides a single access point for managing and viewing data profile information as well as correcting metadata and data. It combines the functionality of a data profiler, a target schema generator, and a data correction generator. As a data profiler, it enables you to perform attribute analysis and structural

analysis of selected objects. As a target schema generator, it enables you to generate a target schema based on the profile analysis and source table rules. Finally, as a data correction generator, it enables you to generate mappings and transformations to provide data correction.

Figure 20–1 displays the Data Profile Editor.

Figure 20–1 Data Profile Editor



The Data Profile Editor consists of the following:

- Menu Bar
- Toolbars
- Object Tree
- Property Inspector
- Monitor Panel
- Profile Results Canvas
- Data Drill Panel
- Data Rule Panel

Menu Bar

The menu bar provides access to the Data Profile Editor commands. The Data Profile Editor contains the following menu items.

Profile The Profile menu contains the following menu options:

- **Close:** Closes the Data Profile Editor.

- **Save All:** Saves the changes made in the Data Profile Editor.
- **Export:** Exports the data displayed in the Profile Results Canvas and the Data Drill Panel. Warehouse Builder can store the exported data in .csv or .html files. The number of files used to store the exported data depends on the amount of profiling data. For example, if you specify the name of the export file as prf_result.html, the files used to store the data begin with this file name and then continue with prf_results2.html, prf_results3.html, and so on depending upon the quantity of data.
- **Add:** Adds objects to the data profile.
- **Profile:** Profiles the objects contained in the data profile.
- **Derive Data Rule:** Derives a data rule using the result of data profiling. This option is enabled only when you select a cell that contains a hyperlink in the Profile Results Canvas.
- **Remove Data Rule:** Deletes the selected data rule.
- **Create Correction:** Creates correction objects and mappings based on the results of data profiling.
- **Print:** Prints the contents of the Data Profile Editor window.

Edit The Edit menu contains the following menu items:

- **Delete:** Deletes the object selected in the Data Profile Editor.
- **Synchronize:** Synchronizes the data profile metadata with the Warehouse Builder repository.
- **Properties:** Displays the properties of the data profile.

Window Use the Window menu to display or hide the various panels of the Data Profile Editor. All the options act as toggle switches. For example, to hide the Data Rule panel, select **Data Rule Panel** from the **Window** menu. You can then display the Data Rule panel by selecting **Data Rule Panel** from the **Window** menu.

Help Use the Help menu options to access the online Help for the product. To display the context-sensitive Help for the current window, use the **Topic** option. This menu also contains options to access the different sections of the Oracle Technology Network.

Toolbars

The Data Profile Editor allows you the flexibility of using either the menu or toolbars to achieve your goals. The toolbar provides icons for commonly used commands that are also part of the menu bar. You can perform functions such as adding objects to a data profile, profiling data, deriving data rules, and creating corrections using the toolbar icons.

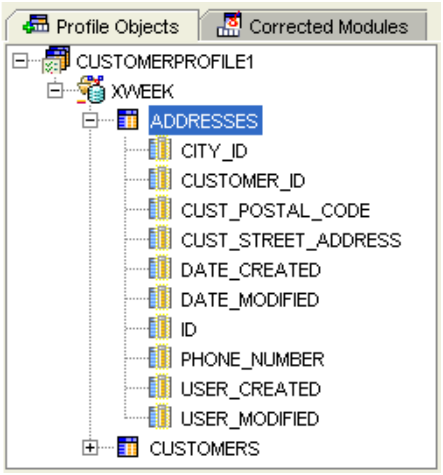
Figure 20–2 displays the various toolbars in the Data Profile Editor. When you first open the Data Profile Editor, all the toolbars are displayed in a single row. You can change their positions by holding down the left mouse button on the gray dots to the left of the toolbar, dragging, and dropping in the desired location.

Figure 20–2 Data Profile Editor Toolbars

Object Tree

The object tree can be used to navigate through the objects included in the data profile. Figure 20–3 displays the object tree that contains two tabs: Profile Objects and Corrected Modules.

Figure 20–3 Profile Objects



The Profile Objects tab contains a navigation tree that includes all of the objects selected to be profiled. The tree goes down to the attributes because you can change properties for each attribute and ensure that on an attribute, you have the correct profiling settings. When you select an object in the Profile Objects tab, the profile details about that object can be viewed in the Profile Results Canvas. You can also open the Data Object Editor for a specific object by double-clicking the object.

The Corrected Modules tab lists the objects created as a result of performing data correction actions. This tab contains objects only if you have performed data correction actions. Data correction is the process of creating corrected source data objects based on the results of the data profiling. For more information about creating corrected schemas, see "Correcting Schemas and Cleansing Data" on page 20-29.

Property Inspector

The Property Inspector enables you to define the configuration parameters for the data profiling operation. Many types of data profiling rely on the configuration parameters to set the limits and the assumptions of analysis.

For example, if the parameter Domain Discovery Max Distinct Values Count is set to 25, then if more than 25 distinct values are found in a column, no domains are considered for this column. Each type of analysis can also be turned off for performance purposes. If you run profiling without making changes to these properties, they will be set to default values.

You also use the Property Inspector to specify the types of data profiling that you want to perform. If you do not want to perform a particular type of profiling, use the configuration parameters to turn off the type of data profiling. By default, Warehouse Builder performs the following types of profiling: aggregation, data type discovery, pattern discovery, domain discovery, unique key discovery, functional dependency discovery, and row relationship discovery. Note that you cannot turn off the aggregation profiling.

For example, your data profile contains two tables called COUNTRIES and JOBS. For the COUNTRIES table, you want to perform aggregation profiling, data type profiling,

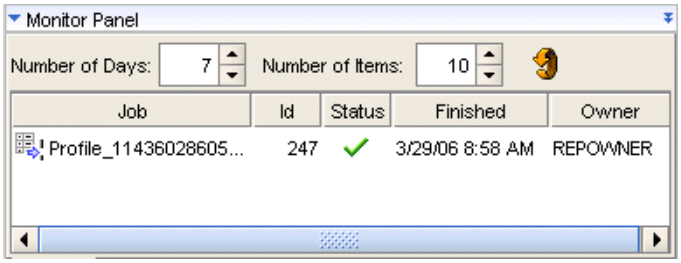
domain discovery, pattern discovery, and data rule profiling. Use the Property Inspector to select the following configuration options: Enable Data Type Discovery, Enable Domain Discovery, Enable Pattern Discovery, Enable Data rule Profiling for table.

For details about the configuration parameters and their settings, see "Configuration Parameters for Data Profiles" on page 20-11.

Monitor Panel

The Monitor panel is where you will be able to view details about currently running as well as past profiling events. The details about each profiling event includes the profile event name, profile job ID, status, timestamp, and the repository owner who executed the profiling. Figure 20–4 displays the Monitor panel.

Figure 20–4 Monitoring Profile Status

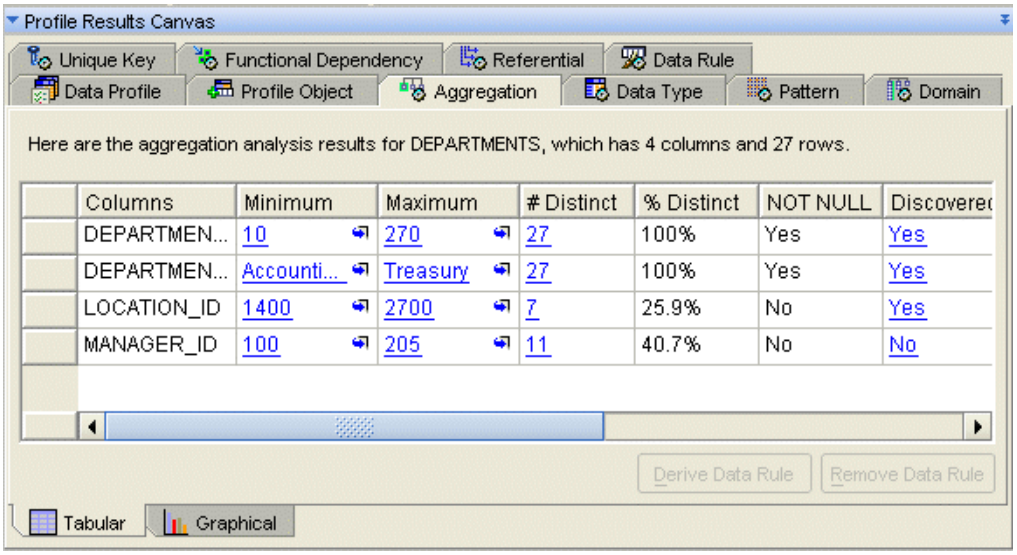


You can view more details about a job by double-clicking a job. The details include what problems were encountered ad how much time each profiling job took.

Profile Results Canvas

The Profile Results Canvas is where the results of the profiling can be viewed. Figure 20–5 displays the Profile Results Canvas panel. You can sort the results by clicking the heading of the column.

Figure 20–5 Profile Results



Click the following tabs for detailed profile analysis of different aspects of the object:

- Data Profile
- Profile Object
- Aggregation
- Data Type
- Domain
- Pattern
- Unique Key
- Functional Dependency
- Referential
- Data Rule

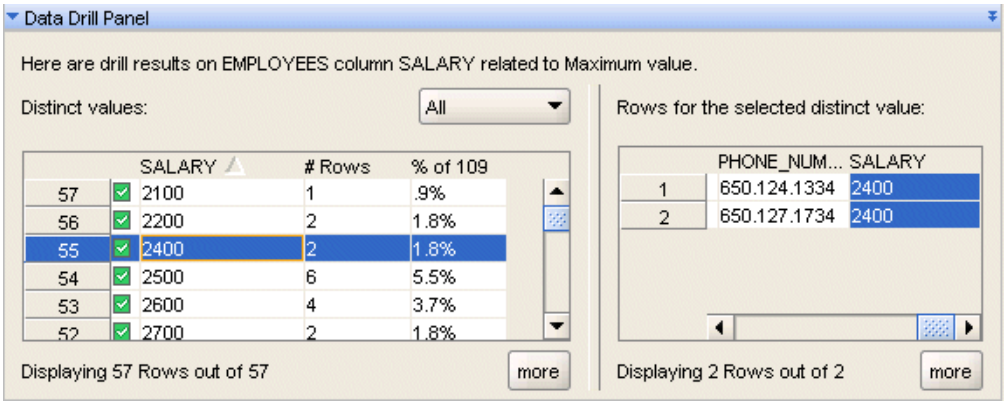
Click the hyperlinks to drill into the data. When you drill into data, the data results appear in the Data Drill panel. Depending on the tab you have selected, the available format sub-tabs may change. The sub-tabs that are available for all analysis tabs are as follows: Tabular, which provides the results in a grid or table format; Graphical, which provides an array of charts and graphs. You can also derive rules from the Profile Results Canvas.

Click any header to sort the values in ascending and descending orders interchangeably. The graphical subtab displays the graphical representation of different measures.

Data Drill Panel

Use the Data Drill panel to view questionable data from a selected object in the Profile Results canvas. Data drill-down is available for all tabs in the profile results canvas. Figure 20–6 displays the Data Drill panel for a profiled object.

Figure 20–6 Data Drill Panel



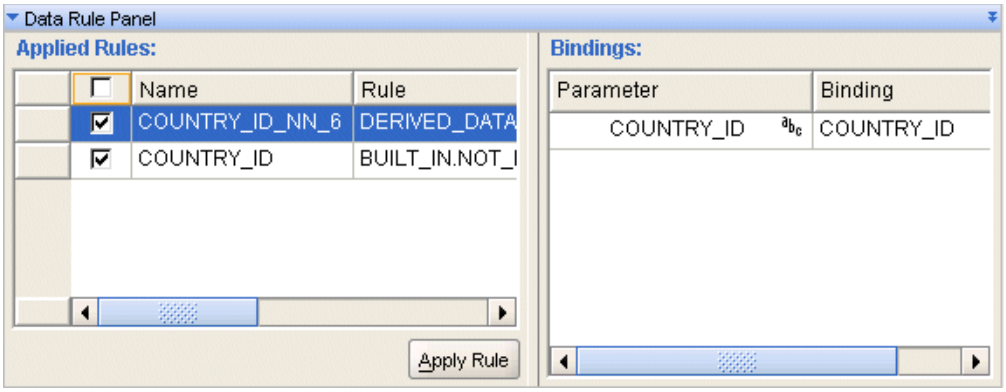
The first sentence on this panel provides information about the attribute to which the displayed drill details belong. When you drill into data, the results appear on the left side of the Data Drill panel. Use the Distinct Values list to filter the data displayed on the left side of the panel. You can further drill into this data to view the entire records. Click any row on the left side of the panel to display the selected records on the right side of the Data Drill panel.

Note that the left side displays aggregated row counts. So there are 2 rows with salary 2400. And so while the entire table contains 109 rows, the left display only shows 57 rows, namely the distinct values found.

Data Rule Panel

On the Data Rule panel, you can view the data rules that have been created as a result of the profiling. Figure 20–7 displays the Data Rule panel.

Figure 20–7 Data Rules Panel



The left side of this panel displays the data rules created for the object that is selected in the Profile Objects tab of the object tree as a result of the data profiling. When you include a table in the data profile, any data rules or check constraints on the table are automatically added. The data rules are added to the Derived_Data_Rules node under the Data Rules node of the project that contains the data profile.

You can add rules to this panel in the following ways:

- Some of the tabs on the Profile Results Canvas contain a Derive Data Rule button. This button is enabled when you select a hyperlink on the tab. Click **Derive Data Rule** to derive a data rule for the selected result.

  If the data rules has already been derived, the Remove Rule button is enabled.

- Click **Apply Rule** in the Data Rules panel. The Apply Data Rule wizard is displayed. For more information about this wizard, see "Deriving Data Rules" on page 20-27. Use this wizard to select an existing data rule and apply it to the table selected in the object tree.

You can disable any of the data rules in the Data Rule panel by unchecking the check box to the left of the data rule. To delete a data rule, right-click the gray cell to the left of the data rule and select **Delete**.

The details displayed for an applied data rule are as follows:

Name: The name of the applied data rule.

Rule: Click this field to display the name of the module that contains the data rule and the name of the data rule. Click the Ellipsis button on this field to launch the Edit Data Rule dialog that enables you to edit the data rule.

Rule Type: The type of data rule. This is not editable.

Description: The description of the applied data rule.

Configuring Data Profiles

You can configure a data profile by setting its configuration parameters in the Property Inspector of the Data Profile Editor. For more information about the properties that you can configure, see "Configuration Parameters for Data Profiles" on page 20-11.

You can set configuration parameters for a data profile at any of the following levels:

- For all objects in the data profile

  To set configuration parameters for all objects contained in the data profile, select the data profile in the Profile Objects tab of the Object Tree. In the Property Inspector, set the configuration parameters to the required values. These parameters are set for all the objects in the data profile.

- For a single object in the data profile

  To set configuration parameters for a single object within a data profile, select the object in the Profile Objects tab of the Object Tree. In the Property Inspector, set the configuration parameters.

- For an attribute in an object

  To set configuration parameters for an attribute within an object, in the Profile Objects tab of the Object Tree, expand the object node to display the attributes it contains. For example, you can expand a table node to display its columns. Select the attribute for which you want to specify configuration parameters. In the Property Inspector, set the configuration parameters.

Configuration Parameters for Data Profiles

The configuration parameters that you can set for data profiles are categorized as follows:

- Load Configuration
- Aggregation Configuration
- Pattern Discovery Configuration
- Domain Discovery Configuration
- Relationship Attribute Count Configuration
- Unique Key Discovery Configuration
- Functional Dependency Discovery Configuration
- Row Relationship Discovery Configuration
- Redundant Column Discovery Configuration
- Data Rule Profiling Configuration

Load Configuration This category contains the following parameters:

- **Enable Data Type Discovery:** Set this parameter to true to enable data type discovery for the selected table.
- **Enable Common Format Discovery:** Set this parameter to true to enable common format discovery for the selected table.
- **Copy Data into Workspace:** Set this parameter to true to enable copying of data from the source to the profile workspace.
- **Random Sample Rate:** This value represents the percent of total rows that will be randomly selected during loading.

- **Sample Set Filter:** This represents the WHERE clause that will be applied on the source when loading data into the profile workspace. Click the Ellipsis button on this field to launch the Expression Builder. Use the Expression Builder to define your WHERE clause.
- **Null Value Representation:** This value will be considered as the null value during profiling. You must enclose the value in single quotation marks. The default value is null, which is considered as a database null.

Aggregation Configuration This category consists of one parameter called **Not Null Recommendation Percentage**. If the percentage of null values in a column is less than this threshold percent, then that column will be discovered as a possible Not Null column.

Pattern Discovery Configuration This category contains the following parameters:

- **Enable Pattern Discovery:** Set this to true to enable pattern discovery.
- **Maximum Number of Patterns:** This represents the maximum number of patterns that the profiler will get for the attribute. For example, when you set this parameter to 10, it means that the profiler will get the top 10 patterns for the attribute.

Domain Discovery Configuration This category contains the following parameters:

- **Enable Domain Discovery:** Set this to true to enable domain discovery.
- **Domain Discovery Max Distinct Values Count:** The maximum number of distinct values in a column in order for that column to be discovered as possibly being defined by a domain. Domain discovery of a column occurs if the number of distinct values in that column is at or below the Max Distinct Values Count property, and, the number of distinct values as a percentage of total rows is at or below the Max Distinct Values Percent property.
- **Domain Discovery Max Distinct Values Percent:** The maximum number of distinct values in a column, expressed as a percentage of the total number of rows in the table, in order for that column to be discovered as possibly being defined by a domain. Domain Discovery of a column occurs if the number of distinct values in that column is at or below the Max Distinct Values Count property, and, the number of distinct values as a percentage of total rows is at or below the Max Distinct Values Percent property.
- **Domain Value Compliance Min Rows Count:** The minimum number of rows for the given distinct value in order for that distinct value to be considered as compliant with the domain. Domain Value Compliance for a value occurs if the number of rows with that value is at or above the Min Rows Count property, and, the number of rows with that value as a percentage of total rows is at or above the Min Rows Percent property.
- **Domain Value Compliance Min Rows Percent:** The minimum number of rows, expressed as a percentage of the total number of rows, for the given distinct value in order for that distinct value to be considered as compliant with the domain. Domain Value Compliance for a value occurs if the number of rows with that value is at or above the Min Rows Count property, and, the number of rows with that value as a percentage of total rows is at or above the Min Rows Percent property.

Relationship Attribute Count Configuration This category contains one parameter called **Maximum Attribute Count**. This is the maximum number of attributes for unique key, foreign key, and functional dependency profiling.

Unique Key Discovery Configuration This category contains the following parameters:

- **Enable Unique Key Discovery:** Set this parameter to true to enable unique key discovery.
- **Minimum Uniqueness Percentage:** This is the minimum percentage of rows that need to satisfy a unique key relationship.

Functional Dependency Discovery Configuration This category contains the following parameters:

- **Enable Functional Dependency Discovery:** Set this parameter to true to enable functional dependency discovery.
- **Minimum Functional Dependency Percentage:** This is the minimum percentage of rows that need to satisfy a functional dependency relationship.

Row Relationship Discovery Configuration This category contains the following parameters:

- **Enable Relationship Discovery:** Set this parameter to true to enable foreign key discovery.
- **Minimum Relationship Percentage:** This is the minimum percentage of rows that need to satisfy a foreign key relationship.

Redundant Column Discovery Configuration This category contains the following parameters:

- **Enable Redundant Columns Discovery:** Set this parameter to true to enable redundant column discovery with respect to a foreign key-unique key pair.
- **Minimum Redundancy Percentage:** This is the minimum percentage of rows that are redundant.

Data Rule Profiling Configuration This category contains one parameter called **Enable Data Rule Profiling for Table**. Set this parameter to true to enable data rule profiling for the selected table. This setting is only applicable for a table, not for an individual attribute.

Profiling the Data

After you have created a data profile in the navigation tree, you can open it in the Data Profile Editor and profile the data. The objects you selected when creating the profile are displayed in the object tree of the Data Profile Editor. You can add objects to the profile by selecting **Profile** and then **Add**.

To profile the data:

1. Expand the Data Profiles node on the navigation tree, right-click a data profile, and select **Open Editor**.

   The Data Profile Editor opens the selected data profile.

2. From the **Profile** menu, select **Profile**.

If this is the first time you are profiling data, the Data Profile Setup dialog is displayed. Enter the details of the profiling workspace in this dialog. For more information see, "Data Profile Setup Dialog" on page 20-14.

Warehouse Builder begins preparing metadata for profiling. The progress window appears and displays the name of the object that Warehouse Builder is creating in order to profile the data. After the metadata preparation is complete, the Profiling Initiated dialog is displayed informing you that the profiling job has started. Once the profiling job starts, the data profiling is asynchronous and you can continue working in Warehouse Builder or even close the client. Your profiling process will continue to run until complete.

3. View the status of the profiling job in the Monitor Panel of the Data Profile Editor.

   You can continue to monitor the progress of your profiling job in the Monitor panel. After the profiling job is complete, the status displays as complete.

4. After the profiling is complete, the Retrieve Profile Results dialog is displayed and you are prompted to refresh the results.

   You can use this option if you have previously profiled data in the same data profile. It enables you to control when the new profiling results become visible in the Data Profile Editor.

   Note: Data profiling results are overwritten on subsequent profiling executions.

Data Profile Setup Dialog

The Data Profile Setup dialog contains details about the data profiling workspace. The profiling workspace is a schema that Warehouse Builder uses to store the results of the profiling. You set the profiling workspace once for each repository.

When the Data Profile Setup dialog is first displayed, it contains two fields.

SYSDBA Name: The name of a database user who has SYSDBA privileges.

SYSDBA Password: The password of the user specified in the SYSDBA Name field.

This information is required by Warehouse Builder to create the data profiling workspace schema. If you click **OK**, Warehouse Builder uses default names and creates a database schema to store data related to the data profiling activity. Additionally, a location that is associated with this schema is created and registered. For example, if the name of the repository owner is `rep_own`, Warehouse Builder creates a schema called `rep_own_prf` and a location called `rep_own_prf_location`.

To use your own name for the data profiling schema and location, click **Show Details**. The following fields are displayed in the Data Profile Setup dialog.

Name: The name of the schema that will be used as the profiling workspace.

Password: The password for the schema.

Confirm Password: Confirm the password for the schema.

Use the **Default** and **Temporary** drop-down lists to select the default tablespace and the temporary tablespace for the data profiling schema. Click **OK** to create the schema and the location associated with this schema. For example, if you specify the schema name as `profile_workspace`, a location called `profile_workspace_location` is created and associated with this schema.

Tip: It is highly recommended to use a different tablespace and use a separate database file for this tablespace.

Warehouse Builder uses the location created in this step as the default location for all data profiles. You can modify the default data profiling location either globally or for a particular data profile. To change the default profile location at a global level, select **Preferences** from the **Tools** menu of the Design Center. Select the new location in the Default Profile Location preference under the Data Profiling node. To change the data profile location for a data profile, see "Data Locations Tab" on page 20-4.

Viewing the Results

After the profile operation is complete, you can open the data profile in the Data Profile Editor to view and analyze the results.

To view the profile results:

1. Select the data profile in the navigation tree, right-click, and select **Open Editor**.

   The Data Profile Editor opens and displays the data profile.

2. If you have previous data profiling results displayed in the Data Profile Editor, refresh the view when prompted so that the latest results are shown.

   The results of the profiling are displayed in the Profile Results Canvas.

3. Minimize the Data Rule and Monitor panels by clicking on the arrow symbol in the upper left corner of the panel.

   This maximizes your screen space.

4. Select objects in the Profile Objects tab of the object tree to focus the results on a specific object.

   The results of the selected object are displayed in the Profile Results Canvas. You can switch between objects. The tab that you had selected for the previous object remains selected.

The following results are available in the Profile Results Canvas:

- Data Profile
- Profile Object
- Aggregation
- Data Type
- Domain
- Pattern
- Unique Key
- Functional Dependency
- Referential
- Data Rule

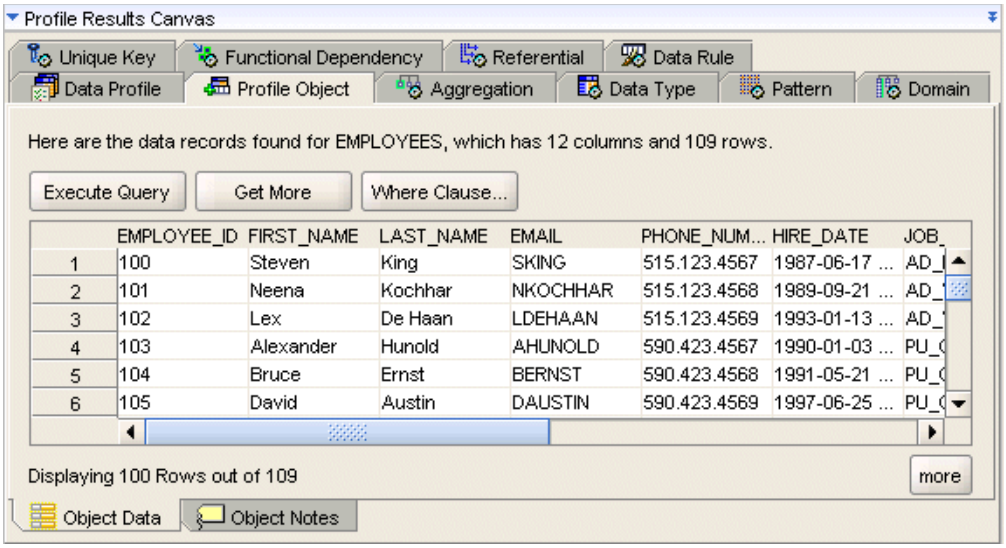
Data Profile

The Data Profile tab contains general information about the data profile. Use this tab to store any notes or information about the data profile.

Profile Object

The Profile Object tab, shown in Figure 20–8, contains two sub-tabs: Object Data and Object Notes. The Object Data tab lists the data records in the object you have selected in the Profile Objects tab of the object tree. The number of rows that were used in the sample are listed. You can use the buttons along the top of the tab to execute a query, get more data, or add a `WHERE` clause.

Figure 20–8 Profile Object Tab



Aggregation

The Aggregation tab, shown in Figure 20–9, displays all the essential measures for each column, such as minimum and maximum values, number of distinct values, and null values. Some measures are only available for specific data types. Those include the average, median, and standard deviation measures. Information can be viewed from either the Tabular or Graphical sub-tabs. Table 20–1 describes the various measurement results available in the Aggregation tab.

Table 20–1 Aggregation Results

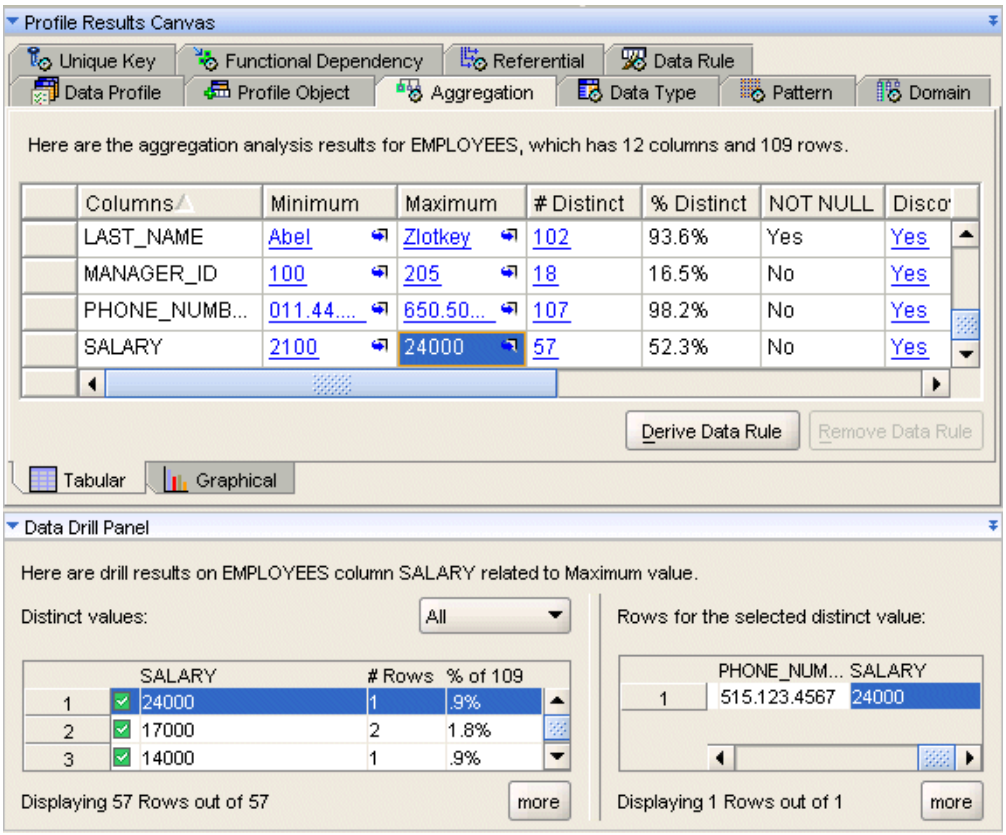
Measurement	Description
Minimum	The minimum value with respect to the inherent database ordering of a specific type
Maximum	The maximum value with respect to the inherent database ordering of a specific type
# Distinct	The total number of distinct values for a specific attribute
% Distinct	The percentage of distinct values over the entire row set number
Not Null	Yes or No
Recommended NOT NULL	From analyzing the column values, data profiling determines that this column should not allow null values
# Null	The total number of null values for a specific attribute
%Null	The percentage of null values over the entire row set number
Six Sigma	For each column, number of null values (defects) to the total number of rows in the table (opportunities).

Table 20–1 (Cont.) Aggregation Results

Measurement	Description
Average	The average value for a specific attribute for the entire row set
Median	The median value for a specific attribute for the entire row set
Std Dev	The standard deviation for a specific attribute.

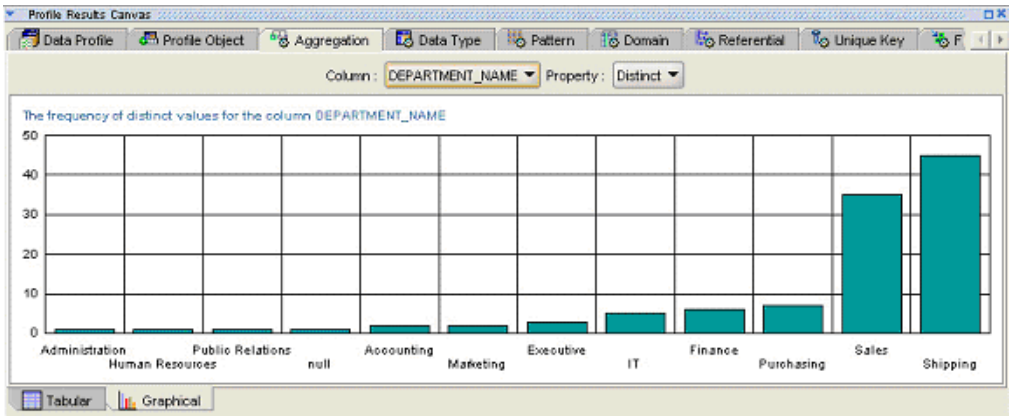
A hyperlinked value in the aggregation results grid indicates that you can click the value to drill down into the data. This enables you to analyze the data in the sample that produced this result. For example, if you scroll to the SALARY column, shown in Figure 20–9, and click the value in the Maximum cell showing 24000, the Data Drill Panel on the bottom changes to show you all the distinct values in this column with counts on the left. On the right, the Data Drill can zoom into the value you select from the distinct values and display the full record where these values are found.

Figure 20–9 Aggregation Tabular Results



The graphical analysis, as shown in Figure 20–10, displays the results in a graphical format. You can use the graphical toolbar to change the display. You can also use the Column and Property drop-down menus to change the displayed data object.

Figure 20–10 Aggregation Graphical Results



Data Type

The Data Type tab, shown in Figure 20–11, provides profiling results about data types. This includes metrics such as length for character data types and the precision and scale for numeric data types. For each data type that is discovered, the data type is compared to the dominant data type found in the entire attribute and the percentage of rows that comply with the dominant measure is listed.

Figure 20–11 Data Type Results

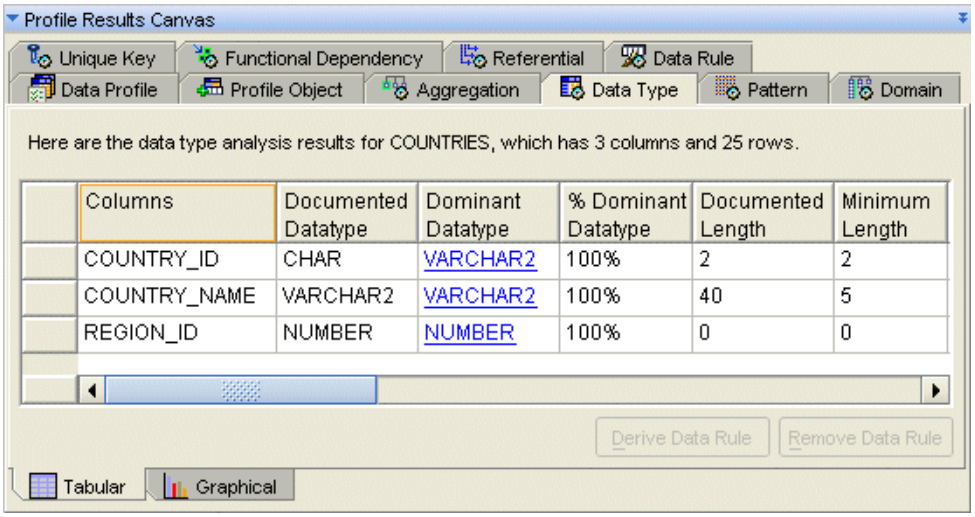


Table 20–2 describes the various measurement results available in the Data Type tab.

Table 20–2 Data Type Results

Measurement	Description
Columns	Name of the column
Documented Data Type	Data type of the column in the source object
Dominant Data Type	From analyzing the column values, data profiling determines that this is the dominant (most frequent) data type.
Percent Dominant Data Type	Percentage of total number of rows where column value has the dominant data type

Table 20–2 (Cont.) Data Type Results

Measurement	Description
Documented Length	Length of the data type in the source object
Minimum Length	Minimum length of the data stored in the column
Maximum Length	Maximum length of data stored in the column
Dominant Length	From analyzing the column values, data profiling determines that this is the dominant (most frequent) length.
% Dominant Length	Percentage of total number of rows where column value has the dominant length
Documented Precision	Precision of the data type in the source object
Minimum Precision	Minimum precision for the column in the source object
Maximum Precision	Maximum precision for the column in the source object
Dominant Precision	From analyzing the column values, data profiling determines that this is the dominant (most frequent) precision.
% Dominant Precision	Percentage of total number of rows where column value has the dominant precision
Documented Scale	Scale specified for the data type in the source object
Minimum Scale	Minimum scale of the data type in the source object
Maximum Scale	Maximum scale of the data type in the source object
Dominant Scale	From analyzing the column values, data profiling determines that this is the dominant (most frequent) scale.
% Dominant Scale	Percentage of total number of rows where column value has the dominant scale

One example of data type profiling would be finding that a column defined as VARCHAR that actually only had numeric values stored. Changing the data type of the column to NUMBER would make storage and processing more efficient.

Domain

The Domain tab, shown in Figure 20–12, displays results about the possible set of values that exist in a certain attribute. Information can be viewed from either the Tabular or Graphical sub-tabs.

Figure 20–12 Domain Discovery Results

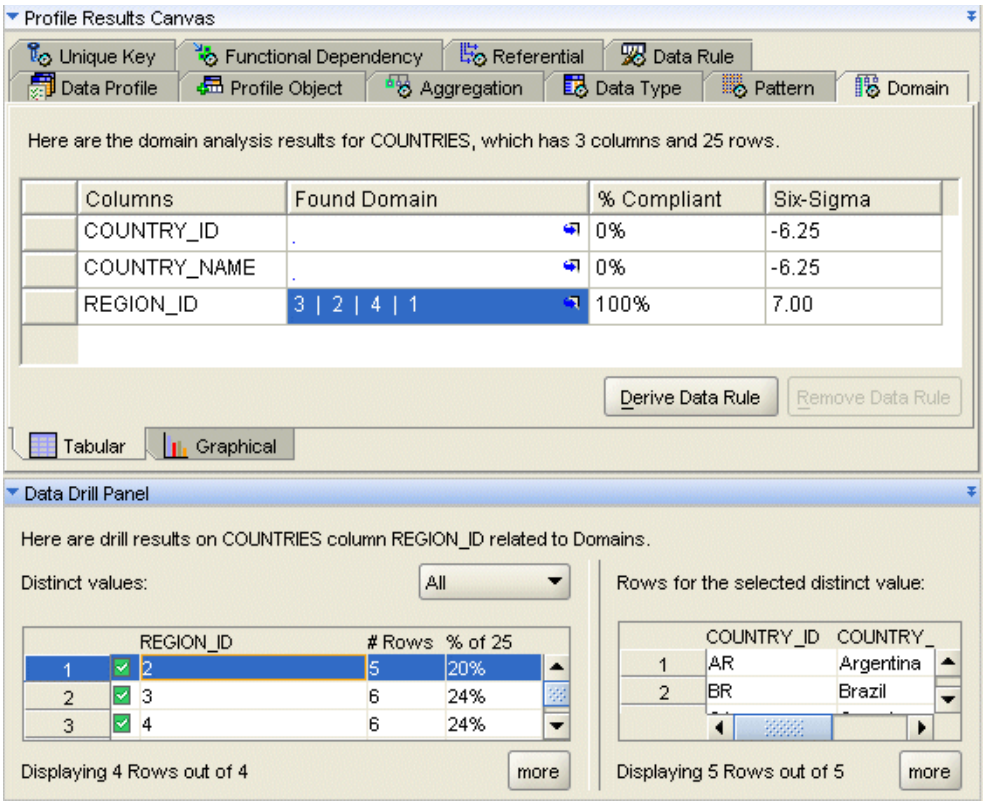


Table 20–3 describes the various measurement results available in the Domain tab.

Table 20–3 Domain Results

Measurement	Description
Found Domain	The discovered domain values
% Compliant	The percentage of all column values that are compliant with the discovered domain values
Six Sigma	The Six-Sigma value for the domain results

The process of discovering a domain on a column involves two phases. First, the distinct values in the column are used to determine whether that column might be defined by a domain. Typically, there are few distinct values in a domain. Then, if a potential domain is identified, the count of distinct values is used to determine whether that distinct value is compliant with the domain. The properties that control the threshold for both phases of domain discovery can be set in the Property Inspector.

If you find a result that you want to know more about, drill down and use the Data Drill panel to view details about the cause of the result.

For example, a domain of four values was found for the column REGION_ID: 3,2,4, and 1. If you want to see which records contributed to this finding, select the REGION_ID row and view the details in the Data Drill panel.

Pattern

The Pattern tab displays information discovered about patterns within the attribute. Pattern discovery is the profiler's attempt at generating regular expressions for data it

discovered for a specific attribute. Note that non-English characters are not supported in the pattern discovery process.

Table 20–4 describes the various measurement results available in the Pattern tab.

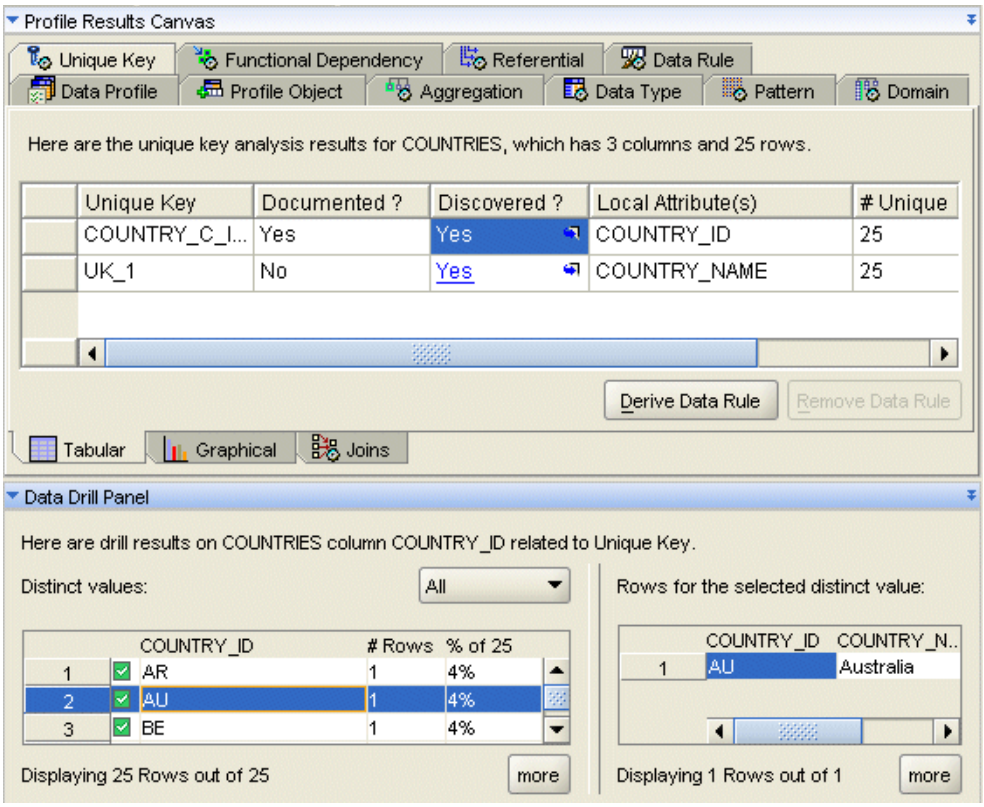
Table 20–4 Pattern Results

Measurement	Description
Dominant Character Pattern	The most frequently discovered character pattern or consensus pattern.
% Compliant	The percentage of rows whose data pattern agree with the dominant character pattern
Dominant Word Pattern	The most frequently discovered word character pattern or consensus pattern
& Compliant	The percentage of rows whose data pattern agree with the dominant word pattern
Common Format	Name, Address, Date, Boolean, Social Security Number, E-mail, URL. This is the profiler's attempt to add semantic understanding to the data that it sees. Based on patterns and some other techniques, it will try to figure out which domain bucket a certain attribute's data belongs to.
% Compliant	The percentage of rows whose data pattern agree with the consensus common format pattern

Unique Key

The Unique Key tab, shown in Figure 20–13, provides information about the existing unique keys that were documented in the data dictionary, and possible unique keys or key combinations that were detected by the data profiling operation. The uniqueness % is shown for each. The unique keys that have No in the Documented? column are the ones that are discovered by data profiling. For example, the unique key UK_1 was discovered as a result of data profiling, whereas COUNTRY_C_ID_PK exists in the data dictionary.

Figure 20–13 Unique Key Results



For example, a phone number is unique in 98% of the records. It can be a unique key candidate, and you can then use Warehouse Builder to cleanse the noncompliant records for you. You can also use the drill-down feature to view the cause of the duplicate phone numbers in the Data Drill panel. Table 20–5 describes the various measurement results available in the Unique Key tab.

Table 20–5 Unique Key Results

Measurement	Description
Unique Key	The discovered unique key
Documented?	Yes or No. Yes indicates that a unique key on the column exists in the data dictionary. No indicates that the unique key was discovered as a result of data profiling.
Discovered?	From analyzing the column values, data profiling determines whether a unique key should be created on the columns in the Local Attribute(s) column.
Local Attribute(s)	The name of the column in the table that was profiled
# Unique	The number of rows, in the source object, in which the attribute represented by Local Attribute is unique
% Unique	The percentage of rows, in the source object, for which the attribute represented by Local Attribute are unique
Six Sigma	For each column, number of null values (defects) to the total number of rows in the table (opportunities)

Functional Dependency

The Functional Dependency tab, shown in Figure 20–14, displays information about the attribute or attributes that seem to depend on or determine other attributes. Information can be viewed from either the Tabular or Graphical sub-tabs. You can use the **Show** drop-down list to change the focus of the report. Note that unique keys defined in the database are not discovered as functional dependencies during data profiling.

Figure 20–14 Functional Dependency

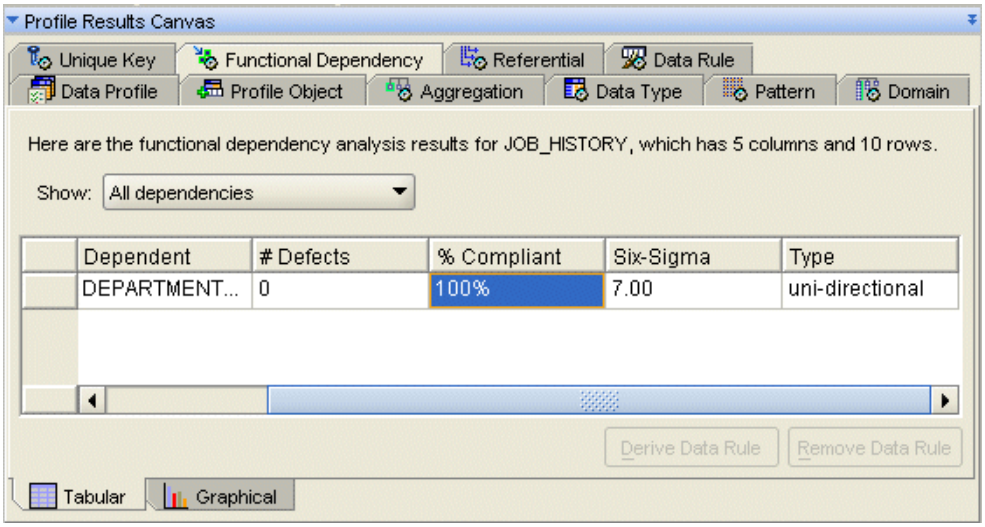


Table 20–6 describes the various measurement results available in the Functional Dependency tab.

Table 20–6 Functional Dependency Results

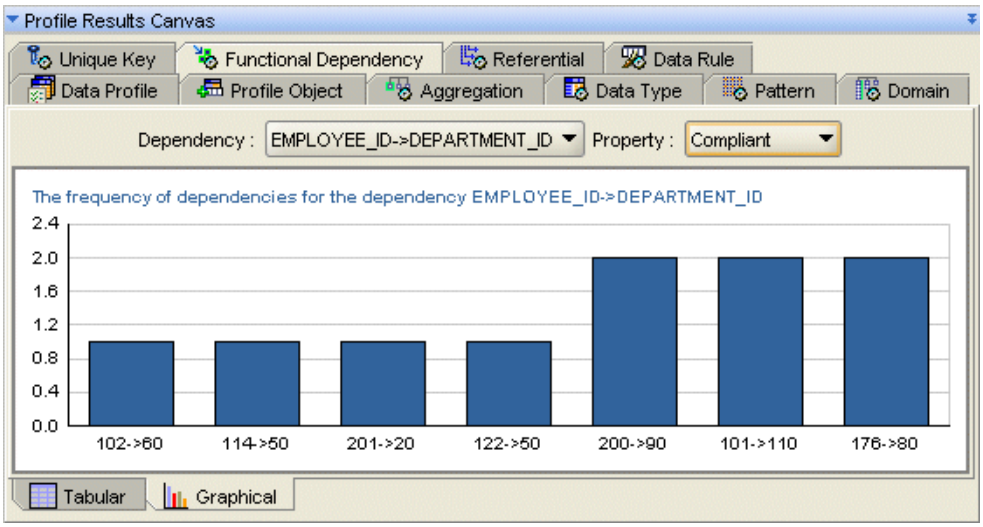
Measurement	Description
Determinant	The name of the attribute that is found to determine the attribute listed in the Dependent
Dependent	The name of the attribute found to be determined by value of another attribute
# Defects	The number of values in the Determinant attribute that were not determined by the Dependent attribute
% Compliant	The percentage of values that met the discovered dependency
Six Sigma	The six sigma value
Type	The suggested action for the discovered dependency

For example, if you select Only 100% dependencies from the **Show** drop down list, the information shown is limited to absolute dependencies. If you have an attribute that is always dependent on another attribute, shown in Figure 20–14, Warehouse Builder can suggest that it be a candidate for a reference table. Suggestions are shown in the Type column. Removing the attribute into a separate reference table normalizes the schema.

The Functional Dependency tab also has a Graphical subtab so that you can view information graphically. You can select a dependency and property from the drop down lists to view graphical data.

For example, in Figure 20–15, you select the dependency where DEPARTMENT_ID seems to determine COMMISSION_PCT. (DEPARTMENT_ID->COMMISSION_PCT). In the majority of cases, COMMISION_PCT is null. Warehouse Builder therefore determines that most DEPARTMENT_ID values determine COMMISION_PCT to be null. By switching the Property to Non-Compliant, you can view the exceptions to this discovery. Figure 20–15 shows that for the DEPARTMENT_ID values of 80, the COMMISION_PCT values are not null. This makes sense after you discover that the department with DEPARTMENT_ID 80 is Sales department.

Figure 20–15 Graphical Functional Dependency



Referential

The Referential tab displays information about foreign keys that were documented in the data dictionary, as well as relationships discovered during profiling. For each relationship, you can see the level of compliance. Information can be viewed from both the Tabular and Graphical subtabs. In addition, two other subtabs are available only in the Referential tab: Joins and Redundant Columns. Table 20–7 describes the various measurement results available in the Referential tab.

Table 20–7 Referential Results

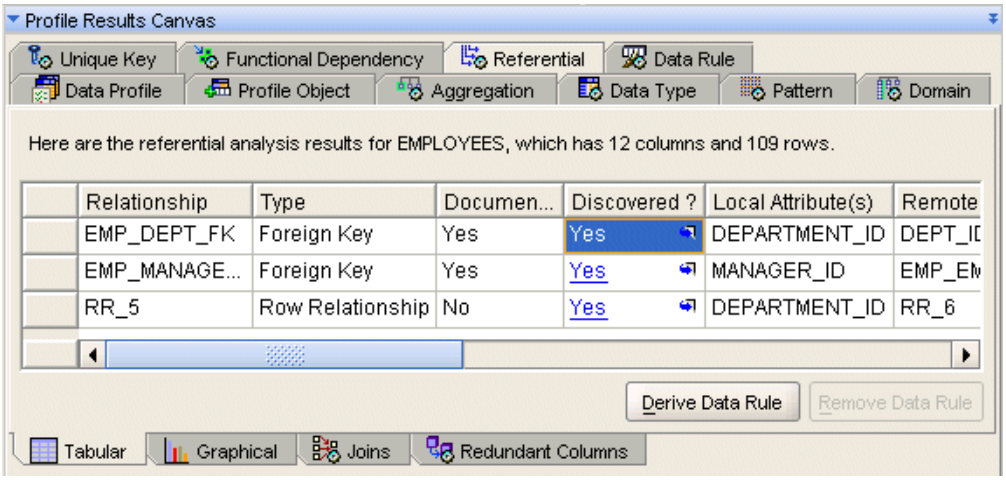
Measurement	Description
Relationship	The name of the relationship
Type	The type of relationship
Documented?	Yes or No. Yes indicates that a foreign key on the column exists in the data dictionary. No indicates that the foreign key was discovered as a result of data profiling.
Discovered?	From analyzing the column values, data profiling determines whether a foreign key should be created on the column represented by Local Attribute(s).
Local Attribute(s)	The name of the attribute in the source object
Remote Key	The name of the key in the referenced object to which the local attribute refers
Remote Attribute(s)	The name of the attributes in the referenced object
Remote Relation	The name of the object that the source object references

Table 20–7 (Cont.) Referential Results

Measurement	Description
Remote Module	The name of the module that contains the references object
Cardinality Range	The range of the cardinality between two attributes For example, the EMP table contains 5 rows of employees data. There are two employees each in department 10 and 20 and one employee in department 30. The DEPT table contains three rows of department data with deptno value 10, 20, and 30. Data profiling will find a row relationship between the EMP and DEPT tables. The cardinality range will be 1-2:1-1. This is because in EMP, the number of rows in each distinct value ranges from 1 (for deptno 30) to 2 (deptno 10 and 20). In DEPT, there is only one row for each distinct value (10, 20, and 30)
# Orphans	The number of orphan rows in the source object
% Compliant	The percentage of values that met the discovered dependency
Six Sigma	For each column, number of null values (defects) to the total number of rows in the table (opportunities)

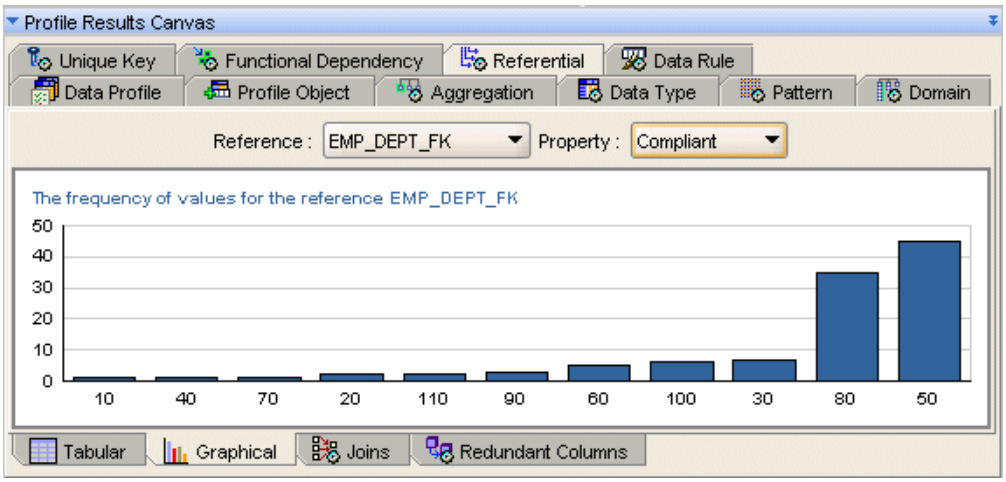
For example, you are analyzing two tables for referential relationships: the Employees table and the Departments table. Using the Referential data profiling results shown in Figure 20–16, you discover that the DEPARTMENT_ID column in the Employees table is found to be related to DEPARTMENT_ID column in the Departments table 98% of the time. You can then click the hyperlinked Yes in the Discovered? column to view the rows that did not comply with the discovered foreign key relationship.

Figure 20–16 Referential Results



You can also select the Graphical subtab to view information graphically. This view is effective for seeing noncompliant records such as orphans. To use the Graphical subtab, make a selection from the Reference and Property drop-down lists. Figure 20–17 shows graphical representation showing records in the Employees table.

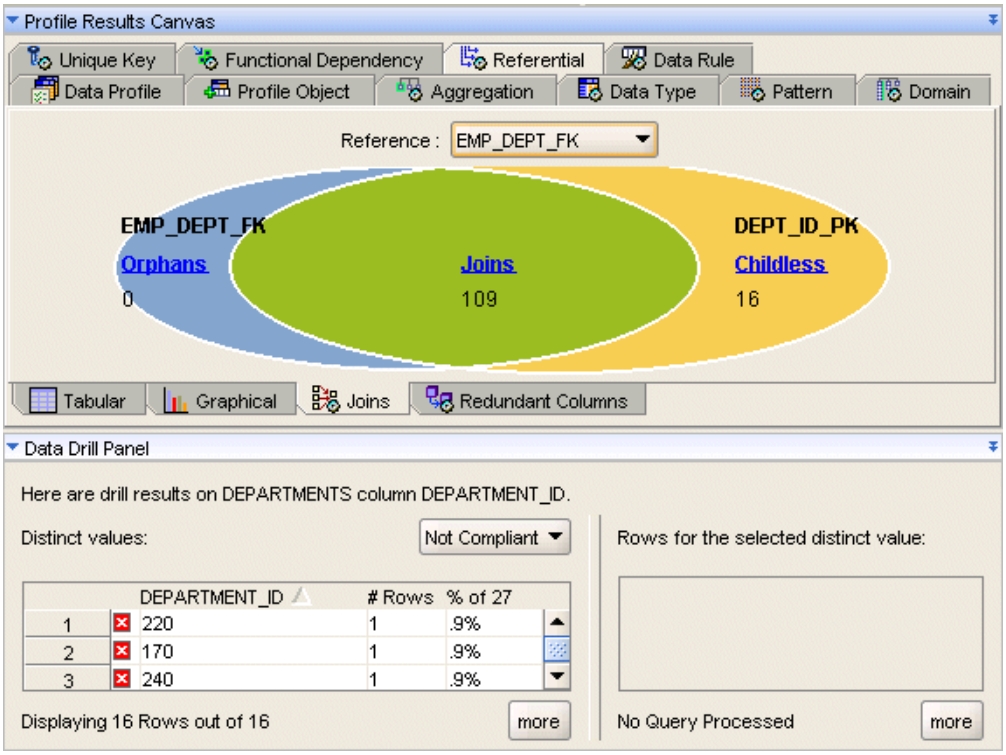
Figure 20–17 Graphical Referential Results



The Joins subtab displays a join analysis on the reference selected in the Reference drop down list. The results show the relative size and exact counts of the three possible outcomes for referential relationships: joins, orphans, and childless objects.

For example, both the EMPLOYEES and DEPARTMENTS tables contain a DEPARTMENT_ID column. There is a one-to-many relationship between the DEPARTMENT_ID column in the DEPARTMENTS table and the DEPARTMENT_ID column in the EMPLOYEES table. The Joins represent the values that have values in both tables. Orphans represent values that are only present in the EMPLOYEES table and not the DEPARTMENTS table. And finally, Childless values are present in the DEPARTMENTS table and not the EMPLOYEES table. You can drill into values on the diagram to view more details in the Data Drill panel, as shown in Figure 20–18.

Figure 20–18 Join Results



The Redundant Columns subtab displays information about columns in the child table that are also contained in the primary table. Redundant column results are only available when perfectly unique columns are found during profiling.

For example, two tables EMP and DEPT, shown in Table 20–8 and Table 20–9, having the following foreign key relationship: EMP.DEPTNO (uk) = DEPT.DEPTNO (fk).

Table 20–8 EMP Table

Employee Number	Dept. No	Location
100	1	CA
200	2	NY
300	3	MN

Table 20–9 DEPT Table

Dept No	Location	Zip
1	CA	94404
3	MN	21122
3	MN	21122
1	CA	94404

In this example, the Location column in the EMP table is a redundant column because you can get the same information from the join.

Data Rule

The Data Rule tab displays the data rules that are defined as a result of data profiling for the table selected in the object tree. The details for each data rule include the following:

- **Rule Name:** Represents the name of the data rule.
- **Rule Type:** Provides a brief description about the type of data rule.
- **Origin:** Represents the origin of the data rule. For example, a value of Derived indicates that the data rule is derived.
- **% Compliant:** Percent of rows that comply with the data rule.
- **# Defects:** Number of rows that do not comply with the data rule.

The data rules on this tab reflect the active data rules in the Data Rule panel. You do not directly create data rules on this tab.

Deriving Data Rules

At this stage in the data profiling process, you can begin to tune the data profiling results by deriving data rules that can be used to clean up your data. A data rule is an expression that determines the set of legal data that can be stored within a data object. Data rules also determine the set of legal relationships that can exist between data objects. Although you can create data rules and apply them manually to your data profile, derived data rules allow you to move quickly and seamlessly between data profiling and data correction. For more information about how to create data rules, see "Using Data Rules" on page 20-35.

For example, you have a table called `Employees` with the following columns: `Employee_Number`, `Gender`, `Employee_Name`. The profiling result shows that 90% of the values in the `Employee_Number` column are unique, making it an excellent candidate for a unique key. The results also show that 85% of the values in the `Gender` column are either 'M' or 'F', making it also a good candidate for a domain. You can then derive these rules directly from the Profile Results Canvas.

To derive a data rule:

1. Select a data profile in the navigation tree, right-click, and select **Open Editor**.

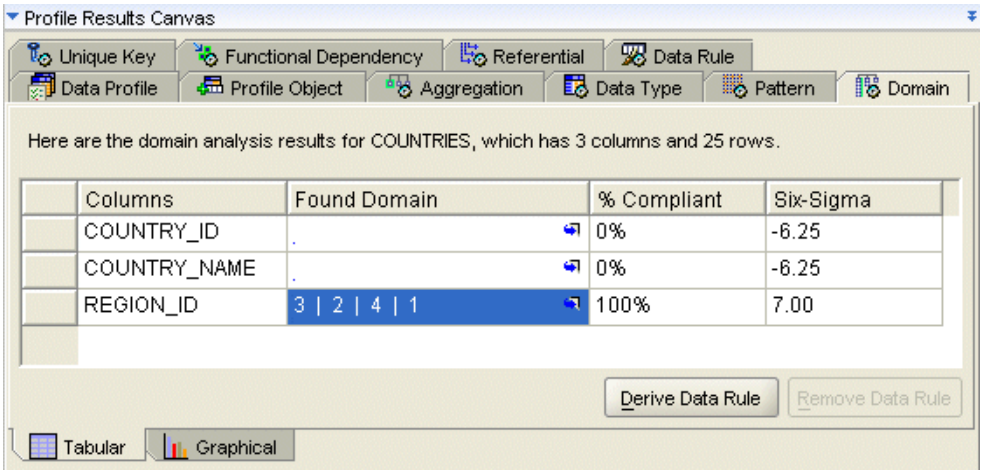
   The Data Profile Editor is displayed with the profiling results.

2. Review the profiling results and determine which findings you want derived into data rules.

   The types of results that warrant data rules vary. Some results commonly derived into data rules include a detected domain, a functional dependency between two attributes, or a unique key.

3. Select the tab that displays the results from which you want to derive a data rule.

Figure 20–19 Tab Selected for Deriving Data Rule



   The Domain tab is selected in Figure 20–19. Data profiling has detected a domain with 100% compliance for the Region_ID attribute.

4. Select the cell that contains the results you want derived into a data rule and then from the **Profile** menu select **Derive Data Rule**. Or click the **Derive Data Rule** button.

   The Derive Data Rule Wizard opens and displays the Welcome page.

5. Click **Next**.

   The Name and Description page is displayed.

6. The **Name** field displays a default name for the data rule. You can either accept the default name or enter a new name.

7. Click **Next**.

   The Define Rule page is displayed.

8. Provide details about the data rule parameters.

The Type field that represents the type of data rule is populated based on the tab from which you derived the data rule. You cannot edit the type of data rule.

Additional fields in the lower portion of this page define the parameters for the data rule. Some of these fields are populated with values based on the result of data profiling. The number and type of fields depends on the type of data rule.

9. Click **Next**.

   The Summary page is displayed. Review the options you set in the wizard using this page. Click **Back** if you want to change any of the selected values.

10. Click **Finish**.

    The data rule is created and it appears in the Data Rule panel of the Data Profile Editor. The derived data rule is also added to the Derived_Data_Rules node under Data Rules node in the Project Explorer. You can reuse this data rule by attaching it to other data objects.

Correcting Schemas and Cleansing Data

After deriving the data rules, you can use Warehouse Builder to automate the process of correcting source data based on the data profiling results. As part of the correction process, Warehouse Builder creates the following:

- Corrected tables that adhere to the newly derived data rules
- Correction mappings that you use to cleanse the data

  The correction mappings enforce the data rules. While moving data from the old "dirty" tables in the profile source tables into the corrected tables, these mappings correct records that do not comply with the data rules.

These objects are defined in Warehouse Builder. To implement them in your target schema, you must deploy the correction tables and correction mappings. Before you deploy a correction mapping, ensure that you do the following:

- Deploy the correction tables created as a result of data profiling. The correction tables have names that are prefixed with TMP__. For example, when you profile the `EMPLOYEES` table, the correction table will be called `TMP__EMPLOYEES`.
- Grant the `SELECT` privilege on the source tables to `PUBLIC`. For example, your correction mapping contains the table `EMPLOYEES` from the `HR` schema. You can successfully deploy this correction mapping only if the `SELECT` privilege if granted to `PUBLIC` on `HR.EMPLOYEES` table.

Creating Corrections

To create corrections, from the Data Profile Editor, select **Profile** and then **Create Correction**. The Create Correction Wizard opens and displays the Welcome page. Click **Next**. The wizard guides you through the following pages:

- Select Target Module on page 20-30
- Select Objects on page 20-30
- Select Data Rules and Data Types on page 20-30
- Data Rules Validation on page 20-31
- Verify and Accept Corrected Tables on page 20-31
- Choose Data Correction Actions on page 20-32

- Summary on page 20-34

Select Target Module

Use the Select Target Module page to specify the target module in which the corrected objects are stored. You can create a new target module or you can select an existing module.

To use an existing module to store the corrected objects, choose **Select an Existing Module**. Select **Create a New Target Module** to create a new module. This launches the Create Module Wizard.

Select Objects

Use the Select Objects page to select the objects you want to correct.

The Filter drop-down list enables you to filter the objects that are available for selection. The default selection is All Objects. You can display only particular types of data objects such as tables or views.

The Available section lists all the objects available for correction. Each type of object is represented by a node. Within this node, the objects are grouped by module. Select the objects that you want to correct and move them to the Selected section.

Select Data Rules and Data Types

Use this page to select the data rules that should be applied to the selected objects. The objects selected for correction are on the left side of the page and are organized into a tree by modules. The right panel contains two tabs: Data Rules and Data Types.

Data Rules

The Data Rules tab displays the available data rules for the object selected in the object tree. Specify the data rules that you want to apply to that corrected object by selecting the check box to the left of the data rule. Warehouse Builder uses these data rules to create constraints on the tables during the schema generation.

The Bindings section contains details about the table column to which the rule is bound. Click a rule name to display the bindings for the rule.

The method used to apply the data rule to the correction table depends on the type of data rule you are implementing. Warehouse Builder uses the following methods of object schema correction:

- Creating constraints

  Creates a constraint reflecting the data rule on the correction table. If a constraint cannot be created, a validation message is displayed on the Data Rules Validation page.

  Constraints are created for the following types of rules: Custom, Domain List, Domain Pattern List, Domain Range, Common Format, No Nulls, and Unique Key.

- Changing the data type

  Changes the data type of the column to NUMBER or DATE according to the results of profiling. The data type is changed for data rules of type IS Number and Is Name.

- Creating a lookup table

Creates a lookup table and adds the appropriate foreign key or unique key constraints to the corrected table and the lookup table. Warehouse Builder creates a lookup table for a Functional Dependency rule.

- Name and Address parse

  Adds additional name and address attributes to the correction table. The name and address attributes correspond to a selection of the output values of the Name and Address operator. In the map that is created to cleanse data, a Name and Address operator is used to perform name and address cleansing.

  A name and address parse is performed for a data rule of type Name and Address.

Data Types

The Data Types tab displays the columns that are selected for correction. The change could be a modification of the data type, precision, or from fixed-length to variable-length. The Documented Data Type column displays the existing column definition and the New Data Type column displays the proposed correction to the column definition.

To correct a column definition, select the check box to the left of the column name.

Data Rules Validation

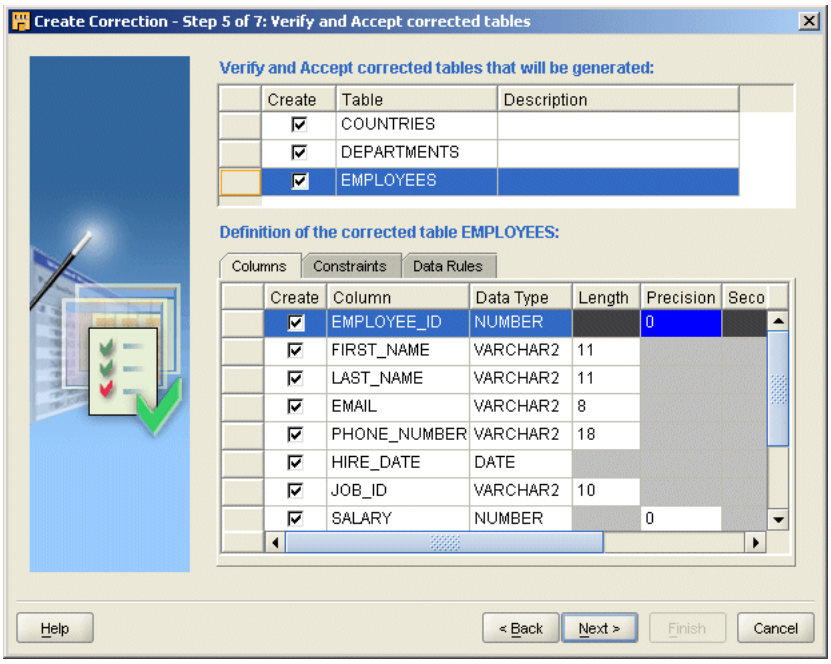
This page displays the validation warnings and errors that result for the data rules selected for correction. If there are any errors, you must first correct the errors and then proceed with the correction process.

Verify and Accept Corrected Tables

Use this page to verify and accept the corrected tables. This page lists the tables that have been modified with the applied rules and the data type changes. Select the tables you want to create in the corrected schema by selecting **Create** to the left of the object.

Figure 20–20 displays the Verify and Accept Corrected Tables page.

Figure 20–20 Verify and Accept Corrected Tables



The lower part of the page displays the details of the object selected on the top part of the page. It contains the following tabs: Columns, Constraints, and Data Rules.

The Columns tab provides details about the columns in the corrected table. You can select columns for creation, deletion, or modification. To specify that a column should be created, select **Create** to the left of the column name. You can modify the data type details of a column. However, you cannot modify a column name.

The Constrains tab displays the constraints for the corrected table. You can add constraints by clicking **Add Constraint**. Click **Delete** to delete the selected constraint.

The Data Rules tab displays the rule bindings for the corrected table. You can add, delete, or modify the rule bindings listed on this tab. These data rules are used to derive the data correction actions in the next step.

Choose Data Correction Actions

Use the Choose Data Correction Actions page to choose the action to perform to correct the source data. This page contains two sections: Select a Corrected Table and Choose Data Correction Actions. The Select a Corrected Table section lists the objects that you selected for corrections. Select a table in this section to display the affiliated data rules in the Choose Data Correction Actions section.

Choose Data Correction Actions

For each data rule, select an action from the drop down menu in the **Action** column. The settings you choose here determine how to handle data values that are not accepted due to data rule enforcement. Choose one of the following actions:

Ignore: The data rule is ignored and, therefore, no values are rejected based on this data rule.

Report: The data rule is run only after the data has been loaded for reporting purposes only. It is like the Ignore option, except a report is created that contains the values that did not adhere to the data rules. This action can be used for some rule types only.

Cleanse: The values rejected by this data rule are moved to an error table where cleansing strategies are applied. When you select this option, you must specify a cleansing strategy. See the following section for details about specifying cleansing strategies.

Cleansing Strategy

Use the **Cleansing Strategy** drop-down list to specify a cleansing strategy. This option is enabled only if you choose Cleanse in the Action column. The cleansing strategy depends on the type of data rule and the rule configuration. Error tables are used to store the records that do not conform to the data rule.

The options you can select for Cleansing Strategy are as follows:

- Remove

  Does not populate the target table with error records. This option is available for all data rules types.

- Custom

  Creates a function in the target table that contains a header, but no implementation details. You must add the implementation details to this function. A custom cleanse strategy is available for all data rule types except Unique Key, Referential, and Functional Dependency.

- Set to Min

  Sets the attribute value of the error record to the minimum value defined in the data rule. This option is available only for Domain Range rules that have a minimum defined.

- Set to Max

  Sets the attribute value of the error record to the maximum value defined in the data rule. This option is available for Domain Range rules that have a maximum defined.

- Similarity

  Uses a similarity algorithm based on permitted domain values to find a value that is similar to the error record. If no similar value is found, the original value is used. This option is available for Domain List rules with character data type only.

- Soundex

  Uses a soundex algorithm based on permitted domain values to find a value that is similar to the error record. If no soundex value is found, the original value is used. This option is available for Domain List rules with character data type only.

- Merge

  Uses the Match-Merge algorithm to merge duplicate records into a single row. You can use this option for Unique Key data rules only.

- Set to Mode

  Uses the mode value to correct the error records if a mode value exists for the functional dependency partition that fails. This option is used for Functional Dependency data rules only.

Summary

The Summary page provides a summary of the selected actions. When you are done assigning actions to the data rules, click **Finish**.

The correction schema is created and added to the Project Explorer. The correction objects and mappings are displayed under the module that you specify as the target module on the Select Target Module page of the Create Correction Wizard.

To create the correction tables in your target schemas, deploy the correction tables. To cleanse data, deploy and execute the correction mappings. The correction mappings have names that are prefixed with M_. For example, if your correction table is called `EMPLOYEES`, the correction mapping is called `M_EMPLOYEES`.

Viewing Correction Tables and Mappings

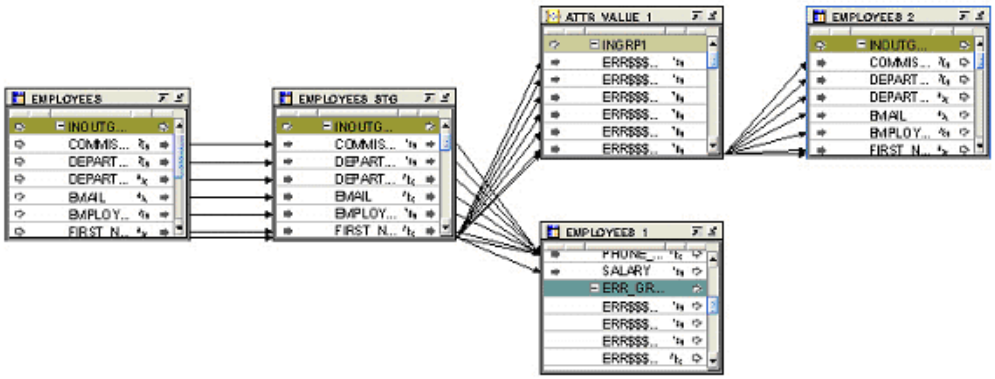
You can review the correction tables in the Data Object Editor to see the data rules and constraints created as part of the design of your table.

To view the correction mappings:

1. Double-click the table or mapping to open the object in their respective editors.
2. After the mapping is open, select **View** and then **Auto Layout** to view the entire mapping.

   Figure 20–21 displays a correction mapping generated by the Create Correction Wizard.

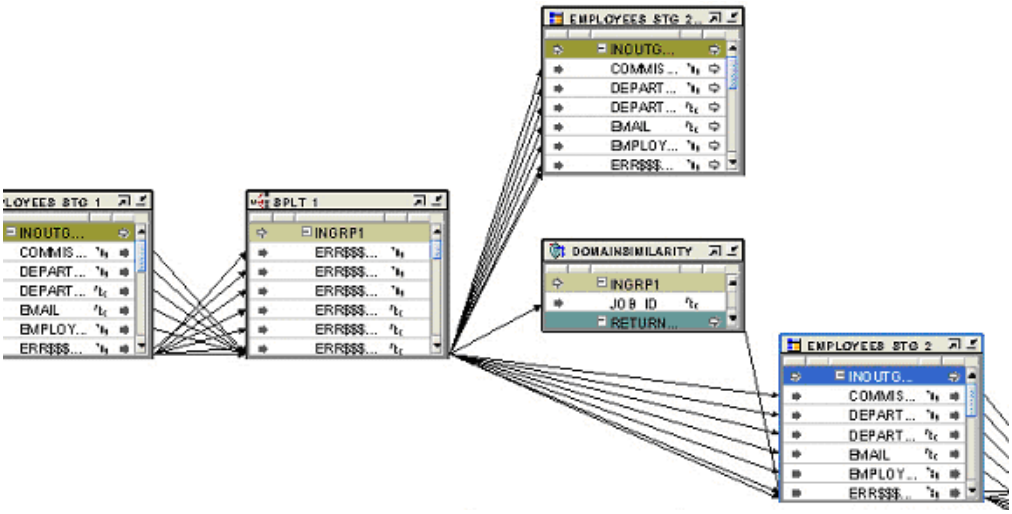
Figure 20–21 Generated Correction Mapping



3. Select the submapping ATTR_VALUE_1 and click the Visit Child Graph icon from the toolbar to view the submapping.

   The submapping is displayed as shown in Figure 20–22.

Figure 20–22 Correction SubMapping



The submapping is the element in the mapping that does the actual correction cleansing you specified in the Create Correction Wizard. In the middle of this submap is the DOMAINSIMILARITY transformation that was generated as a function by the Create Correction Wizard.

Using Data Rules

In addition to deriving data rules based on the results of data profiling, you can define your own data rules. You can bind a data rule to multiple tables within the project in which the data rule is defined. An object can contain any number of data rules.

You use the Design Center to create and edit data rules.

Once you create a data rule in Warehouse Builder, you can use it in any of the following scenarios.

Using Data Rules in Data Profiling

When you are using data profiling to analyze tables, you can use data rules to analyze how well data complies with a given rule and to collect statistics. From the results, you can derive a new data rule. If data profiling determines the majority of records have a value of red, white, and blue for a particular column, a new data rule can be derived that defines the color domain (red, white and blue). This rule can then be reused to profile other tables, or reused in cleansing, and auditing.

Using Data Rules in Data Cleansing and Schema Correction

Data rules can be used in two ways to cleanse data and correct schemas. The first way is to convert a source schema into a new target schema where the structure of the new tables strictly adheres to the data rules. The new tables would then have the right data types, constraints are enforced, and schemas are normalized. The second way data rules are used is in a correction mapping that will validate the data in a source table against the data rules, to determine which records comply and which do not. The analyzed data set is then corrected (for example, orphan records are removed, domain value inaccuracies are corrected, and so on) and the cleansed data set is loaded into the corrected target schema.

Using Data Rules in Data Auditing

Data rules are also used in data auditing. Data auditors are processes that validate data against a set of data rules to determine which records comply and which do not. Data auditors gather statistical metrics on how well the data in a system complies with a rule, and they report defective data into auditing and error tables. In that sense they are like data-rule-based correction mappings, which also offer a report-only option for data that does not comply with the data rules. For more information about data auditors, see "About Data Auditors" on page 10-17.

Types of Data Rules

Data rules in Warehouse Builder can be categorized as described in this section.

Domain List

A domain list rule defines a list of values that an attribute is allowed to have. For example, the Gender attribute can have 'M' or 'F'.

Domain Pattern List

A domain pattern list rule defines a list of patterns that an attribute is allowed to conform to. The patterns are defined in the Oracle Database regular expression syntax. An example pattern for a telephone number is as follows:

(^[[:space:]]*[0-9]{ 3 }[[:punct:]|:space:]]?[0-9]{ 4 }[[:space:]]*$)

Domain Range

A domain range rule defines a range of values that an attribute is allowed to have. For example, the value of the salary attribute can be between 100 and 10000.

Common Format

A common format rule defines a known common format that an attribute is allowed to conform to. This rule type has many subtypes: Telephone Number, IP Address, SSN, URL, E-mail Address.

No Nulls

A no nulls rule specifies that the attribute cannot have null values. For example, the department_id attribute for an employee in the Employees table cannot be null.

Functional Dependency

A functional dependency defines that the data in the data object may be normalized.

Unique Key

A unique key data rule defines whether an attribute or group of attributes are unique in the given data object. For example, the name of a department should be unique.

Referential

A referential data rule defines the type of a relationship (1:x) a value must have to another value. For example, the department_id attribute of the Departments table should have a 1:n relationship with the department_id attribute of the Employees table.

Name and Address

A name and address data rule uses the Warehouse Builder Name and Address support to evaluate a group of attributes as a name or address.

Custom

A custom data rule applies a SQL expression that you specify to its input parameters. For example, you can create a custom rule called VALID_DATE with two input parameters, START_DATE and END_DATE. A valid expression for this rule is: "THIS"."END_DATE" > "THIS"."START_DATE".

Creating Data Rule Folders

Each data rule belongs to a data rule folder, which is a container object used to group related data rules. To create a data rule, you must first create a data rule folder. The data rule folder acts is a container object that groups related data rules.

To create a data rule folder, in the navigation tree, right-click **Data Rules** and select **New**. The Create Data Rule Folder dialog is displayed.

Create Data Rule Folder

The **Name** field represents the name of the data rule folder. Enter a name for the data rule folder. To rename a data rule folder, select the name and enter the new name.

In the **Description** field, enter an optional description for the data rule folder.

You can choose to create a data rule immediately after you create a data rule folder. Select the **Proceed to Data Rule wizard** option to launch the Create Data Rule Wizard that helps you create a data rule.

Click **OK** to close the Create Data Rule Folder dialog.

Creating Data Rules

A data rule is an expression that determines the legal data within a data object or legal relationships between data objects. You can provide a definition for a data rule or derive a data rule based on the results of data profiling. For more information about data rules, see "About Data Rules" on page 10-16.

To create a data rule, right-click the data rule folder in which you want to create the data rule, and select **New**. The Welcome page of the Create Data Rule Wizard is displayed. Click **Next**. The wizard guides you through the following steps:

- Naming the Data Rule on page 20-37
- Defining the Rule on page 20-38
- Summary Page on page 20-38

Naming the Data Rule

The **Name** field represents the name of the data rule. Use this field to enter a name for the data rule. If you are deriving a data rule, a default name is assigned to the data rule. You can accept the default name or enter a different name.

On the Name tab of the Edit Data Rule dialog, use the **Name** field to rename the data rule. Select the name and enter the new name. Renaming a data rule has no effect on the data rule bindings. The binding names are not changed and the data rule remains in effect for the table.

Use the **Description** field to specify an optional description for the data rule.

Defining the Rule

Use the Define Rule page or the Define Rule tab to provide details about the data rule. The top portion of the page displays the **Type** drop-down list that represents the type of data rule. When you are deriving a data rule, Warehouse Builder automatically populates the Type field and you cannot edit this value. When you are creating a data rule, expand the Type field to view the types of data rules and select the type you want to create. When you edit a data rule, the Type field is disabled as you cannot change the type of data rule once it is created. For more information about types of data rules, see "Types of Data Rules" on page 20-36.

The bottom portion of this page specifies additional details about the data rule. The number and names of fields displayed here depend on the type of data rule you create. For example, if you select Custom, use the Attributes section to define the attributes required for the rule. Use the Ellipsis button on the Expression field to define a custom expression involving the attributes you defined in the Attributes section. If you select Domain Range as the type of data rule, the bottom portion of the page provides fields to specify the data type of the range, the minimum value, and the maximum value. When you are deriving a data rule, some of these fields are populated based on the profiling results from which you are deriving the rule. You can edit these values.

Summary Page

The Summary page displays the settings that you selected on the wizard pages. Review these settings. Click **Back** to modify any selections you made. Click **Finish** to create the data rule.

Data rules that are derived from the Data Profile Editor belong to the data rule folder called Derived_Data_Rules in the project that contains the data profile. Data rules that you create are part of the data rule folder under which you create the data rule.

Editing Data Rules

After you create a data rule, you can edit its definition. You can rename the data rule and edit its description. You cannot change the type of data rule. However, you can change the other parameters specified for the data rule. For example, for a Domain Range type of data rule, you can edit the data type of the range, the minimum range value, and the maximum range value.

To edit a data rule, in the Project Explorer, right-click the data rule and select **Open Editor**. You can also double-click the name of the data rule. The Edit Data Rule dialog is displayed. This dialog contains the following tabs:

- Name tab, see "Naming the Data Rule" on page 20-37.
- Define Rule tab, see "Defining the Rule" on page 20-38.

Applying Data Rules

Applying a data rule to an object binds the definition of the data rule to the object. For example, binding a rule to the table `Dept` ensures that the rule is implemented for the specified attribute in the table. You apply a data rule using the Data Object Editor. You can also apply a derived data rule from the Data Rule panel of the Data Profile Editor.

The Apply Data Rule Wizard enables you to apply a data rule to an object. Open the Data Object Editor for the object to which you want to apply the data rule. Navigate to the Data Rules tab. If any data rules are bound to the data object, these are displayed on this tab. To apply a new data rule to this object, click **Apply Rule**. The Welcome

page of the Apply Data Rule Wizard is displayed. Click **Next**. The wizard guides you through the following pages:

- Select Rule on page 20-39
- Name and Description on page 20-39
- Bind Rule Parameters on page 20-39
- Summary on page 20-39

Select Rule

Use the Select Rule page to select the data rule that you want to apply to the object. This page lists all available data rules. The data rules are grouped under the nodes BUILT_IN, DERIVED_DATA_RULES, and other data rule folders that you create.

The BUILT_IN node contains the default data rules that are defined in the repository. These include rules such as foreign key, unique key, not null. The DERIVED_DATA_RULES node lists all the data rules that were derived as a result of data profiling.

Name and Description

The Name and Description page enables you to specify a name and a optional description for the applied data rule.

The **Name** field contains the name of the data rule that you selected on the Define Rule page. You can accept this name or enter a new name.

Use the **Description** field to enter an optional description for the applied data rule.

Bind Rule Parameters

Use the Bind Rule Parameters page to bind the data rule to a column in your data object. The Binding column lists all columns in the data object to which the data rule is being applied. Use the drop-down list on this column to select the column to which the rule is bound.

Summary

The Summary page displays the options that you chose on the wizard pages. Click **Back** to go to the previous pages and change some options. Click **Finish** to apply the data rule. This adds information about the data rule bindings to the object metadata.

Tuning the Data Profiling Process

Data profiling is a highly processor and I/O intensive process and the execution time for profiling ranges from a few minutes to a few days. You can achieve the best possible data profiling performance by ensuring that the following conditions are satisfied:

- Your database is set up correctly for data profiling.
- The appropriate data profiling configuration parameters are used when you perform data profiling.

Tuning Warehouse Builder for Better Data Profiling Performance

You can configure a data profile to optimize data profiling results. Use the configuration parameters to configure a data profile. For more information about

configuration parameters, see "Configuration Parameters for Data Profiles" on page 20-11.

Use the following guidelines to make your data profiling process faster.

- Perform only the types of analysis that you require

  If you know that certain types of analysis are not required for the objects that you are profiling, use the configuration parameters to turn off these types of data profiling.

- Analyze lesser amount of data

  Use the `WHERE` clause and Sample Rate configuration parameters

If the source data for profiling stored in an Oracle Database, it is recommended that the source schema be located on the same database instance as the profile workspace. You can do this by installing the Warehouse Builder repository into the same Oracle instance as the source schema location. This avoids using a database link to move data from source to profiling workspace.

Tuning the Oracle Database for Better Data Profiling Performance

To ensure good data profiling performance, the machine that runs the Oracle Database must have certain hardware capabilities. In addition to this, you must optimize the Oracle Database instance on which you are performing data profiling.

For efficient data profiling, the following considerations are applicable:

- Multiple Processors
- Memory
- I/O System

Multiple Processors

The machine that runs the Oracle Database needs multiple processors. Data profiling has been designed and tuned to take maximum advantage of the parallelism provided by the Oracle Database. While profiling large tables (more than ten million rows), it is highly recommended to use a multiple processor machine.

Hints are used in queries required to perform data profiling. It picks up the degree of parallelism from the initialization parameter file of the Oracle Database. The default initialization parameter file contains parameters that take advantage of parallelism.

Memory

It is important that you ensure a high memory hit ratio during data profiling. You can do this by assigning a larger size of the System Global Area. It is recommended that the size of the System Global Area be configured to be no less that 500 MB. If possible, configure it to 2 GB or 3GB.

For advanced database users, it is recommended that you observe the buffer cache hit ratio and library cache hit ratio. Set the buffer cache hit ratio to higher than 95% and the library cache hit ratio to higher than 99%.

I/O System

The capabilities of the I/O system have a direct impact on the data profiling performance. Data profiling processing frequently runs performs full table scans and massive joins. Since today's CPUs can easily out-drive the I/O system, you must

carefully design and configure the I/O subsystem. Keep in mind the following considerations that aid better I/O performance.

- You need a large number of disk spindles to support uninterrupted CPU and I/O cooperation. If you have only a few disks, the I/O system is not geared towards a high degree of parallel processing. It is recommended to have a minimum of two disks for each CPU.
- Configure the disks. It is recommended that you create logical stripe volumes on the existing disks, each striping across all available disks. Use the following formula to calculate the stripe width.

  ```
  MAX(1,DB_FILE_MULTIBLOCK_READ_COUNT/number_of_disks)*DB_
  BLOCK_SIZE
  ```

  Here, DB_FILE_MULTIBLOCK_SIZE and DB_BLOCK_SIZE are parameters that you set in your database initialization parameter file. You can also use a stripe width that is a multiple of the value returned by the formula.

  To create an maintain logical volumes, you need a volume management software such as Veritas Volume Manager or Sun Storage Manager. If you are using Oracle Database 10*g* and you do not have any volume management software, you can use the Automatic storage Management feature of the Oracle Database to spread workload to disks.
- Create different stripe volumes for different tablespaces. It is possible that some of the tablespaces occupy the same set of disks.

  For data profiling, the USERS and the TEMP tablespaces are normally used at the same time. So you can consider placing these tablespaces on separate disks to reduce interference.

Using Data Auditors

Data auditors are Warehouse Builder objects that you can use to continuously monitor your source schema to ensure that data adheres to the defined data rules. You can monitor an object only of you have defined data rules for the object. You can create data auditors for tables, views, materialized views, and external tables.

Data auditors can be deployed and executed as standalone processes, but they are typically run to monitor data quality in an operational environment like a data warehouse or ERP system and, therefore, can be added to a process flow and scheduled.

Note: You cannot import metadata for data auditors in Merge mode. For more information about import mode options, see "Import Option" on page 33-10.

Creating Data Auditors

Data auditors enable you to monitor the quality of data in an operational environment. For more information about data auditors, see "About Data Auditors" on page 10-17.

Use the Create Data Auditor Wizard to create data auditors. Data auditors are part of an Oracle module in a project.

Use the following steps to create a data auditor:

1. Expand the Oracle module in which you want to create the data auditor.

2. Right-click **Data Auditors** and select **New**.

   The Create Data Auditor Wizard is displayed and it guides you through the following steps:

   - Naming the Data Auditor on page 20-42
   - Selecting Objects on page 20-42
   - Choosing Actions on page 20-42
   - Summary Page on page 20-43

Naming the Data Auditor

The **Name** field represents the name of the data auditor. The name of the data auditor should be unique within the Oracle module to which it belongs. Use the Name tab of the Edit Data Auditor dialog to rename a data auditor. You select the name and then enter the new name.

The **Description** field represents the optional description that you can provide for the data auditor.

Selecting Objects

Use the Select Objects page or the Select Objects tab to select the data objects that you want to audit. The Available section displays the list of available objects for audit. This list contains only objects that have data rules bound to them. The Selected section displays the objects that are selected for auditing. Use the shuttle buttons to move objects from the Available section to the Selected section.

On the Select Objects tab, the Selected section lists the objects currently selected for auditing. You can add more objects or remove existing objects from the lists using the shuttle arrows.

Choosing Actions

Use the Choose Action page or the Choose Action tab to select the action to be taken for records that do not comply with the data rules that are bound to the selected objects. Provide information in the following sections of this page: Error Threshold Mode and Data Rules.

Error Threshold Mode

Error threshold mode is the mode used to determine the compliance of data to data rules in the objects. You can select the following options:

- **Percent:** The data auditor will set the audit result based on the percentage of records that do not comply with the data rule. This percentage is specified in the rule's defect threshold.
- **Six Sigma:** The data auditor will set the audit result based on the Six Sigma values for the data rules. If calculated Six Sigma value for any rule is less than the specified Six Sigma Threshold value, then the data auditor will set the AUDIT RESULT to 2.

Data Rules

This section contains the following details:

- **Data Rule:** The name of the table suffixed with the name of the bound data rule.
- **Rule Type:** The type of data rule.

- **Action:** The action to be performed if data in the source object does not comply with the data rule. Select **Report** to ensure that the data rule is audited. Select **Ignore** to cause the data rule to be ignored.
- **Defect Threshold:** The percent of records that should comply to the data rules to ensure successful auditing. Specify a value between 1 and 100. This value is ignored if you set the Error Threshold Mode to Six Sigma.
- **Sigma Threshold:** The required success rate. Specify a number between 0 and 7. If you set the value to 7, no failures are allowed. This value is ignored if you set the Error Threshold Mode to Percent.

Summary Page

The Summary page lists the options that you selected in the previous pages of the wizard. To modify any of the selected options, click **Back**. To proceed with the creation of the data auditor, click **Finish**. A data auditor is created using the settings displayed on the Summary page.

Editing Data Auditors

After you create a data auditor, you can edit it and modify any of its properties. To edit a data auditor, in the Design Center, right-click the data auditor and select **Open Editor**. The Edit Data Auditor dialog is displayed. Use the following tabs to modify the properties of the data auditor:

- Name tab, see Naming the Data Auditor on page 20-42
- Select Objects tab, see Selecting Objects on page 20-42
- Choose Actions tab, see Choosing Actions on page 20-42
- Reconcile Objects tab, see Reconciling Objects on page 20-43

Reconciling Objects

If the definitions of the objects included in the data auditor were modified after the data auditor was created, the objects in the data auditor will not be identical with the actual objects. The table on this tab lists the audit objects and the source objects that were included for audit. To reconcile the definition of an audit object with the source object, select the check box on the row that corresponds to the changed object and click **Reconcile**. This reconciles the audit object definition with the source object.

Auditing Objects Using Data Auditors

After you create a data auditor, you can use it to monitor the data in your data objects. This ensures that the data rule violations for the objects are detected. When you run a data auditor, any records that violate the data rules defined on the data objects are written to the error tables.

There are two ways of using data auditors:

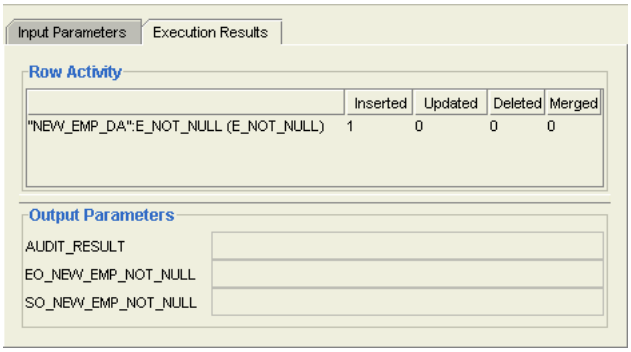
- Manually Running Data Auditors
- Automatically Running Data Auditors

Data Auditor Execution Results

After you run a data auditor, the Job Details dialog displays the details of the execution. The Job Details dialog contains two tabs: Input Parameters and Execution Results. Figure 20–23 displays the Execution Results tab of the Job Details dialog. Note

that the Job Details dialog is displayed only when you set the deployment preference Show Monitor to true. For more information about deployment preferences, see "Deployment Preferences" on page 3-6.

Figure 20–23 Data Auditor Execution Results



The Input Parameters tab contains the values of input parameters used to run the data auditor. The Execution Results tab displays the results of running the data auditor. This tab contains two sections: Row Activity and Output Parameters.

The Row Activity section contains details about the inserts into the error table for each step. In Figure 20–23, the data rule called E_NOT_NULL inserted one record into the error table. Note that when more than one data rule is specified, multi-table insert may be used in the data auditor. In this case, the count of the number of rows will not be accurate.

The Output Parameters section contains the following three parameters:

- **AUDIT_RESULT:** Indicates the result of running the data auditor. The possible values for this parameter are as follows:

  0: No data rule violations occurred.

  1: At least one data rule violation occurred, but no data rule failed to meet the minimum quality threshold as defined in the data auditor. For more information about setting the threshold, see the section on Data Rules in "Choosing Actions" on page 20-42.

  2: At least one data rule failed to meet the minimum quality threshold.

- **EO_<data_rule_name>:** Represents the calculated error quality for the specified data rule. 0 indicates all errors and 100 indicates no errors.
- **SO_<data_rule_name>:** Represents the Six Sigma quality calculated for the specified data rule.

Manually Running Data Auditors

You run a data auditor to check if the data in the data object adheres to the data rules defined for the object. You can run data auditors from the Design Center or the Control Center Manager. To run a data auditor from the Design Center, right-click the data auditor and select **Start**. In the Control Center Manager, select the data auditor, and from the File menu, select **Start**. The results are displayed in the Job Details panel described in "Data Auditor Execution Results" on page 20-43.

Automatically Running Data Auditors

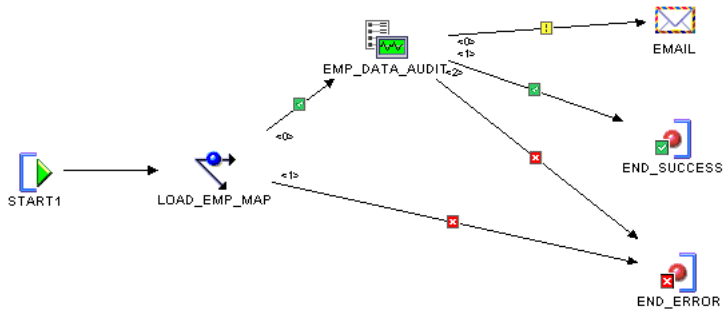
You can automate the process of running a data auditor using the following steps:

1. Create a process flow that contains a Data Auditor Monitor activity.
2. Schedule this process flow to run at a predefined time.

   For more information about scheduling objects, see "Process for Defining and Using Schedules" on page 11-4.

Figure 20–24 displays a process flow that contains a Data Auditor Monitor activity. In this process flow, LOAD_EMP_MAP is a mapping that loads data into the EMP table. If the data load is successful, the data auditor EMP_DATA_AUDIT is run. The data auditor monitors the data in the EMP table based on the data rules defined for the table.

Figure 20–24 Data auditor Monitor Activity in a Process Flow



Configuring Data Auditors

During the configuration phase, you assign physical deployment properties to the data auditor you created by setting the configuration parameters. The Configuration Properties dialog enables you to configure the physical properties of the data auditor.

To configure a data auditor:

1. From the Project Explorer, expand the Databases node and then the Oracle node.
2. Right-click the name of the data auditor you want to configure and select **Configure**.

   The Configuration Properties dialog is displayed.
3. Configure the parameters listed in the following sections.

Run Time Parameters

Default Purge Group: This parameter is used when executing the package. Each audit record in the runtime schema is assigned to the purge group specified.

Bulk size: The number of rows to be fetched as a batch while processing cursors.

Analyze table sample percentage: The percentage of rows to be samples when the target tables are analyzed. You analyze target tables to gather statistics that you can use to improve performance while loading data into the target tables.

Commit frequency: The number of rows processed before a commit is issued.

Maximum number of errors: The maximum number of errors allowed before Warehouse Builder terminates the execution of this step.

Default Operating Mode: The operating mode used. The options you can select are Row based, Row based (target only), Set based, Set based fail over to row based, Set based fail over to row based (target only).

Default Audit Level: Use this parameter to indicate the audit level used when executing the package. When the package is run, the amount of audit information captured in the runtime schema depends on the value set for this parameter.

The options you can select are as follows:

ERROR DETAILS: At runtime, error information and statistical auditing information is recorded.

COMPLETE: All auditing information is recorded at runtime. This generates a huge amount of diagnostic data which may quickly fill the allocated tablespace.

NONE: No auditing information is recorded at runtime.

STATISTICS: At runtime, statistical auditing information is recorded.

Data Auditor

This category uses the same name as the data auditor and contains the following generic data auditor configuration parameters.

Generation comments: Specify additional comments for the generated code.

Threshold Mode: Specify the mode that should be used to measure failure thresholds. the options are PERCENTAGE and SIX SIGMA.

Language: The language used to define the generated code. The options are PL/SQL (default) and UNDEFINED. Ensure that PL/SQL (default) is selected.

Deployable: Select this option to indicate that you want to deploy this data auditor. Warehouse Builder generates code only if the data auditor is marked as deployable.

Referred Calendar: Specify the schedule to associate with the data auditor. The schedule defines when the data auditor will run.

Code Generation Options

ANSI SQL syntax: Select this option to use ANSI SQL code in the generated code. If this option is not selected, Warehouse Builder generates Oracle SQL syntax.

Commit control: Specifies how commit is performed. The options available for this parameter are: Automatic, Automatic Correlated, and Manual. Ensure that this parameter is set to Automatic.

Enable Parallel DML: Select this option to enable parallel DML at runtime.

Analyze table statistics: Select this option to generate the statement used to collect statistics for the data auditor.

Optimized Code: Select this option to indicate to Warehouse Builder that it should generate optimized code.

Generation Mode: Select the mode in whichWarehouse Builder should generate optimized code. the option you can select are as follows: All Operating Modes, Row based, Row based (target only), Set based, Set based fail over to row based, and Set based fail over to row based (target only)

Use Target Load Ordering: Select this option to generate code for target load ordering.

Error Trigger: Specify the name of the error trigger procedure.

Bulk Processing code: Select this option to generate bulk processing code.

21

Data Quality Operators

Because reporting on erroneous data wastes time and money, data quality is a key element of Business Intelligence. Include Warehouse Builder data quality operators in your mappings to load clean, accurate records to your targets.

This chapter discusses the Warehouse Builder mapping operators that help you achieve data quality. This chapter contains the following topics:

Using the Match-Merge Operator

This section includes information and examples on how to use the Match-Merge operator in a mapping. The Match-Merge operator can be used with the Name and Address operator to support **householding**, which is the process of identifying unique households in name and address data.

Example of Matching and Merging Customer Data

Consider how you could utilize the Match-Merge operator to manage a customer mailing list. Use matching to find records that refer to the same person in a table of customer data containing 10,000 rows. For example, you can define a match rule that screens records that have similar first and last names. Through matching you may discover that 5 rows refer to the same person. You can merge those records into one new record. For example, you can create a merge rule to retain the values from the one of the five matched records with the longest address. The newly merged table now contains one record for each customer.

Table 21–1 shows records that refer to the same person prior to using the Match-Merge operator.

Table 21–1 Sample Records

Row	First Name	Last Name	SSN	Address	Unit	Zip
1	Jane	Doe	NULL	123 Main Street	NULL	22222
2	Jane	Doe	111111111	NULL	NULL	22222
3	J.	Doe	NULL	123 Main Street	Apt 4	22222
4	NULL	Smith	111111111	123 Main Street	Apt 4	22222
5	Jane	Smith-Doe	111111111	NULL	NULL	22222

Table 21–2 shows the single record for Jane Doe after using the Match-Merge operator. Notice that the new record retrieves data from different rows in the sample.

Table 21–2 Jane Doe Record After Using Match-Merge Operator

First Name	Last Name	SSN	Address	Unit	Zip
Jane	Doe	111111111	123 Main Street	Apt 4	22222

Understanding Matching Concepts

When you use Warehouse Builder to match records, you can define a single match rule or multiple match rules. If you create more than one match rule, Warehouse Builder determines two rows match if those rows satisfy any of the match rules. In other words, Warehouse Builder evaluates multiple match rules using OR logic.

Example of Multiple Match Rules

The following example illustrates how Warehouse Builder evaluates multiple match rules using OR logic.

In the top portion of the Match Rules tab, create two match rules as described in Table 21–3:

Table 21–3

Name	Position	Rule Type	Usage	Description
Rule_1	1	Conditional	Active	Match SSN
Rule _2	2	Conditional	Active	Match Last Name and PHN

In the lower portion of the tab, assign the details to Rule_1 as described in Table 21–4:

Table 21–4

Attribute	Position	Algorithm	Similarity Score	Blank Matching
SSN	1	Exact	0	Do not match if either is blank

For Rule_2, assign the details as described in Table 21–5:

Table 21–5

Attribute	Position	Algorithm	Similarity Score	Blank Matching
LastName	1	Exact	0	Do not match if either is blank
PHN	2	Exact	0	Do not match if either is blank

Assume you have the data listed in Table 21–6:

Table 21–6

Row	First Name	Last Name	PHN	SSN
A	John	Doe	650-123-1111	NULL
B	Jonathan	Doe	650-123-1111	555-55-5555

Table 21–6 (Cont.)

Row	First Name	Last Name	PHN	SSN
C	John	Dough	650-123-1111	555-55-5555

According to Rule_1, rows B and C match. According to Rule_2, rows A and B match. Therefore, since Warehouse Builder handles match rules using OR logic, all three records match.

Example of Transitive Matching

The general rule is, if A matches B, and B matches C, then A matches C. Assign a conditional match rule based on similarity such as described in Table 21–7:

Table 21–7 Conditional Match Rule

Attribute	Position	Algorithm	Similarity Score	Blank Matching
LastName	1	Similarity	80	Do not match if either is blank

Assume you have the data listed in Table 21–8:

Table 21–8 Sample Data

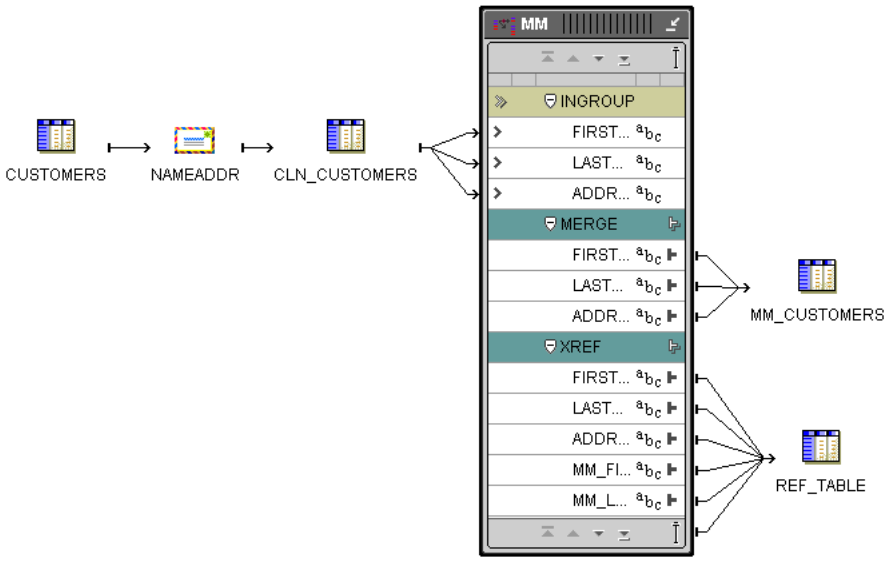
Row	First Name	Last Name	PHN	SSN
A	John	Jones	650-123-1111	NULL
B	Jonathan	James	650-123-1111	555-55-5555
C	John	Jamos	650-123-1111	555-55-5555

Jones matches James with a similarity of 80, and James matches Jamos with a similarity of 80. Jones does not match Jamos because the similarity is 60, which is less than the threshold of 80. However, because Jones matches James, and James matches Jamos, all three records match (Jones, James, and Jamos).

Designing Mappings with a Match-Merge Operator

Figure 21–1 shows a mapping you can design using a Match-Merge operator. Notice that the Match-Merge operator is preceded by a Name and Address operator, NAMEADDR, and a staging table, CLN_CUSTOMERS. You can design your mapping with or without a Name and Address operator. Preceding the Match-Merge operator with a Name and Address operator provides clean and standardized data before launching time consuming match and merge operations.

Figure 21–1 Match-Merge Operator in a Mapping



Whether you include a Name and Address operator or not, be aware of the following considerations as you design your mapping:

- **Operating modes:** Warehouse Builder operators may accept either set-based or row-based input and generate either set-based or row-based output. SQL is set-based, so a set of records is processed at one time. PL/SQL is row-based, so each row in processed separately. When the Match-Merge operator matches records, it compares each row with the subsequent row in the source and generates row-based code only. A mapping that contains a Match-Merge operator can only run in row-based mode. For more information about set-based and row-based operators, refer to Chapter 8, "Understanding Performance and Advanced ETL Concepts".
- **SQL based operators before Match-Merge:** The Match-Merge operator accepts set-based SQL input, but generates only row-based PL/SQL output. Any operators that generate only SQL code must precede the Match-Merge operator. For example, the Joiner, Key Lookup, and Set operators generate set-based SQL output, so they must precede Match-Merge. If set-based operators appear after Match-Merge, then the mapping is invalid.
- **PL/SQL input:** The Match-Merge operator requires SQL input except from another Match-Merge operator, as described in "Using Two Match-Merge Operators" on page 21-5. If you want to precede a Match-Merge with an operator that generates only PL/SQL output such as the Name and Address operator, you must first load the data into a staging table.
- **Refining Data from Match-Merge operators:** To achieve greater data refinement, map the XREF output from one Match-Merge operator into another Match-Merge operator. This scenario is the one exception to the SQL input rule for Match-Merge operators. With additional design elements, the second Match-Merge operator accepts PL/SQL. For more information, see "Using Two Match-Merge Operators".

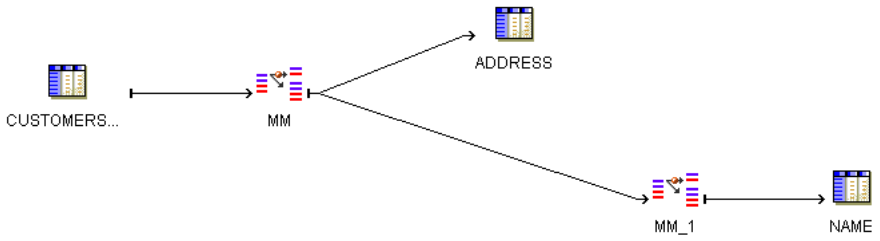
Using Two Match-Merge Operators

Most match-merge operations can be performed by a single match-merge operator. However, if you are directing the output to two different targets, then you may need to use two match-merge operators in succession.

For example, when householding name and address data, you may need to merge the data first for addresses and then again for names. Assuming you map the MERGE output to a target table, you can map the `XREF` group to another Match-Merge operator. Although you could map the XREF group to a staging table, this practice can lead to significant loss of performance.

Figure 21–2 shows a mapping that uses two match-merge operators. The `XREF` group from `MM` is mapped directly to `MM_1`. For this mapping to be valid, you must assign the Match ID generated for the first `XREF` group as the Match Bin rule on the second Match-Merge operator.

Figure 21–2 Householding Data: XREF Group Mapped to Second Match-Merge Operator



Match-Merge Wizard and Editor: Name

Use the Name page to specify a name and optional description for the operator. By default, the wizard names the Match-Merge operator MATCHMERGE.

Match-Merge Wizard and Editor: Groups

Use the Groups page to enter customized names and descriptions of the input and output groups.

Group

Lists the predefined input and output groups. You can rename the groups, but you cannot add or delete groups in the Match-Merge operator.

Direction

Identifies whether the group is for input or output. The Match-Merge operator accepts one SQL input group and generates two PL/SQL output groups. The MERGE group has the merged data. The XREF group is an optional group for documenting the merge process.

Description

Enter an optional description.

Match-Merge Wizard and Editor: Input Connections

Use the Input Connections page to select attributes for the input group (INGRP1).

To complete the Input connections page for an operator:

1. Select complete groups or individual attributes from the left panel.

   To search for a specific attribute or group by name, type the text in **Search for** and click **Go.** To find the next match, click **Go** again.

   Hold the Shift key down to select multiple groups or attributes. If you want to select attributes from different groups, you must first combine the groups with a Joiner or Set operator.

2. Use the left to right arrow button in the middle of the page to move your selections to the right side of the wizard page.

   You can use the right to left arrow to move groups or attributes from the input connections list. Warehouse Builder removes the selection from the input group and removes the mapping line between the source operator and the current operator.

Match-Merge Wizard and Editor: Input Attributes

Use the **Input Attributes** page to assign input roles to each input attribute.

Attribute

Automatically lists the attributes that you selected on the Input Connections page. Click **Add** to add a new input attribute.

Input Role

Input roles indicate what kind of information resides in a line of data. For each attribute, select the input role that most closely matches the data contained in the source attribute.

Data Type, Length, Precision, Scale, Seconds Precision

Attributes are set automatically to `NUMBER`.

Description

Enter an optional description of the input attributes.

Add

Adds a row so that you can define a new attribute.

Delete

Deletes the selected attribute.

Match-Merge Wizard and Editor: Merge Output

Use this page to specify the attributes for the output MERGE group. The MERGE group produces a consolidated record from the attributes you selected.

Source Attributes

Lists all attributes defined for this Match-Merge operator. Use the shuttle buttons to move selected attributes to the Output Attributes field.

Output Attributes

Lists attributes selected for the output MERGE group.

Match-Merge Wizard and Editor: Cross-Reference Output

Use the Cross-Reference Output page to optionally select attributes for the XREF group. Although the Match-Merge operator creates the XREF group by default, you have the option of adding attributes to the group or leaving it empty.

You can use the XREF group to document the merge process. Create a foreign key relationship between the original data set and the new merged data set. Then send the attributes from the XREF group to a table that records the corresponding source row for each merged row.

Alternatively, you can use the XREF group as input to a second Match-Merge operator. By using two operators, you can direct the merged output from a set of attributes to two different target. For an example, refer to "Using Two Match-Merge Operators" on page 21-5.

Source Attributes

Lists the input attributes (INGRP1) and an XREF output attribute for each MATCH output attribute. The XREF attributes are distinguished by a prefix, which has a default value of MM_.

Output Attributes

The attributes that you want to cross-reference. To move selected attributes between the Source Attributes and Output Attributes lists, use the shuttle keys.

Merge Prefix

The prefix used to distinguish cross-reference output from data output.

Set Prefix

Changes the prefix on XREF attributes in both lists to the value displayed in the Merge Prefix field.

Match-Merge Wizard and Editor: Match Bins

Use the Match Bins page to limit the number of rows to be compared. When Warehouse Builder matches the rows, it compares each row with the subsequent row for all rows within the same grouping. Limiting the number of rows can greatly enhance performance, because Warehouse Builder searches for matches only within a bin and not throughout the entire data set.

Ideally, keep the number of rows in each grouping under 2000. The number of comparisons Warehouse Builder performs is based on the following formula:

```
n=(b*(b-1))/2
```

where n is the number of comparisons, and b is the number of records in a bin.

For example, matching 5 records requires 10 comparisons, matching 50 records requires 1,225 comparisons, and matching 500 records requires 124,750 comparisons.

While you want to define Match Bins that separate rows into manageable groupings, you also want to avoid separating rows that should be matched. The attributes you select for grouping similar rows depends on your data. For example, if you have a

table of customer addresses with a million rows, you may want to group the data by partial street name, city name, and zip code.

Available Attributes

Lists all input attributes, from which you can select the ones to use for binning.

Selected Attributes

One or more attributes that must match for a row to be included in a particular bin. To move attributes between the Source Attributes and Output Attributes lists, select one or more attributes and click the arrow keys located between the two lists. Use the arrow keys at the right to order the attributes from the more general at the top (such as Country) to the more specific at the bottom (such as Street).

Match New Records Only

After the first deployment, you can choose whether to match and merge all records or only new records. You do not want to match and merge the same data twice because of the impact on performance. Instead, you can just match and merge the new, uncleansed data. This option enables you to add uncleansed data into the data warehouse.

New Record Condition

Displays the conditional expression used to identify new records. Click the Ellipsis button to display the Match New Record Condition Editor (also known as the Expression Builder User Interface).

Match-Merge Wizard and Editor: Match Rules

The Match Rules are used to identify duplicate records, even though some fields may have different values. You can define match rules for a single attribute or multiple attributes in the operator. On the Match Rules tab, create match rules at the top of the page. In the lower portion of Match Rules tab, specify the details for each match rule. Table 21–9 describes the match rules.

If you create more than one match rule, Warehouse Builder determines a match when those two rows satisfy any of the match rules. In other words, Warehouse Builder evaluates multiple match rules using OR logic. This is indicated by the OR icon to the left of each additional row. For more information, see "Understanding Matching Concepts" on page 21-2.

Name

An arbitrary name for the rule. Warehouse Builder creates a default name such as MA_0 and MA_1 for each match rule. You can replace these names with meaningful ones. Meaningful names are particularly helpful when referencing the rules from a custom PL/SQL program.

Position

The order of execution. You can change the position of a rule by clicking on the row header and dragging the row to its new location. The row headers are the boxes to the left of the Name column.

Rule Type

Assign one of the rule types listed in Table 21–9. When you select a rule type, the lower portion of the Match Rules tab activates and you can enter details for the match rule.

Usage

You can designate a match rule as either active or passive.

- **Active:** Warehouse Builder executes a match rule if you designate it as active. If more than one match rule is active, each is evaluated in turn until a match is found or until all rules are evaluated. If any rule identifies a match, those records are considered a match.
- **Passive:** Warehouse Builder does not directly execute passive match rules. It only executes passive rules when they are called through an active custom match rule. All defined match rules appear in a list of available functions in the Custom Match Rule Editor.

Description

Displays an optional description, which you can enter.

Descriptions of Match Rules

Table 21–9 describes the types of match rules.

Table 21–9 Match Rule Descriptions

Match Rule	Description
All Match	Matches all the rows within the match bin.
None Match	Turns off matching. No rows match within the match bin.
Conditional	Matches rows based on an algorithm you select. For details, see "Conditional Match Rule" on page 21-9.
Weight	Matches rows based on scores that you assign to the attributes. For details, see "Weight Match Rule" on page 21-12.
Person	Matches records based on people's names. For details, see "Person Match Rule" on page 21-13.
Firm	Matches records based on business names. For details, see "Firm Match Rule" on page 21-15.
Address	Matches records based on postal addresses. For details, see "Address Match Rule" on page 21-16.
Custom	Create a custom comparison algorithm. Select **Edit** to launch the Custom Match Rule Editor. For more information, see "Custom Match Rule" on page 21-18.

Conditional Match Rule

Use the Conditional Match Rule to combine multiple attribute comparisons into one composite rule. When you assign multiple attributes for comparison, all the comparisons must be true for the records to be considered a match. Warehouse Builder displays the AND icon in the left-most column of subsequent conditions.

Attribute

Identifies the attribute that will be tested for a particular condition. You can select from any input attribute (INGRP1).

Position

The order of execution. You can change the position of a rule by clicking on the row header and dragging the row to its new location. The row headers are the boxes to the left of the Attribute column.

Algorithm

A list of methods that can be used to determine a match. Table 21–10 describes the algorithms.

Similarity Score

The minimum similarity value required for two strings to match, as calculated by the Edit Distance, Standardized Edit Distance, Jaro-Winkler, or Standardized Jaro-Winkler algorithms. Enter a value between 0 and 100. A value of 100 indicates an exact match, and a value of 0 indicates no similarity.

Blank Matching

Lists options for handling empty strings in a match.

Defining a Conditional Match Rule

To define a Conditional match rule, complete the following steps:

1. On the top portion of the Match Rules tab, select **Conditional** for the rule type.

   The operator displays a Details section at the bottom of the tab.

2. Click **Add** to add a new row.
3. Select an attribute.
4. Select an algorithm. Refer to Table 21–10 for descriptions.
5. Specify a similarity score for the Edit Distance, Standardized Edit Distance, Jaro-Winkler, or Standardized Jaro-Winkler algorithms.
6. Select a method for handling blanks.

Algorithms for Conditional Match Rules

Table 21–10 describes the algorithms available for defining conditional match rules.

Table 21–10 Algorithms for Conditional Match Rules

Algorithm	Description
Exact	Matches values only when they are exactly the same. For example, 'Dog' and 'dog!' do not match because the second string is not capitalized and contains an extra character. For numeric, date, and other non-character data types, this is the only type of comparison allowed.
Standardized Exact	Eliminates case, spaces, and non-alphanumeric characters before comparing for an exact match. For example, 'Dog' and 'dog!' do match.
Soundex	Matches phonetically similar stings. The operator converts the strings to phonetic codes. If the codes match, then the two strings match. This algorithm is often used for matching names. It is an older phonetic algorithm than Double Metaphone. The Soundex algorithm ignores case and spaces. It basically retains the first letter, eliminates vowels, and replaces consonants with numbers. Consonants with similar sounds have the same numeric value.

Table 21–10 (Cont.) Algorithms for Conditional Match Rules

Algorithm	Description
Edit Distance	Matches strings with a similarity value equal to or greater than the Similarity Score that you specify. This algorithm is often used for correcting typographical errors such as transposed characters. The Edit Distance algorithm calculates the number of deletions, insertions, or substitutions required to transform one string into another. A similarity value of 100 indicates that the two strings are identical. A value of zero indicates no similarity whatsoever. For example, if the string 'tootle' is compared with the string 'tootles', then the edit distance is 1. The length of the string 'tootles' is 7. The similarity value is therefore 6/7*100 or 85.
Standardized Edit Distance	Eliminates case, spaces, and non-alphanumeric characters before using the Edit Distance algorithm to determine a match.
Partial Name	Matches strings when one string is contained in the other, starting with the first word. The algorithm performs a Standardized Exact comparison on the entire string before attempting to match a partial name. For example, 'Midtown Power' matches 'Midtown Power and Light,' but would not match 'Northern Midtown Power'.
Abbreviation	Matches strings when one string contains words that are abbreviations of corresponding words in the other. The operator first performs a Standardized Exact comparison on the entire string. It then looks for abbreviations for each word. If the larger of the words contains all of the letters from the shorter word and the letters appear in the same order as the shorter word, then the words are considered a match. For example, 'Intl. Business Products' matches 'International Bus Prd'.
Acronym	Matches strings if one string is an acronym for the other. The operator first performs a Standardized Edit Distance comparison on the entire string. If no match is found, then each word of one string is compared to the corresponding word in the other string. If the entire word does not match, each character of the word in one string is compared to the first character of each remaining word in the other string. If the characters are the same, the names match. For example, 'Chase Manhattan Bank NA' matches 'CMB North America.' The comparison ignores case, non-alphanumeric characters, and noise words such as 'and' and 'the.'
Jaro-Winkler	Matches strings based on their similarity value using an improved comparison system over the Edit Distance algorithm. It accounts for the length of the strings and penalizes more for errors at the beginning. It also recognizes common typographical errors. The strings match when their similarity value is equal to or greater than the Similarity Score that you specify. A similarity value of 100 indicates that the two strings are identical. A value of zero indicates no similarity whatsoever. Note that the value actually calculated by the algorithm (0.0 to 1.0) is multiplied by 100 to correspond to the Edit Distance scores.
Standardized Jaro-Winkler	Eliminates case, spaces, and non-alphanumeric characters before using the Jaro-Winkler algorithm to determine a match.

Table 21–10 (Cont.) Algorithms for Conditional Match Rules

Algorithm	Description
Double Metaphone	Matches phonetically similar strings using an improved coding system over the Soundex algorithm. It generates two codes for strings that could be pronounced in multiple ways. If the primary codes match for the two strings, or if the secondary codes match, then the strings match. The Double Metaphone algorithm accounts for alternate pronunciations in Italian, Spanish, French, and Germanic and Slavic languages. Unlike the Soundex algorithm, Double Metaphone encodes the first letter, so that 'Kathy' and 'Cathy' evaluate to the same phonetic code.

Weight Match Rule

Use this rule to match rows based on a weight value. A weighted match rule is most useful when comparing a large number of attributes. This rule can prevent a single attribute from invalidating a match, which results from the AND logic of conditional rules.

Warehouse Builder compares each attribute using a similarity algorithm that returns a score between 0 and 100 to represent the similarity between the rows. For two rows to be considered a match, the total counts must be greater than the overall score you designate.

Similarity Algorithm

The method used to determine a match. Choose from these algorithms:

- **Edit Distance:** Calculates the number of deletions, insertions, or substitutions required to transform one string into another.
- **Jaro-Winkler:** Uses an improved comparison system over the Edit Distance algorithm. It accounts for the length of the strings and penalizes more for errors at the beginning. It also recognizes common typographical errors.

Attribute

Identifies the attribute that will be tested for a particular condition. You can select from any input attribute (INGRP1).

Maximum Score

The weight value for the attribute. This value should be greater than the value of Required Score to Match.

Score When Blank

The similarity value when one of the records is empty.

Required Score to Match

A value that represents the similarity required for a match. A value of 100 indicates that the two values are identical. A value of zero indicates there is no similarity.

Using the Weight Match Rule

To use the Weight match rule, complete the following steps:

1. On the Match Rules tab, select Weight as the Rule Type.

   The Details tab is displayed at the bottom of the page.

2. Select **Add** at the bottom of the page to add a new row.
3. For each row, select an attribute to add to the rule.
4. In **Maximum Score,** assign a weight to each attribute. Warehouse Builder compares each attribute using a similarity algorithm that returns a score between 0 and 100 to represent the similarity between the rows.
5. In **Required score to match,** assign an overall score for the match.

   For two rows to be considered a match, the total counts must be greater than the Required score.

Example of a Weight Match Rule

Assume you want to apply the Weight match rule to the data in Table 21–11.

Table 21–11 Example Records for Matching

Record Number	Attr_1	Attr_2
Rec_1	CA	QQ
Rec_2	CA	QQ
Rec_3	CA	QR

For **Maximum score**, assign a value of 50 to both Att_1 and Att_2. Assign a value of 80 for the **Required score to match.** You can expect the following results:

- Rec_1 is the new record. The operator reads it first.
- In Rec_2, the value for Attr_1 is CA. That value has a similarity of 100 with the value in the new record, Rec_1. Since the weight value for Attr_1 is 50, its score is 50 (100% of 50).
- In Rec_2, the value for Attr_2 is QQ and has a similarity of 100. The weight value for Attr_2 is also 50 and its score is therefore 50 (100% of 50). The total maximum score is 100 (50 + 50). This equals or exceeds the value of the **Required score for match**, so Rec_2 and Rec_1 match.
- In Rec_3, Attr_1 is CA and has a similarity of 100 with Rec_1. Since the weight value for Attr_1 is 50, its weighted score is 50 (100% of 50).
- In Rec_3, the value for Attr_2 is QR and that has a similarity of 50. The maximum value for Attr_2 is 50, so its score is 25 (50% of 50). The total weighted score is 75 (50+25). This is less than the value of the **Required score to match.** Therefore, Rec_3 and Rec_1 do not match.

Person Match Rule

Use the Person match rule to match records based on names. Matching by names is most effective when you first correct the address data using the Name and Address operator before the Match-Merge operator.

When you select the Person match rule, the Match Rules page displays the Person Attributes and Details tabs.

Person Attributes Tab

- **Eligible Attributes:** Lists all input attributes.
- **Attributes:** The attributes containing parts of names. Use the shuttle keys to move attributes from Eligible Attributes to the Attributes column of Name Roles.

- **Name Roles:** Lists the roles for different parts of a name. Select the appropriate role for each attribute. Table 21–12 describes the roles.

Table 21–12 Name Roles for Person Match Rules

Role	Description
Prename	The operator compares prenames only if First Name Standardized is blank for one of the records, the 'Mrs.' option is selected, and the Last Name and any Middle Name role match. Given that criteria, the operator matches the record 'Mrs. William Webster' with 'Mrs. Webster'.
First Name Standardized	First names match if both are blank. A blank first name will not match a non-blank first name unless the Prename role has been assigned and the 'Mrs. Match' option is set.
Middle Name Standardized, Middle Name 2 Standardized, Middle Name 3 Standardized	The operator compares and cross compares any of the middle names assigned. By default, the middle names must match exactly. Middle names match if either or both are blank. To assign any of the middle name roles, you must also assign the First Name Standardized role.
Last Name	The operator assigns last names as matching if both are blank and not matching if only one is blank.
Maturity Post Name	This is the same as post names such as 'Jr.' and 'Sr.'. The operator assigns these as matching if the values are exact or if either is blank.

Person Details Tab

Use the Details tab to set options for determining a match. Table 21–13 lists the rule options you can select for each component of the name. For a description of the algorithms, refer to Table 21–10.

Table 21–13 Options for Person Match Rule

Option	Description
Detect switched name order	Detects switched name orders such as matching 'Elmer Fudd' to 'Fudd Elmer'. You can select this option if you selected First Name and Last Name roles for attributes on the Person Attributes tab.
Match on initials	Matches initials to names such as 'R.' and 'Robert'. You can select this option for first name and middle name roles.
Match on substrings	Matches substrings to names such as 'Rob' to 'Robert'. You can select this option for first name and middle name roles.
Similarity Score	Uses a similarity score to determine a match, as calculated by the Edit Distance or Jaro-Winkler algorithms. Enter a value between 0 and 100 as the minimum similarity value required for a match. A value of 100 requires an exact match, and a value of 0 requires no similarity whatsoever.
Match on Phonetic Codes	Determines a match using either the Soundex or the Double Metaphone algorithms.
Detect compound name	Matches compound names to names such as 'De Anne' to 'Deanne'. You can select this option for the first name role.
"Mrs" Match	Matches prenames to first and last names such as 'Mrs. Washington' to 'George Washington'. You can select this option for the prename role.
Match hyphenated names	The operator matches hyphenated names to unhyphenated names such as 'Reese-Jones' to 'Reese'. You can select this option for the last name role.

Table 21–13 (Cont.) Options for Person Match Rule

Option	Description
Detect missing hyphen	The operator detects missing hyphens, such as matching 'Hillary Rodham Clinton' to 'Hillary Rodham-Clinton'. You can select this option for the last name role.

Defining a Person Match Rule

To define a Person match rule, complete the following steps:

1. On the Match Rules tab, select Person as the Rule Type.

   The Person Attributes tab and Details tab are displayed at the bottom of the page.

2. In the left panel of the Person Attributes tab, select the attributes that describe a full name and use the arrow key to move them to Name Roles Attributes.

3. For each attribute, select the role it plays in a name. You must define either the Last Name or First Name Standardized for the match rule to be effective. See Table 21–12 for the types of roles you can assign.

4. Select the Details tab and select the applicable options as listed in Table 21–13.

Firm Match Rule

Use the Firm match rule to match records by business name. This type of match is most effective when you first correct the address data using the Name and Address operator before the Match-Merge operator.

When you select the Firm match rule, the Match Rules page displays the Firm Attributes and Details tabs.

Firm Attributes tab

Use the Firm Attributes tab to identify attributes that contain business names.

- **Eligible Attributes:** Lists all input attributes.
- **Attributes:** The attributes containing the names of businesses. Move one or two attributes containing business names from Eligible Attributes to the Attributes column of Firm Roles.
- **Firm Roles:** Lists two roles, Firm 1 and Firm 2. If you selected one attribute, then designate Firm 1 as its role. If you selected two attributes, then designate one of them as Firm 1 and the other as Firm 2.

Firm Details Page

Use the Details tab to set options for determining a match. By default, the operator compares the values in Firm 1 for exact matches.

Table 21–14 lists the rule options you can select for each component of the name. For a description of the algorithms, refer to Table 21–10.

Table 21–14 Options for Firm Rules

Option	Description
Strip noise words	Removes words such as 'and' and 'the.'
Cross-match firm 1 and firm 2	Attempts to find matches between the Firm 1 attribute and the Firm 2 attribute.

Table 21–14 (Cont.) Options for Firm Rules

Option	Description
Match on partial firm name	Uses the Partial Name algorithm to determine a match.
Match on abbreviations	Uses the Abbreviation algorithm to determine a match.
Match on acronyms	Uses the Acronym algorithm to determine a match.
Similarity score	Uses a similarity score to determine a match, as calculated by the Edit Distance or Jaro-Winkler algorithms. Enter a value between 0 and 100 as the minimum similarity value required for a match. A value of 100 requires an exact match, and a value of 0 requires no similarity whatsoever.

Defining a Firm Match Rule

To define a Firm match rule, complete the following steps:

1. On the Match Rules tab, select **Firm** as the Rule Type.

   The Firm Attributes tab and Details tab are displayed at the bottom of the page.

2. In the left panel of the Firm Attributes tab, select one or two attributes that represent the firm name and click the right shuttle button.

   The attributes move to the Firm Roles box.

3. For each attribute, click **Role Required**. From the list, select Firm 1 for the first attribute, and Firm 2 for the second attribute, if it exists.

4. On the Details tab, select the applicable options.

Address Match Rule

Use the Address match rule to match records based on postal addresses.

Matching by address is most effective when you first correct the address data using the Name and Address operator before the Match-Merge operator. The Name and Address operator identifies addresses as existing in a postal matching database and designates the records with the Is Found flag. The Match-Merge operator processes addresses with the Is Found role faster because the data is known to be syntactically correct, legal, and existing.

Address Attributes tab

Use the Address Attributes tab to identify attributes that contain addresses.

- **Eligible Attributes:** Lists all input attributes.
- **Attributes:** The attributes containing parts of addresses. Use the shuttle keys to move attributes containing address information from Eligible Attributes to the Attributes column of Address Roles.
- **Address Roles:** Lists the various parts of an address. Select the one that most closely matches each attribute. Table 21–15 describes the address roles.

Table 21–15 Address Roles

Role	Description
Primary Address	A street address such as 100 Main Street or a post office box such as PO Box 100. Assign this role to one attribute; otherwise the match rule is invalid.

Table 21–15 (Cont.) Address Roles

Role	Description
Unit Number	Suite numbers, floor numbers, or apartment numbers for the Primary Address. For addresses with matching primary addresses, the operator compares unit numbers. If both unit numbers are blank, then they match. If only one unit number is blank, they match only when you selected the **Match on blank secondary address** option.
PO Box	A post office box number in the Primary Address. The operator compares the post office box number with the number portion of the primary address, when the primary address is a PO Box. When the primary address represents a street address, the PO Box number is blank.
Dual Primary Address	A second address, which may represent either an additional location or a former location. For addresses with matching primary addresses, the operator compares the dual primary addresses
Dual Unit Number	Suite numbers, floor numbers, or apartment numbers for the Dual Primary Address. The operator compares the Dual Unit Number in one record with the Unit Number and Dual Unit Number of another record. Unit numbers match when one or both are blank.
Dual PO Box	A post office box number in the Dual Primary Address. The operator compares the Dual PO Box in one record with both the PO Box and Dual PO Box of another record.
City	Assign this role only when also assigning the State role. For uncorrected address data, the operator compares each City. For addresses already corrected by the Name and Address operator, Match-Merge only compares City when the postal codes do not match. If both City and State match, then the operator compares the address roles. Cities match when both are blank but not when only one is blank.
State	Assign this role only when also assigning the City role. For uncorrected address data, the operator compares each State. For addresses already corrected by the Name and Address operator, Match-Merge only compares States when the postal codes do not match. If both City and State match, then the operator compares the address roles. States match when both are blank but not when only one is blank.
Postal Code	For uncorrected address data, the operator does not use Postal Code. For addresses already corrected by the Name and Address operator, Match-Merge only compares each Postal Code. If the codes match, then the operator compares the address roles. If the codes do not match, then the operator compares City and State to determine if it should compare address roles such as Primary Address.
Is Found	Assign this role to an address that you previously cleansed and standardized with the Name and Address operator. The Name and Address operator marks records with the Is Found flag when it identifies the address as part of a country postal matching database.

Address Details Page

Use the Details tab to set options for determining a match. Table 21–16 describes the options you can assign to Address Roles.

Table 21–16 Options for Address Roles

Option	Description
Allow differing secondary address	Matches addresses with different unit numbers.
Match on blank secondary address	Matches addresses with one blank unit number.
Match on either street or post office box	Matches records if either the street address or the post office box match.
Address line similarity	Uses a similarity score to determine a match in the address lines, as calculated by the Edit Distance or Jaro-Winkler algorithms. Enter a value between 0 and 100 as the minimum similarity value required for a match. A value of 100 requires an exact match, and a value of 0 requires no similarity whatsoever. In an address, the street name is the Primary Address and the unit number is the Secondary Address. Address Line Similarity evaluates both the primary and secondary addresses.
Last line similarity	Uses a similarity score to determine a match in the City, State, and Postal Code attributes, as calculated by the Edit Distance or Jaro-Winkler algorithms. Enter a value between 0 and 100 as the minimum similarity value required for a match. A value of 100 requires an exact match, and a value of 0 requires no similarity whatsoever.

Defining an Address Match Rule

To define an Address match rule, complete the following steps:

1. On the Match Rules tab, select Address as the Rule Type.

   The Address Attributes tab and Details tab are displayed at the bottom of the page.

2. In the left panel of the Address Attributes tab, select the attribute that represents the primary address. Use the right shuttle key to move it to the Address Roles Attributes column.

3. Click **Role Required** and designate that attribute as the Primary Address.

   You must perform this step. If you do not assign the Primary Address role, the match rule is invalid.

4. Add other attributes and designate their roles as necessary. See Table 21–15 for the types of roles you can assign.

5. Select the Details tab and select the applicable options as listed in Table 21–16.

Custom Match Rule

Use the Custom match rule to execute your own PL/SQL program as a comparison algorithm.

Match Rules Detail

Displays the PL/SQL code composing your custom algorithm. You can edit code directly in this field or use the Custom Match Rule Editor.

Edit

Displays the Custom Match Rule Editor.

Custom Match Rule Editor

Use the Custom Match Rule Editor to develop custom PL/SQL programs to use as comparison algorithms.

The editor provides basic program development support. It consists of these components:

- Menu Bar
- Search For Field
- Navigation Tree
- Implementation Field
- Definition Tab
- Messages Tab

Menu Bar

The menu bar contains the following menus:

- **Code**: Enables you to read from and write to files on your local computer network, save your editing changes, and so forth.
- **Edit**: Provides basic cut, copy, and paste functionality.
- **Search**: Provides basic search and replace functionality.
- **Test**: Checks for syntax errors in the code entered in the Implementation field.
- **Help**: Displays this topic.

Search For Field

All or part of a function name you wish to find in the navigation tree. Click **Go** to search for the first instance, click **Go** again to find the next instance, and so forth.

Navigation Tree

Lists input parameters and transformations that you can include in your program. The navigation tree contains these folders:

- **Match Functions**: List both active and passive rules that you have defined. You can call these rules the same as any other PL/SQL function.
- **Parameters**: Lists all input attributes under two subfolders named THIS_ and THAT_. Your program can compare two rows in the same attribute or different ones.
- **Transformation Library**: Lists all Warehouse Builder transformations.

To insert a function, parameter, or transformation into your code at the cursor, double-click or drag-and-drop it into the Implementation field.

Implementation Field

Displays your program code.

Definition Tab

Displays the signature of the selected function.

Messages Tab

Displays information about the success or failure of validating the code.

Defining a Custom Match Rule

To define a Custom match rule, complete the following steps:

1. On the Match Rules tab, select **Custom** as the Rule Type.

   A Details field is displayed at the bottom of the page with the skeleton of a PL/SQL program.

2. Click **Edit** to open the Custom Match Rules Editor.
3. To enter PL/SQL code, do any combination of the following:
   - To read in a file, choose **Open File** from the Code menu.
   - To enter text, first position the cursor using the mouse or arrow keys, then begin typing. You can also use the commands on the Edit and Search menus.
   - To reference any function, parameter, or transformation in the navigation tree, first position the cursor, then double-click or drag-and-drop the object onto the Implementation field.
4. To validate your code, choose **Validate** from the Test menu.

   The validation results appear on the Messages tab.
5. To save your code, choose **Save** from the Code menu.
6. To close the Custom Match Rules Editor, choose **Close** from the Code menu.

Merge Rules Page

Use the Merge Rules page to select values for the attributes in the merged record.

Name

An arbitrary name for the rule. Warehouse Builder creates a default name such as ME_0 for each merge rule. You can replace these names with meaningful ones. Meaningful names are particularly helpful when calling rules from a PL/SQL custom rule.

Position

The order of execution. You can change the position of a rule by clicking on the row header and dragging the row to its new location. The row headers are the boxes to the left of the Name column.

Rule Type

Assign one of the rule types listed in Table 21–17. When you select a rule type, the lower portion of the Merge Rules tab activates and you can enter details for the merge rule.

Attribute

Identifies the attribute whose values are selected by the merge rule. Provides a list of all merged attributes.

When you select a Record rule, Warehouse Builder disables the list of attributes. You select multiple attributes on the Attributes tab, which appears on the lower portion of the page.

Description

An optional description of the rule, which you can enter.

Descriptions of Merge Rules

When you define a merge rule, you can define one rule for all the attributes in the merged record or define a rule for each attribute. For instance, if the merged record is a customer record, it may have attributes such as ADDRESS1, ADDRESS2, CITY, STATE, and ZIP. You can write five rules that select the value of each attribute from up to five different records, or one Record rule that selects that values of all five attributes from one record. Use record rules when multiple attributes compose a logical unit, such as an address. For example, City, State, and Zip Code might be three different attributes, but the data for these attributes should all come from the same record.

Table 21–17 describes the types of merge rules.

Table 21–17 Merge Rule Types

Merge Rule	Description
Any	Uses the first non-blank value.
Match ID	Merges records that have already been output from another Match-Merge operator. See Match ID Merge Rule for details.
Rank	Uses the ranked values in a second attribute to select the preferred value. See Rank and Rank Record Merge Rules for details.
Sequence	Uses the values in a sequence to generate unique keys as the data is loaded. See Sequence Merge Rule for details.
Min Max	Uses the first value based on the order of another attribute. See Min Max and Min Max Record Merge Rules for details.
Copy	Uses the values from another merged attribute. See Copy Merge Rule for details.
Custom	Uses the PL/SQL code that you provide as the criteria for merging records. See Custom and Custom Record Merge Rules for details.
Any Record	Identical to the Any rule, except that an Any Record rule applies to multiple attributes.
Rank Record	Identical to the Rank rule, except that a Rank Record rule applies to multiple attributes
Min Max Record	Identical to the Min Max rule, except that a Min Max Record rule applies to multiple attributes.
Custom Record	Identical to the Custom rule, except that a Custom Record rule applies to multiple attributes.

Match ID Merge Rule

Use the Match ID merge rule to merge records that have already been output in the XREF group from another Match-Merge operator. No other operator is valid for this type of input. For more information, refer to "Using Two Match-Merge Operators" on page 21-5.

Next Value of the Sequence

Identifies the sequence that will be used by the rule.

sequences list

Lists all sequences defined in the current project.

Select Sequence

Sets the sequence for the rule to the sequence currently selected in the list. Move a sequence from the sequences list to Select Sequence.

Rank and Rank Record Merge Rules

Use the Rank and Rank Record rules when merging data from multiple sources. These rules enable you to identify your preference for certain sources. Your data must have a second input attribute on which the rule is based.

For example, the second attribute might identify the data source, and these data sources are ranked in order of reliability. The most reliable value would be used in the merged record. The merge rule might look like this:

```
INGRP1.SOURCE = 'Order Entry'
```

Name

An arbitrary name for the rule. Warehouse Builder creates a default name such as RULE_0 for each rank merge rule. You can replace these names with meaningful ones.

Position

The order of execution. You can change the position of a rule by clicking on the row header and dragging the row to its new location. The row headers are the boxes to the left of the Name column.

Expression Record Selection

The custom SQL expression used in the ranking. Click the Ellipsis button to display the Rank Rule Editor (also called the Expression Builder User Interface). Use this editor to develop the ranking expression.

Sequence Merge Rule

The Sequence rule uses the next value in a sequence.

Next Value of the Sequence

Identifies the sequence that will be used by the rule.

sequences list

Lists all sequences defined in the current project.

Select Sequence

Sets the sequence for the rule to the sequence currently selected in the list.

Min Max and Min Max Record Merge Rules

The Min Max and Min Max Record rules select an attribute value based on the size of another attribute value in the record.

For example, you might select the First Name value from the record in each bin that contains the longest Last Name value.

Selecting Attribute

Lists all input attributes. Select the attribute whose values provide the order.

Attribute Relation

Select the characteristic for choosing a value in the selected attribute.

- **Minimum**. Selects the smallest numeric value or the oldest date value.
- **Maximum**. Selects the largest numeric value or the most recent date value.
- **Shortest**. Selects the shortest character value.
- **Longest**. Selects the longest character value.

Copy Merge Rule

The Copy rule uses the values from another merged attribute.

Merged Attribute

Lists the other merged attributes, which you selected on the Merge Attributes page.

Custom and Custom Record Merge Rules

The Custom and Custom Record rules use PL/SQL code that you provide to merge the records. The following is an example of a Custom merge rule, which returns the value of the TAXID attribute for record 1.

```
BEGIN
RETURN M_MATCHES(1)."TAXID";
END;
```

Following is an example of a Custom Record merge rule, which returns a record for record 1:

```
BEGIN
RETURN M_MATCHES(1);
END;
```

Merge Rules Detail

Displays the PL/SQL code composing your custom algorithm. You can edit code directly in this field or use the Custom Merge Rule Editor.

Edit

Displays the Custom Merge Rule Editor.

Custom Merge Rule Editor

Use the Custom Merge Rule Editor to develop custom PL/SQL programs to use as merging algorithms.

The editor provides basic program development support. It consists of these components:

- Menu Bar
- Search For Field

- Navigation Tree
- Implementation Field
- Definition Tab
- Messages Tab

Menu Bar

The menu bar contains the following menus:

- **Code**: Enables you to read from and write to files on your local computer network, save your editing changes, and so forth.
- **Edit**: Provides basic cut, copy, and paste functionality.
- **Search**: Provides basic search and replace functionality.
- **Test**: Checks for syntax errors in the code entered in the Implementation field.
- **Help**: Displays this topic.

Search For Field

All or part of a function name you wish to find in the navigation tree. Click **Go** to search for the first instance, click **Go** again to find the next instance, and so forth.

Navigation Tree

Lists input parameters and transformations that you can include in your program. The navigation tree contains these folders:

- Parameters: Lists all input attributes under the M_MATCHES folder and the merged attributes under the M_MERGE folder.
- Transformation Library: Lists all Warehouse Builder transformations.

To insert a parameter or a transformation into your code at the cursor, double-click or drag-and-drop it into the Implementation field.

Implementation Field

Displays your program code.

Definition Tab

Displays the signature of the selected function.

Messages Tab

Displays information about the success or failure of validating the code.

Using the Name and Address Operator in a Mapping

The Name and Address operator accepts one PL/SQL input and generates one PL/SQL output.

If you experience time-out errors, you may need to increase the socket time-out setting of the Name and Address Server. The time-out setting is the number of seconds the server will wait for a parsing request from a mapping before the server drops a connection. The default setting is 600 seconds (10 minutes). After the server drops a connection because of inactivity, subsequent parsing requests fail with a NAS-00021 error.

For most mappings, long time lapses between parsing requests are rare. However, maps operating in row based mode with a filter operator may have long time lapses between record parsing requests, because of the inefficiency of filtering records in row based mode. For this type of mapping, you may need to increase the socket time-out value to prevent connections from being dropped.

To increase the socket time-out setting, refer to "Configuring the Name and Address Server" on page 21-42.

Name and Address Wizard and Editor: General

Use the **General** page to specify a name and optional description for the operator. By default, the wizard names the Name and Address operator `NAMEADDR`.

Name and Address Wizard and Editor: Definitions

Characterize the nature of your input data by assigning general definitions to this Name and Address operator. In the **Definitions** page, select a Parsing Type, Primary Country, and Dual Address Assignment.

Parsing Type

Select one of the following parsing types from the drop-down list:

> **Note:** You can only specify the parsing type when you first add the Name and Address operator to your mapping. You cannot modify the parsing type in the editor.

- **Name Only:** Select this option when the input data contains only name data. Names can include both personal and business names. Selecting this option instead of the more generic Name and Address option may improve performance and accuracy, depending on the adapter.
- **Address Only:** Select this option when the input data contains only address data and no name data. Selecting this option instead of the more generic Name and Address option may improve performance and accuracy, depending on the adapter.
- **Name and Address:** Select this option when the input data contains both name and address data.

Primary Country

Select the country that best represents the country distribution of your data. The primary country is used by some providers of name and address cleansing software as a hint for the appropriate parser or parsing rules to use on the initial parse of the record. For other name and address service providers, external configuration of their installation controls this behavior.

Dual Address Assignment

A dual address contains both a Post Office (PO) box and a street address for the same address record. For records that have dual addresses, your selection determines which address becomes the *normal address* and which address becomes the *dual address*. A sample dual address is:

```
PO Box 2589
4439 Mormon Coulee Rd
```

```
La Crosse WI 54601-8231
```

Note that the choice for Dual Address Assignment affects which postal codes are assigned during postal code correction, because the street address and PO box address may correspond to different postal codes.

- **Street Assignment:** The street address is the *normal address* and the PO Box address is the *dual address*. This means that the `Address` component is assigned the street address. In the preceding example, the `Address` is 4439 MORMON COULEE RD. This choice corrects the postal code to 54601-8220.
- **PO Box Assignment:** The PO Box address is the *normal address* and the street address is the *dual address*. This means that the `Address` component is assigned the Post Office (PO) box address. In the preceding example, the `Address` is PO BOX 2589. This choice corrects the postal code to 54602-2589.
- **Closest to Last Line:** Whichever address occurs closest to the last line is the *normal address*; the other is the *dual address*. This means that the `Address` component is assigned the address line closest to the last line. In the preceding example, the `Address` is 4439 MORMON COULEE RD. This choice corrects the postal code to 54601-8220.

This option has no effect for records having a single street or PO box address.

Note: Dual Address Assignment may not be supported by all name and address cleansing software providers.

Name and Address Wizard and Editor: Groups

The **Groups** page lists the input and output groups defined for an operator. By definition, the Name and Address operator has one input group and one output group. You cannot edit, add, or delete groups in the Name and Address operator. The input group is called INGRP1 and the output group is OUTGRP1. You can edit these names. If the input data requires multiple groups, create a separate Name and Address operator for each group.

Name and Address Wizard and Editor: Input Connections

Use the **Input Connections** page to select attributes from any operator in your mapping that you want to copy and map into the operator. The Available Attributes box lists the available attributes. The Mapped Attributes box lists the attributes that will be processed by the Name and Address operator. You can move an entire group of attributes, or individual attributes from a single group.

If you have not created one or more operators for the source data yet, the Available Attributes column will be empty.

To complete the Input Connections page for an operator:

1. Select complete groups or individual attributes from the Available Attributes panel. The Available Attributes panel enables you to select attributes from any operator in your mapping.

   To search for a specific attribute or group by name, type the text in **Search for** and click **Go.** To find the next match, click **Go** again.

   Hold the **Shift** key down to select multiple groups or attributes. If you want to select attributes from different groups, you must first combine the groups with a Joiner or Set operator.

2. Use the left to right arrow button between the two panels to move your selections to the Mapped Attributes panel.

Name and Address Wizard and Editor: Input Attributes

Use the **Input Attributes** page to assign input roles to each input attribute.

Attribute

Automatically lists the attributes that you selected on the Input Connections page. Otherwise, the list may be empty. Click **Add** to add each input attribute.

Map all attributes through the Name and Address operator, including those that you do not wish to process. Bypassing the Name and Address operator was valid in earlier releases of Warehouse Builder, but may cause problems now. Instead, assign these attributes the `Pass Through` input role in the Name and Address operator.

Input Role

Input roles indicate what kind of name or address information resides in a line of data. For each attribute, select the input role that most closely matches the data contained in the source attribute. Refer to "Input Role Descriptions" on page 21-27 for a complete list of input roles and their descriptions.

Whenever possible, choose discrete roles (such as `City`, `State`, and `Postal Code`) rather than non-discrete, line-oriented roles (such as `Last Line`). Discrete roles give the Name and Address operator more information about the data content and result in better parsing.

Data Type, Length, Precision, Scale, Seconds Precision

Set the data type and related parameters for attributes given the `Pass Through` input role. Attributes with other input roles are set automatically to VARCHAR2 and cannot be changed.

Description

Enter an optional description of the input attributes.

Input Role Descriptions

Table 21–18 describes the input roles for the Name and Address Operator.

Table 21–18 Name and Address Operator Input Roles

Input Role	Description
Pass Through	Any attribute that requires no processing.
First Name	First name, nickname, or shortened version of the first name.
Middle Name	Middle name or initial. Use when there is only one middle name, or for the first of several middle names; for example, 'May' in Ethel May Roberta Louise Mertz.
Middle Name 2	Second middle name; for example, 'Roberta' in Ethel May Roberta Louise Mertz.
Middle Name 3	Third middle name; for example, 'Louise' in Ethel May Roberta Louise Mertz.
Last Name	Last name or surname.

Table 21–18 (Cont.) Name and Address Operator Input Roles

Input Role	Description
First Part Name	First part of the Person name, including: ■ Pre name ■ First name ■ Middle name(s) Use when these components are contained in one source column.
Last Part Name	Last part of Person Name, including: ■ Last name ■ Post Name Use when these components are all contained in one source column.
Pre Name	Information that precedes and qualifies the name; for example, Ms., Mr., or Dr.
Post Name	Generation or other information qualifying the name; for example, Jr. or Ph.D.
Person	Full person name, including: ■ First Part Name (consisting of Pre Name, First Name, and Middle Names) ■ Last Part Name (consisting of Last Name and Post Name) Use when these components are all contained in one source column.
Person 2	Designates a second person if the input includes multiple personal contacts.
Person 3	Designates a third person if the input includes multiple personal contacts.
Firm Name	Name of the company or organization.
Primary Address	Box, route, or street address, including: ■ Street name ■ House number ■ City map grid direction; for example, SW or N ■ Street type; for example, Avenue, Street, or Road. This does not include the Unit Designator or the Unit Number.
Secondary Address	The second part of the street address, including: ■ Unit Designator ■ Unit Number For example, in a secondary address of Suite 2100, the Unit Designator is STE (a standardization of 'Suite') and the Unit Number is 2100.
Address	Full address line, including: ■ Primary Address ■ Secondary Address Use when these components share one column.
Address 2	Generic address line.
Neighborhood	Neighborhood or barrio, common in South and Latin American addresses.
Locality Name	The city (shi) or island (shima) in Japan.
Locality 2	The ward (ku) in Japan.

Table 21–18 (Cont.) Name and Address Operator Input Roles

Input Role	Description
Locality 3	The district (machi) or village (mura) in Japan.
Locality 4	The subdistrict (aza, bu, chiwari, or sen) in Japan.
City	Name of city.
State	Name of state or province.
Postal Code	Postal code, such as a ZIP code in the United States or a postal code in Canada.
Country Name	Full country name.
Country Code	The ISO 3166-1993 (E) two- or three-character country code. For example, US or USA for United States; CA or CAN for Canada.
Last Line	Last address line, including: ■ City ■ State or province ■ Postal code Use when these components are all contained in one source column.
Last Line 2	For Japanese adaptors, specifies additional line information that appears at the end of an address.
Line1... Line10	Use for free-form name, business, personal, and address text of any type. These roles do not provide the parser with any information about the data content. Whenever possible, use the discrete input roles provided instead.

Name and Address Wizard and Editor: Output Attributes

Use the **Output Attributes** page to define output attributes that determine how the Name and Address operator handles parsed data. Specifically, the output attribute properties characterize the data extracted from the parser output.

The Output Attributes page is empty initially. You can create and edit attributes.

Note: The attributes for output components with the Pass Through role cannot be changed.

Add button

To create a new output attribute, click **Add.**

Attribute column

A new attribute has a default name such as OUTPUT1. This default name changes when you select an output component to a default descriptive name, such as `Primary_Address`. It does not change if you already replaced the name. Click the name to replace it with a new name.

Output Component column

Select an output component for every output attribute. Click the Ellipsis button to the right of the cell to open the Output Attribute Components dialog. See "Descriptions of Output Components" on page 21-30 for a complete list of output components.

Be sure to add error handling flags, such as `Is Parsed`, `Is Good Name`, and `Is Good Address`. These flags can be used with the Splitter operator to separate good records from records with errors, and load them into different targets.

Data Type column

Pass Through output components retain their input data type. All other output component types are VARCHAR2. This column is read-only.

Length column

Adjust the field length to match the length of the target attribute to which you intend to map the output attribute. This practice helps prevent data truncation warnings during code generation, or errors during execution.

Output Attribute Components Dialog Box

Use the **Output Attribute Components** dialog to select a component for each output attribute. The output components indicate which component an attribute constitutes, such as the first name, last name, street name, city, or state.

Select an Output Component

Select a component from the navigation tree to apply to the output attribute. See "Descriptions of Output Components" on page 21-30 for a description of these components.

Output Component

Identifies the component currently selected from the tree. If this field is empty, the current selection is a folder. Expand the folder and select a valid component. Note that some folders are valid components.

Address Type

Available only for dual addresses, and not supported by all name and address cleansing software providers. The Dual Address Assignment option you specified in the Definitions Page page determines whether the street address or the PO box address is used as the dual address. Select either NORMAL or DUAL. For more information on dual addresses, see "Dual Address Assignment" on page 21-25.

Instance

Specify which instance of an output component to use when there are multiple occurrences of the same attribute in a single record. The instance control applies to all name components and several address components, such as `Miscellaneous Address` and `Complex`. This setting enables you to extract numerous attributes of the same nature. The number of instances allowed for various components depends on the third-party name and address cleansing software.

For example, an input record containing John and Jane Doe has two name occurrences: John Doe and Jane Doe. You can extract John Doe by assigning Instance 1 to the First Name and Last Name components. Similarly, you can extract Jane Doe by assigning Instance 2 to the First Name and Last Name components.

Descriptions of Output Components

Output components are grouped in the following categories:

- Pass Through

- Name
- Address
- Extra Vendor
- Error Status
- Country-Specific

Pass Through

The `Pass Through` output component is for any attribute that requires no processing. When you create a `Pass Through` input role, the corresponding `Pass Through` output component is created automatically. You cannot edit a `Pass Through` output component, but you can edit the corresponding input role.

Name

Table 21–19 describes the Name output components. Many components can be used multiple times to process a record, as noted in the table. For example, in records with two occurrences of Firm Name, you can extract both by adding two output attributes. Assign one as the First instance, and the other as the Second instance.

Table 21–19 Name Output Components

Subfolder	Output Component	Description
None	Pre Name	Title or salutation appearing before a name; for example, Ms. or Dr. Can be used multiple times.
None	First Name Standardized	Standard version of first name; for example, Theodore for Ted or James for Jim. Can be used multiple times.
None	Middle Name Standardized	Standardized version of the middle name; for example, Theodore for Ted or James for Jim. Use when there is only one middle name, or for the first of several middle names. Can be used multiple times.
None	Middle Name 2 Standardized	Standardized version of the second middle name; for example, Theodore for Ted or James for Jim. Can be used multiple times.
None	Middle Name 3 Standardized	Standardized version of the third middle name; for example, Theodore for Ted or James for Jim. Can be used multiple times.
None	Post Name	Name suffix indicating generation; for example, Sr., Jr., or III. Can be used multiple times.
None	Other Post Name	Name suffix indicating certification, academic degree, or affiliation; for example, Ph.D., M.D., or R.N. Can be used multiple times.
None	Title	Personal title, for example, Manager.
None	Name Designator	Personal name designation; for example, ATTN (to the attention of) or C/O (care of). Can be used multiple times.
None	Relationship	Information related to another person; for example, Trustee For. Can be used multiple times.
None	SSN	Social security number.
None	Email Address	E-mail address.

Table 21–19 (Cont.) Name Output Components

Subfolder	Output Component	Description
None	Phone Number	Telephone number.
None	Name/Firm Extra	Extra information associated with the firm or personal name.
None	Person	First name, middle name, and last name. Can be used multiple times.
Person	First Name	The first name found in the input name. Can be used multiple times.
Person	Middle Name	Middle name or initial. Use this for a single middle name, or for the first of several middle names; for example, 'May' in Ethel May Roberta Louise Mertz. Can be used multiple times.
Person	Middle Name 2	Second middle name; for example, 'Roberta' in Ethel May Roberta Louise Mertz. Can be used multiple times.
Person	Middle Name 3	Third middle name; for example, 'Louise' in Ethel May Roberta Louise Mertz. Can be used multiple times.
Person	Last Name	Last name or surname. Can be used multiple times.
Derived	Gender	Probable gender: ■ M = Male ■ F = Female ■ N = Neutral (either male or female) ■ Blank = Unknown Can be used multiple times.
Derived	Person Count	Number of persons the record references; for example, a record with a Person name of 'John and Jane Doe' has a Person Count of 2.
Business	Firm Name	Name of the company or organization, including divisions. Can be used multiple times.
Business	Firm Count	Number of firms referenced in the record. Can be used multiple times.
Business	Firm Location	Location within a firm; for example, Accounts Payable

Address

Table 21–20 describes the Address output components. In records with dual addresses, you can specify which line is used as the Normal Address (and thus assigned to the Address component) and which is used as the Dual Address for many output components, as noted in the table.

Table 21–20 Address Output Components

Subfolder	Output Component	Description
None	Address	Full address line, including: ■ Primary Address ■ Secondary Address Can be used as the Normal Address or the Dual Address.

Table 21–20 (Cont.) Address Output Components

Subfolder	Output Component	Description
None	Primary Address	Box, route, or street address, including: ■ Street name ■ House number ■ City map grid direction; for example, SW or N ■ Street type; for example, Avenue, Street, or Road. Does not include the `Unit Designator` or the `Unit Number`. Can be used as the Normal Address or the Dual Address.
Primary Address	Street Number	Number that identifies the address, such as a house or building number, sometimes referred to as the primary range. For example, in 200 Oracle Parkway, the `Street Number` value is 200. Can be used as the Normal Address or the Dual Address.
Primary Address	Pre Directional	Street directional indicator appearing before the street name; for example, in 100 N University Drive, the `Pre Directional` value is 'N'. Can be used as the Normal Address or the Dual Address.
Primary Address	Street Name	Name of street. Can be used as the Normal Address or the Dual Address.
Primary Address	Primary Name 2	Second street name, often used for addresses at a street intersection.
Primary Address	Street Type	Street identifier; for example, ST, AVE, RD, DR, or HWY. Can be used as the Normal Address or the Dual Address.
Primary Address	Post Directional	Street directional indicator appearing after the street name; for example, in 100 15th Ave. S., the `Post Directional` value is 'S'. Can be used as the Normal Address or the Dual Address.
None	Secondary Address	The second part of the street address, including: ■ Unit Designator ■ Unit Number For example, in a secondary address of Suite 2100, `Unit Designator` is 'STE' (a standardization of 'Suite') and `Unit Number` is '2100'. Can be used as the Normal Address or the Dual Address.
Secondary Address	Unit Designator	Type of secondary address, such as APT or STE. For example, in a secondary address of Suite 2100, `Unit Designator` is 'STE' (a standardization of 'Suite'). Can be used as the Normal Address or the Dual Address.
Secondary Address	Unit Number	A number that identifies the secondary address, such as the apartment or suite number. For example, in a secondary address of Suite 2100, `Unit Number` is '2100'. Can be used as the Normal Address or the Dual Address.
Secondary Address	Non-postal Secondary Address	A secondary address that is not in official postal format.
Secondary Address	Non-postal Unit Designator	A unit designator that is not in official postal format.

Table 21–20 (Cont.) Address Output Components

Subfolder	Output Component	Description
Secondary Address	Non-postal Unit Number	A unit number that is not in official postal format.
Address	Last Line	Final address line, including: ■ City ■ state, province, or county ■ Formatted postal code if the address was fully assigned
Last Line	Neighborhood	Neighborhood or barrio, common in South and Latin American addresses.
Last Line	City	Name of city. The U.S. city names may be converted to United States Postal Service preferred names.
Last Line	City Abbreviated	Abbreviated city name, composed of 13 characters for the United States.
Last Line	City Abbreviated 2	Alternative abbreviation for the city name.
Last Line	Alternate City	An alternate name for a city that may be referenced by more than one name. In the United States, a city may be referenced by its actual name or the name of a larger urban area. For example, Brighton Massachusetts may have Boston as an alternate city name.
Last Line	Locality Code	The last three digits of the International Mailsort Code, which represents a geographical region or locality within each country. Locality Codes are numeric in the range 000-999.

Last Line	Locality Name	In the United Kingdom, the following address is assigned Locality Name KNAPHILL:
		`Chobham Rd`
		`Knaphill`
		`Woking GU21 2TZ`

Last Line	Locality 2	The ward (ku) in Japan.
Last Line	Locality 3	The district (machi) or village (mura) in Japan.
Last Line	Locality 4	The subdistrict (aza, bu, chiwari, or sen) in Japan.
Last Line	County Name	The name of a county in the United Kingdom, United States, or other country.
Last Line	State	Name of state or province.
Last Line	Postal Code	Full postal code with spaces and other non-alphanumeric characters removed.
Last Line	Postal Code Formatted	Formatted version of postal code that includes spaces and other non-alphanumeric characters, such as dashes.
Last Line	Delivery Point	A designation used in the United States and Australia. ■ For the United States, this is the two-digit postal delivery point, which is combined with a full nine-digit postal code and check digit to form a delivery point bar code. ■ For Australia, this is a nine-digit delivery point.

Table 21–20 (Cont.) Address Output Components

Subfolder	Output Component	Description
Last Line	Country Code	The ISO 3166-1993 (E) two-character country code, as defined by the International Organization for Standardization; for example, 'US' for United States or 'CA' for Canada.
Last Line	Country Code 3	The ISO 3166-1993 (E) three-character country code, as defined by the International Organization for Standardization; for example, 'USA' for United States, 'FRA' for France, or 'UKR' for Ukraine.
Last Line	Country Name	The full country name.
Address	Address 2	A second address line, typically used for Hong Kong addresses that have both a street address and a building or floor address.
Address	Last Line 2	Additional information that appears at the end of an address in Japan.
Other Address Line	Box Name	The name for a post office box address; for example, for 'PO Box 95', the Box Name is 'PO BOX'. Can be used as the Normal Address or the Dual Address.
Other Address Line	Box Number	The number for a post office box address; for example, for 'PO Box 95', the Box Number is '95'. Can be used as the Normal Address or the Dual Address.
Other Address Line	Route Name	Route name for a rural route address. For an address of 'Route 5 Box 10', the Route Name is 'RTE' (a standardization of 'Route'). Can be used as the Normal Address or the Dual Address.
Other Address Line	Route Number	Route number for a rural route address. For an address of 'Route 5 Box 10', the Route Number is '5'. Can be used as the Normal Address or the Dual Address.
Other Address Line	Building Name	Building name, such as 'Cannon Bridge House'. Building names are common in the United Kingdom.

Other Address Line	Complex	Building, campus, or other complex. For example,
		`USS John F. Kennedy`
		`Shadow Green Apartments`
		`Cedarvale Gardens`
		`Concordia College`
		You can use an the Instance field in the Output Components dialog to specify which complex should be returned in cases where an address has more than one complex.

Other Address Line	Miscellaneous Address	Miscellaneous address information. In records with multiple miscellaneous fields, you can extract them by specifying which instance to use in the Output Components page.
Geography	Latitude	Latitude in degrees north of the equator: Positive for north of the equator; negative for south (always positive for North America).
Geography	Longitude	Longitude in degrees east of the Greenwich Meridian: positive for east of GM; negative for west (always negative for North America).
Geography	Geo Match Precision	Indicates how closely the location identified by the latitude and longitude matches the address.

Extra Vendor

Twenty components are open for vendor-specified uses.

Error Status

Table 21–21 describes the Error Status output components. Refer to "Handling Errors in Name and Address Data" on page 10-15 for usages notes on the Error Status components.

Table 21–21 Error Status Output Components

Subfolders	Output Component	Description
Name and Address	Is Good Group	Indicates whether the name group, address group, or name and address group was processed successfully. ■ T = For name groups, the name has been successfully parsed. For address groups, the address has been found in a postal matching database if one is available, or has been successfully parsed if no postal database is installed. For name and address groups, both the name and the address have been successfully processed. ■ F = The group was not parsed successfully. Using this flag in conjunction with another flag, such as the `Is Parsed` flag, followed by the Splitter operator, enables you to isolate unsuccessfully parsed records in their own target, where you can address them separately.
Name and Address	Is Parsed	Indicates whether the name or address was parsed: ■ T = The name or address was parsed successfully, although some warning conditions may have been flagged. ■ F = The name or address cannot be parsed. Check the status of warning flags such as `Name Warning` or `City Warning`.
Name and Address	Parse Status	Postal matching software parse status code.
Name and Address	Parse Status Description	Text description of the postal matching software parse status.
Name Only	Is Good Name	Indicates whether the name was parsed successfully: ■ T = The name was parsed successfully, although some warning conditions may have been flagged. ■ F = The name cannot be parsed.
Name Only	Name Warning	Indicates whether the parser found unusual or possibly erroneous data in a name: ■ T = The parser had difficulty parsing a name or found unusual data. Check the `Parse Status` component for the cause of the warning. ■ F = No difficulty parsing name.

Table 21–21 (Cont.) Error Status Output Components

Subfolders	Output Component	Description
Address Only	Is Good Address	Indicates whether the address was processed successfully: ■ T = Successfully processed. Either the address was found in the postal matching database or, if no postal matching database is installed for the country indicated by the address, the address was successfully parsed. ■ F = Not successfully processed. If a postal matching database is installed for the country indicated by the address, the address was not found in the database. If no postal matching database is available for the country, the address cannot be parsed. Use this component when you have a mix of records from both postal-matched and non-postal-matched countries.
Address Only	Is Found	Indicates whether the address is listed in the postal matching database for the country indicated by the address: ■ T = The address was found in a postal matching database. ■ F = The address was not found in a postal matching database. This status may indicate either that the address is not a legal address, or that postal matching is not available for the country. This flag is true only if all of the other 'Found' flags are true. If postal matching is available, this flag is the best indicator of record quality.
Address Only: Is Found	City Found	T = The postal matcher found the city; otherwise, F.
Address Only: Is Found	Street Name Found	T = The postal matcher found the street name; otherwise, F.
Address Only: Is Found	Street Number Found	T = The postal matcher found the street number within a valid range of numbers for the named street, otherwise, F.
Address Only: Is Found	Street Components Found	T = The postal matcher found the street components, such as the Pre Directional or Post Directional; otherwise, F.
Address Only: Is Found	Non-ambiguou s Match Found	Indicates whether the postal matcher found a matching address in the postal database: ■ T = The postal matcher found a match between the input record and a single entry in the postal database. ■ F = The address is ambiguous. The postal matcher found that the address matched several postal database entries and could not make a selection. For example, if the input address is '100 4th Avenue,' but the postal database contains '100 4th Ave N' and '100 4th Ave S,' the input's missing directional causes the match to fail.
Address Only	City Warning	T = The parser found unusual or possibly erroneous data in a city; otherwise, F.

Table 21–21 (Cont.) Error Status Output Components

Subfolders	Output Component	Description
Address Only	Street Warning	T = The parser found unusual or possibly erroneous data in a street address otherwise, F.
Address Only	Is Address Verifiable	T = Postal matching is available for the country of the address; otherwise, F. F does not indicate whether or not a postal matching database is installed for the country in the address. It only indicates that matching is not available for a particular address.
Address Only	Address Corrected	Indicates whether the address was corrected in any way during matching. Standardization is not considered correction in this case. ■ T = Some component of the address was changed, aside from standardization. One of the other Corrected flags must also be true. ■ F = No components of the address were changed, with the possible exception of standardization.
Address Only: Address Corrected	Postal Code Corrected	T = The postal code was corrected during matching, possibly by the addition of a postal extension; otherwise, F.
Address Only: Address Corrected	City Corrected	T = The city name was corrected during matching; otherwise, F. Postal code input is used to determine the city name preferred by the postal service.
Address Only: Address Corrected	Street Corrected	T = The street name was corrected during matching; otherwise, F. Some correct street names may be changed to an alternate name preferred by the postal service.
Address Only: Address Corrected	Street Components Corrected	T = One or more street components, such as `Pre Directional` or `Post Directional`, were corrected during matching.
Address Only	Address Type	Type of address. The following are common examples; actual values vary with vendors of postal matching software: ■ B= Box ■ F = Firm ■ G= General Delivery ■ H= High-rise apartment or office building ■ HD= High-rise default, where a single Zip+4 postal code applies to the entire building. The Name and Address operator can detect a finer level of postal code assignment if a floor or suite address is provided, in which case the record is treated as an H type, with a more specific Zip+4 code for that floor or suite. ■ M= Military ■ P= Post Office Box ■ R= Rural Code ■ S= Street

Table 21–21 (Cont.) Error Status Output Components

Subfolders	Output Component	Description
Address Only	Parsing Country	Country parser that was used for the final parse of the record.

Country-Specific

Table 21–22 describes the output components that are specific to a particular country.

Table 21–22 Country-Specific Output Components

Subfolder	Output Component	Description
United States	ZIP5	The five-digit United States postal code.
United States	ZIP4	The four-digit suffix that is added to the five-digit United States postal code to further specify location.
United States	Urbanization Name	Urban unit name used in Puerto Rico.
United States	LACS Flag	T = Address requires a LACS conversion and should be submitted to a LACS vendor; otherwise, F. The Locatable Address Conversion System (LACS) provides new addresses when a 911 emergency system has been implemented. 911 address conversions typically involve changing rural-style addresses to city-style street addresses, but they may involve renaming or renumbering existing city-style addresses.
United States	CART	Four-character USPS Carrier route.
United States	DPBC Check Digit	Check digit for forming a delivery point bar code.
United States	Automated Zone Indicator	T = The mail in this zip code is sorted by bar code sorting equipment; otherwise, F.
United States	Urban Indicator	T = An address is located within an urban area; otherwise, F.
United States	Line of Travel	United States Postal Service (USPS) line of travel
United States	Line of Travel Order	United States Postal Service (USPS) line of travel order
United States: Census/Geography	Metropolitan Statistical Area	Metropolitan Statistical Area (MSA) number. For example, '0000' indicates that the address does not lie within any MSA, and typically indicates a rural area.
United States: Census/Geography	Minor Census District	Minor Census District.
United States: Census/Geography	CBSA Code	A 5-digit Core Based Statistical Area code that identifies metropolitan and micropolitan areas.
United States: Census/Geography	CBSA Descriptor	Indicates whether the CBSA is metropolitan (population of 50,000 or more) or micropolitan (population of 10,000 to 49,999).
United States: Census/Geography	FIPS Code	The complete (state plus county) code assigned to the county by the Federal Information Processing Standard (FIPS). Because FIPS county codes are unique within a state, a complete FIPS Code includes the two-digit state code followed by the three-digit county code.

Table 21–22 (Cont.) Country-Specific Output Components

Subfolder	Output Component	Description
United States: Census/Geography	FIPS County	The three-digit county code as defined by the Federal Information Processing Standard (FIPS).
United States: Census/Geography	FIPS Place Code	The five-digit place code as defined by the Federal Information Processing Standard (FIPS).
United States: Geography	Census ID	United States Census tract and block-group number. The first six digits are the tract number; the final digit is the block-group number within the tract. These codes are used for matching to demographic-coding databases.
Canada	Installation Type	A type of Canadian postal installation: ■ STN= Station ■ RPO = Retail Postal Outlet For example, for the address, 'PO Box 7010, Scarborough ON M1S 3C6,' the Installation Type is 'STN'.
Canada	Installation Name	Name of a Canadian postal installation. For example, for the address, 'PO Box 7010, Scarborough ON M1S 3C6,' the Installation Name is 'AGINCOURT'.

Hong Kong	Delivery Office Code	A mailing code used in Hong Kong. For example, the following address is assigned the Delivery Office Code 'WCH':
		`Oracle`
		`39/F The Lee Gardens`
		`33 Hysan Ave`
		`Causeway Bay`

Hong Kong	Delivery Beat Code	A mailing code used in Hong Kong. For example, the following address is assigned the Delivery Beat Code 'S06':
		`Oracle`
		`39/F The Lee Gardens`
		`33 Hysan Ave`
		`Causeway Bay`

Name and Address Wizard and Editor: Postal Reporting Page

Postal reporting applies only to countries that support address correction and postal matching. Country certification varies with different vendors of name and address cleansing software. The most common country certifications are United States, Canada, and Australia. The process provides mailers a common platform to measure the quality of address-matching software, focusing on the accuracy of postal codes (in the case of the United States, of five-digit ZIP Codes and ZIP+4 Codes), delivery point codes, and carrier route codes applied to all mail. Some vendors of name and address cleansing software may ignore these parameters and require external setup for generating postal reports. For more information, see "About Postal Reporting" on page 21-41.

Postal Report

Select **Yes** for a postal report for the Primary Country you chose in the Definitions Page page. Only one postal report can be active.

Processor Name

The use of this field varies with vendors of name and address cleansing software. Typically, this value appears on the United States Coding Accuracy Support System (CASS) report.

List Name

An optional reference field that appears on the United States and United Kingdom reports under the List Name section, but is not included in other reports. The list name provides a reference for tracking multiple postal reports; for example, 'July 2005 Promotional Campaign'.

Processor Address Lines

These address lines may appear on various postal reports. Various name and address cleansing software vendors use these fields differently. They often contain the full address of your company.

About Postal Reporting

All address lists used to produce mailings for discounted automation postal rates must be matched by postal report-certified software. Certifications depend on the third-party vendors of name and address software and data. The certifications may include the following:

- **United States Postal Service:** Coding Accuracy Support System (CASS)
- **Canada Post:** Software Evaluation and Recognition Program (SERP)
- **Australia Post:** Address Matching Approval System (AMAS)

United States Postal Service CASS Certification

The Coding Accuracy Support System (CASS) was developed by the United States Postal Service (USPS) in cooperation with the mailing industry. The system provides mailers a common platform to measure the quality of address-matching software, focusing on the accuracy of five-digit ZIP Codes, ZIP+4 Codes, delivery point codes, and carrier route codes applied to all mail. All address lists used to produce mailings for automation rates must be matched by CASS-certified software.

To meet USPS requirements, the mailer must submit a CASS report in its original form to the USPS.

Canada Post SERP Certification

Canada Post developed a testing program called Software Evaluation and Recognition Program (SERP), which evaluates software packages for their ability to validate, or validate and correct, mailing lists to Canada Post requirements. Postal programs that meet SERP requirements are listed on the Canada Post Web site.

Canadian postal customers who use Incentive Lettermail, Addressed Admail, and Publications Mail must meet the Address Accuracy Program requirements. Customers can obtain a Statement of Accuracy by comparing their databases to Canada Post's address data.

Australia Post AMAS Certification

The Address Matching Approval System (AMAS) was developed by Australia Post to improve the quality of addressing. It provides a standard by which to test and measure the ability of address-matching software to:

- Correct and match addresses against the Postal Address File (PAF)
- Append a unique Delivery Point Identifier (DPID) to each address record, which is a step toward barcoding mail.

AMAS allows companies to develop address matching software which:

- Prepares addresses for barcode creation
- Ensures quality addressing
- Enables qualification for discounts on PreSort letter lodgements

PreSort Letter Service prices are conditional upon customers using AMAS Approved Software with Delivery Point Identifiers (DPIDs) being current against the latest version of the PAF.

A declaration that the mail was prepared appropriately must be made when using the Presort Lodgement Document, available from post offices.

Managing the Name and Address Server

An external Name and Address server provides an interface between Oracle Database and third-party name and address processing libraries. This section discusses methods of configuring, starting, and stopping the Name and Address Server.

Configuring the Name and Address Server

The Name and Address operator generates PL/SQL code, which calls the `UTL_NAME_ADDR` package installed in the Runtime Schema. A private synonym, `NAME_ADDR`, is defined in the target schema to reference the `UTL_NAME_ADDR` package. The `UTL_NAME_ADDR` package calls Java packages, which send processing requests to an external Name and Address server, which then interfaces with third-party Name and Address processing libraries, such as Trillium.

You can use the server property file, `NameAddr.properties`, to configure server options. This file is located in `owb/bin/admin` under the Oracle home of your Oracle Warehouse Builder server-side installation. The following code illustrates several important properties with their default settings.

```
TraceLevel=0
SocketTimeout=180
ClientThreads=4
Port=4040
```

The `TraceLevel` property is often changed to perform diagnostics on server communication and view output from the postal matching program parser. Other properties are rarely changed.

- **TraceLevel:** Enables output of file `NASvrTrace.log` in the `owb/bin/admin` folder. This file shows all incoming and outgoing data, verifies that your mapping is communicating with the Name and Address Server, and that the Name and Address Server is receiving output from the service provider. The trace log shows all server input and output and is most useful for determining whether any parsing requests are being made by an executing mapping. Set `TraceLevel=1` to enable logging. However, tracing degrades performance and creates a large log file. Set `TraceLevel=0` to disable logging for production.
- **SocketTimeOut:** Specifies the number of seconds the name/address server will wait for a parsing request before closing the connection. You can increase this time to 1800 (30 minutes) when running concurrent mappings to prevent timing out.

- **ClientThreads:** Specifies the number of threads used to service client connections. One client connection is made for each database session or slave session if a map is parallelized. Most maps are parallelized, and the number of parallel processes is proportional to the number of processors. On a single processor computer, two parallel processes are spawned for large maps. On a four processor computer, up to eight processes may be spawned. Parallelism may also be controlled by database initialization settings such as Sessions.

  For the best performance, set ClientThreads to the maximum number of clients that will be connected simultaneously. The actual number of connected clients is recorded in `NASvr.log` after a map run. You should increase the value of ClientThreads when the number of client connections shown in the log is greater.

  When the number of clients exceeds the number of threads, all clients are still serviced because the threads are shared among clients.

- **Port:** Specifies the port on which the server listens and was initially assigned by the installer. This value may be changed if the default port conflicts with another process. If the port is changed, the port attribute must also be changed in the *runtime_schema*`.nas_connection` table to enable the `utl_name_addr` package to establish a connection.

Starting and Stopping the Name and Address Server

Whenever you edit the properties file or perform table maintenance, you must stop and restart the Name and Address Server for the changes to take effect.

To manually stop the Name and Addresss Server:

- In Windows, run *OWB_ORACLE_HOME*`/owb/bin/win32/NAStop.bat`.
- In UNIX, run *OWB_ORACLE_HOME*`/owb/bin/unix/NAStop.sh`.

You can automatically restart the Name and Address Server by invoking a mapping in Warehouse Builder. You can also restart the server manually.

To manually restart the Name and Address Server:

- In Windows, run *OWB_ORACLE_HOME*`/owb/bin/win32/NAStart.bat`.
- In UNIX, run *OWB_ORACLE_HOME*`/owb/bin/unix/NAStart.sh`.

Part IV

ETL Design Reference

This part contains the following chapters:

22

Using Activities in Process Flows

After you design mappings that define the operations for moving data from sources to targets, you can create and define process flows. Use process flows to interrelate mappings and activities external to Warehouse Builder such as email, FTP commands, and operating system executables.

Using Activities in Process Flows

Use this section as a reference for all the activities. The section begins by categorizing activities by type. For detailed descriptions of each activity, see the alphabetical listing in the remainder of this section.

Oracle Warehouse Builder Specific Activities

For each of the Oracle Warehouse Builder specific activity as listed in Table 22–1, you can specify one or more incoming transitions. For outgoing transitions, you can use the success, warning, error, and unconditional transitions once each and then also define an unlimited number of complex condition transitions.

Table 22–1 Warehouse Builder Specific Activities

Icon	Activity	Brief Description
	Data Auditor	Adds to the process flow an existing data auditor monitor used in data profiling.
	Mapping	Adds an existing mapping to the process flow.
	Subprocess	Embeds an existing process flow within the process flow.
	Transform	Adds an existing transformation to the process flow.

Committing Data

When you add Warehouse Builder Specific activities to a process flow, the process flow evaluates each of these activities as separate transactions. For example, when you add mapping activities, the process flow commits and rolls back each mapping independently. In this design, it is not possible to control all the mappings by one commit or rollback statement.

To collectively commit or rollback multiple mappings, consider designing the process flow with a Sqlplus activity associated with a script that calls each mapping. For instructions, see "Committing Mappings Using the Process Flow Editor" on page 8-12.

Utility Activities

Table 22–2 lists each utility activity and shows the associated icon.

Table 22–2 Utility Activities

Icon	Activity	Brief Description
	Assign	Assigns a value to a variable.
	Email	Sends an email. For example, send an email message about the status of activities in the process flow.
	File Exists	Use the File Exists activity to check if a file is located on a specified drive or directory.
	FTP	Launches a file transfer protocol command during a process flow. For example, use the FTP activity to move data files to the machine where a mapping executes.
	Manual	Halts a process flow and requires manual intervention to resume the process flow.
	Notification	Can send an email to a user and allows the user to select from a list of responses that dictates how the process flow proceeds.
	Set Status	Interjects a success, warning, or error status.
	Sqlplus	Runs a SQL*Plus script in a process flow.
	User Defined	Represents an activity not defined by Warehouse Builder and enables you to incorporate it into a process flow.
	Wait	Delays the progress of the process flow by a specified amount of time.

Control Activities

Table 22–3 lists the activities you use to control the process flow. The table shows the associated icon. It also lists the number of incoming and outgoing transitions allowed for each activity.

Table 22–3 Control Activities

Icon	Activity	Brief Description	Incoming Transitions	Outgoing Transitions
	AND	Specifies the completion of all incoming activities before launching another activity.	2 or more allowed. The number of incoming transitions must be less than or equal to the number of outgoing transitions from the upstream FORK.	Unconditional and complex transitions are not allowed.
	End (successfully)	Designates a path as being successful.	1 or more allowed	not allowed

Table 22–3 (Cont.) Control Activities

Icon	Activity	Brief Description	Incoming Transitions	Outgoing Transitions
	End (with errors)	Designates a path as ending in errors.	1 or more allowed	not allowed
	End (with warnings)	Designates a path as ending with warnings.	1 or more allowed	not allowed
	End Loop	Defines the end of a For Loop or While Loop	1 or more allowed	1 to For Loop or While Loop only
	For Loop	Use this activity with an End Loop to define constructs that repeat.	1 from End Loop required plus more from other activities	1 Loop condition and 1 Exit required
	FORK	Launches two or more activities after completing an activity.	1 or more allowed	2 or more unconditional transitions only
	OR	Launches an activity after the completion of any of two or more specified activities.	2 or more allowed	1 unconditional transition only
	Route	Defines exclusive OR and if-then-else scenarios.		
	While Loop	Executes other activities while a condition is true.	1 from End Loop required plus more from other activities	1 Loop condition and 1 Exit required

AND

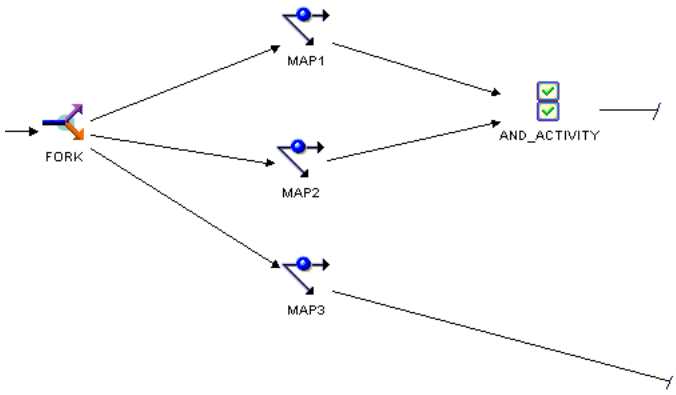
Use the AND activity to specify the completion of two or more activities before resuming the process flow.

The AND activity can have two or more incoming transitions. To correctly design process flows with an AND activity, you must place a FORK activity upstream of the AND. Also, the number of transitions going into the AND must be less than or equal to the number of outgoing transitions from the upstream FORK. The FORK is the only activity that enables you to assign multiple unconditional transitions and therefore ensure the completion of multiple activities as required by the AND activity.

The AND activity enables you to aggregate the outcome of the upstream activities. If all the upstream activities return SUCCESS then the AND activity returns SUCESSES. If any upstream activities return an ERROR, then the AND activity returns ERROR, else a WARNING is returned. Any activity that does not have an outcome is considered to have returned SUCCESS. Use the SET_STATUS to force an outcome. The feature is particularly useful to test if a set of mappings that are running in parallel have all successfully completed.

Figure 22–1 shows the AND and FORK activities in a process flow. In this example, AND_ACTIVITY triggers downstream activities based on the completion of MAP1 and MAP2. The process flow is valid because the FORK activity has 3 outgoing transitions while AND_ACTIVITY has 2 incoming transitions. The process flow would also be valid if the transition and activities associated with MAP3 were deleted.

Figure 22–1 AND Activity in a Process Flow



For outgoing conditions, the AND activity can have one, two, or three conditional transitions. This results in three possible paths terminating in success, warning, and error activities.

Assign

Assigns a value to a variable. For example, use this activity to initialize a variable back to zero.

Table 22–4 Assign Activity Parameters

Parameter	Description
Value	Type in the value to assign to the variable.
Variable	Select a variable that you previously defined in the editor.

Data Auditor

You can design process flows that proceed based on the results of profiling data. For example, you create logic that runs a mapping only if the quality of data meets a standard as determined by the threshold parameter.

Table 22–5 Data Auditor Monitor Activity Parameters

Parameter	Description
AUDIT_LEVEL	NONE STATISTICS ERROR_DETAILS COMPLETE
BLUK_SIZE	1+
COMMIT_ FREQUENCY	1+
MAX_NO_OF_ ERRORS	Maximum number of errors allowed after which the mapping terminates

Table 22–5 (Cont.) Data Auditor Monitor Activity Parameters

Parameter	Description
OPERATING_MODE	SET_BASED
	ROW_BASED
	ROW_BASED_TARGET_ONLY
	SET_BASED_FAIL_OVER_TO_ROW_BASED
	SET_BASED_FAIL_OVER_TO_ROW_BASED_TARGET_ONLY

Email

You can instruct Warehouse Builder to send email notifications after the completion of an activity in a process flow. You may find this useful, for example, for notifying administrators when activities such as mappings end in error or warnings.

Table 22–6 lists the parameters you set for the email activity.

Table 22–6 Email Activity Parameters

Parameter	Description
SMTP Server	The name of outgoing mail server. The default value is `localhost`.
Port	The port number for the outgoing mail server. The default value is `25`.
From_Address	The email address from which process flow notifications are sent.
Reply_To_ Address	The email address or mailing list to which recipients should respond.
To_Address	The email address(es) or mailing list(s) that receive the process flow notification. Use a comma or a semi-colon to separate multiple email addresses.
CC_Address	The email address(es) or mailing list(s) that receive a copy of the process flow notification. Use a comma or a semi-colon to separate multiple email addresses
BCC_Address	The email address(es) or mailing list(s) that receive a blind copy of the process flow notification. Use a comma or a semi-colon to separate multiple email addresses
Importance	The level of importance for the notification. You can type in any text such as, for example, `High`, `Medium`, or `Low`.
Subject	The text that appears in the email subject line.
Message_Body	The text that appears in the body of the email. To type in or paste text, select **Value** at the bottom of the Activity panel. The Process Flow Editor does not limit you on the amount of text you can enter.

For email addresses, you can enter the email address with or without the display name. For example, the following entries are correct:

```
jack.emp@oracle.com
```

```
Jack Emp<jack.emp@oracle.com>
```

```
Jack Emp[jack.emp@oracle.com]
```

```
Jack Emp[jack.emp@oracle.com],Jill Emp[jill.emp@oracle.com]
```

```
Jack Emp[jack.emp@oracle.com];Jill Emp[jill.emp@oracle.com]
```

End

Every path in the process flow must terminate in an End activity.

When you first create a process flow, Warehouse Builder includes a success type End activity by default. Use end types to indicate the type of logic contained in a path. Since a given activity such as a mapping has three possible outcomes, the editor includes three ending types, as shown in Table 22–7. You can use these ending types to design error handling logic for the process flow.

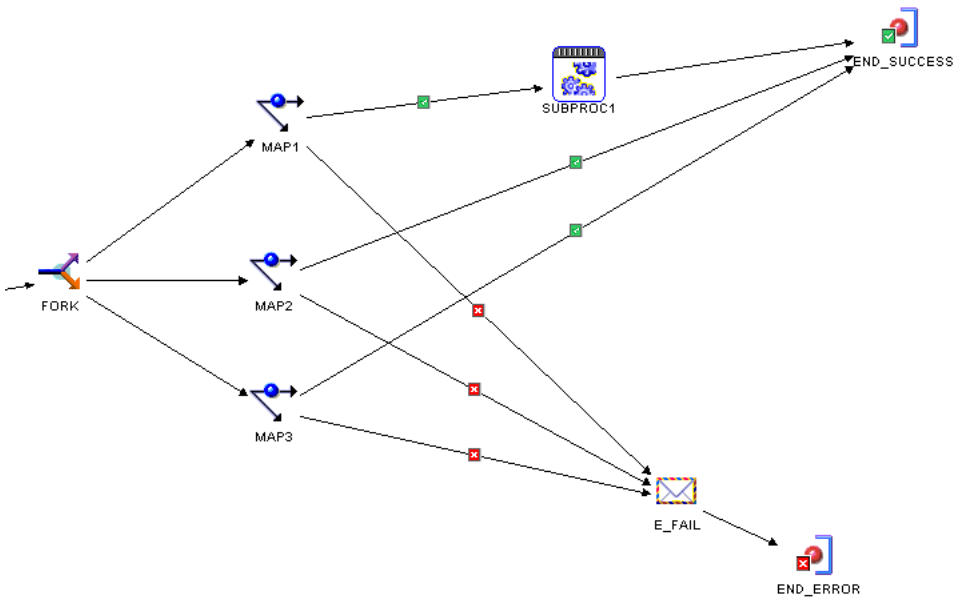
Table 22–7 Types of End Activities

Icon	End Type	Description
	Success	Indicates that the path or paths contain logic dependent upon the successful completion of an upstream activity.
	Warning	Indicates that the path or paths contain logic dependent upon an upstream activity completing with warnings.
	Error	Indicates that the path or paths contain logic dependent upon an upstream activity completing with errors.

You can design a process flow to include one, two, or all three types of endings. You can use each ending type only once; duplicate ending types are not allowed. Each End activity can have a single or multiple incoming transitions.

In Figure 22–2, END_SUCCESS has three incoming transitions, each dependent upon the successful completion of upstream activities. END_ERROR has one incoming transition from an email activity that runs when any of the upstream mapping activities completes with errors.

Figure 22–2 End Activities in a Process Flow



By default, every process flow includes an END_SUCCESS. Although you cannot change an end activity to another type, you can add others of different types.

To add end activities to a process flow:

1. From the palette on the Process Flow Editor, drag and drop the desired End icon onto the canvas.
2. Warehouse Builder does not allow you to select ending types already present in the process flow.
3. Select **OK.**

   Warehouse Builder adds the End activity or activities to the canvas.

End Loop

The editor adds an End Loop for each for loop and while loop you add to the canvas.

The End Loop activity must have a single unconditional outgoing transition to its For Loop or While Loop activity. All the flows that for part of the loop must converge on the End Loop activity to ensure that no parallel flows remain for either the next loop interaction or the exit of the loop.

File Exists

Use the File Exists activity to verify the existence of a file before executing the next activity. In the Activities panel, type the name of the file.

The File Exists activity checks to see if the file exists only once. If the file exists, the process flow proceeds with the success transition. If the file does not exist, the process flow precedes with the warning transition. The File Exists activity triggers the error transition only in the case of a catastrophic failure such as TCL error when using OMB Plus.

The File Exists activity has one parameter called PATH. Specify either a fully qualified filename, directory name, or a semi-colon separated list for this parameter. The paths are normally tested in the same host that is running the Control Center service.

The security constraints of the underlying operating system may disallow access to one or more files, giving the impression that they do not exist. If all the paths exist, then the activity return EXISTS. If none of the paths exist, then it returns MISSING. If some paths exist, it returns SOME_EXIST.

FORK

Use the FORK activity to launch multiple, concurrent activities after the completion of an activity.

You can assign multiple incoming transitions to a FORK. The FORK is the only activity that enables you to assign multiple unconditional outgoing transitions for parallel execution. For example, in Figure 22–3, the process flow executes the activities named FTP, FDS, and EMAIL in parallel after completing MAP1.

Figure 22–3 FORK Activity Ensures Parallel Execution

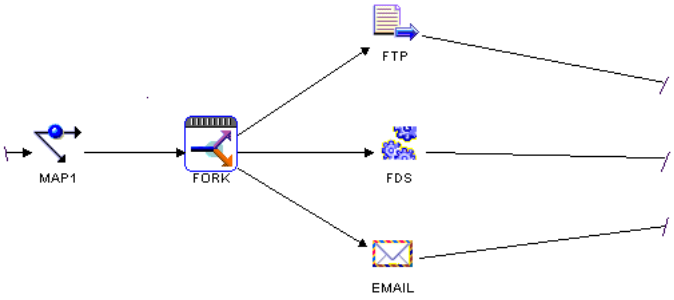
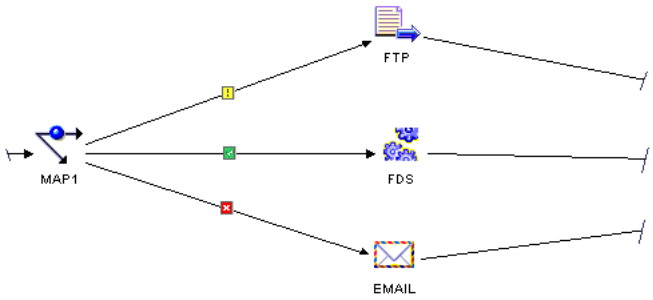


Figure 22–4 shows the same activities without the FORK activity. In this case, only one of the activities runs based on the completion state of MAP1.

Figure 22–4 Absence of FORK Activity Results in Conditional Execution



The Process Flow Editor does not limit you on the number of outgoing transitions or concurrent activities you can assign from a FORK. When designing for concurrent execution, design the FORK based on limitations imposed by the workflow engine or server you use to run the process flow.

The outgoing FORK activity transition cannot have complex expressions

For Loop

Use the For Loop to repeatedly execute activities you include in the loop and then exit and resume the process flow.

When you add a For Loop, the editor also adds an End Loop activity and a transition to the End Loop. For outgoing transitions, define one with a loop condition and one with an exit condition. Select an outgoing transition and click on Condition in the object details.

Table 22–8 For Loop Activity Parameters

Parameter	Description
Condition	An expression which when evaluated to true executes the loop transition other the exit transition.
Variable	Bound to a variable or parameter, its value is incremented every iteration.

Table 22–8 (Cont.) For Loop Activity Parameters

Parameter	Description
Initial_Value	The initial value of the variable upon entering the loop. By default, you must enter an expression.
Next_Value	.The next value of the variable. By default you must enter an expression

FTP

Use the FTP activity to transfer files from one file location to another based on a script of FTP commands that you provide. The FTP activity is a specialization of the User Defined activity. The difference between these two is that the FTP activity should be configured with the remote file location.

For the process flow to be valid, the FTP commands must involve transferring data either from or to the server with the Control Center Service installed. To move data between two machines, neither of which host the Control Center Service, first transfer the data to the Control Center Service host machine and then transfer the data to the second machine.

Before you design a process flow with an FTP activity, ensure that the sources and destinations have defined locations.

The FTP activity relies on a script of FTP commands that you provide. You have a choice of either writing that script within Warehouse Builder or directing Warehouse Builder to a file containing the script. Choose one of the following methods:

- Writing a Script Within Warehouse Builder
- Calling a Script Outside of Warehouse Builder

Writing a Script Within Warehouse Builder

Choose this method when you want to maintain the script of FTP commands in Warehouse Builder and/or when password security to servers is a requirement.

For this method, in the COMMAND parameter, enter the path to the FTP executable. Also, for file transfer protocols other than UNIX, type additional parameters for the protocol in the PARAMETER_LIST parameter. Enter a script into the VALUE column of the SCRIPT parameter.

Table 22–9 lists the parameters you set for the FTP activity when writing the script within Warehouse Builder.

Table 22–9 FTP Activity Parameters for Script in Warehouse Builder

Parameter	Description
COMMAND	Enter the path to file transfer protocol command such as `c:\WINNT\System32\ftp.exe` for Windows operating systems.

Table 22–9 (Cont.) FTP Activity Parameters for Script in Warehouse Builder

Parameter	Description
PARAMETER_ LIST	This is a list of parameter that will be passed to the command. Parameters are separated from one another by a token. The token is taken as the first character on the parameter list string and the string must also end in that token. Warehouse Builder recommends the '?' character, but any character can be used. For example, to pass 'abc', 'def', and 'ghi' you can use the following equivalent: ?abc?def?ghi? or !abc!def!ghi! or |abc|def|ghi| If the token character or '\' needs to be included as part of the parameter then it must be escaped with '\', for example '\\'. If '\' is the token character, then '/' becomes the escape character. Enter any additional parameters necessary for the file transfer protocol. For Windows, enter *?"-s:${Task.Input}"?* The ${Task.Input} token prompts Warehouse Builder to store the script in temporary file and replaces the token with the name of the temporary file. The script is therefore not past on as standard input. Note: The -s parameter is set for the Windows FTP command because it cannot driven with standard input except from a file. For UNIX, you should leave this value blank. In general, UNIX FTPs read from standard input and therefore do not require any other parameters.
SUCCESS_ THERSHOLD	Designates the FTP command completion status.Type in the highest return value from the operating system that indicates a successful completion. When the operating system returns a higher value, it indicates that the command failed. The default value is `0`.
SCRIPT	You can type the required script for FTP in this parameter. To type or paste text, select **Value** at the bottom of the Object Details panel. The Process Flow Editor does not limit you on the amount of text you can enter. Each carriage return in the script is equivalent to pressing the *`Enter`* key. The script should end with bye or quit followed by a carriage return to ensure that the FTP command is terminated.

Figure 22–5 shows an example of the FTP activity parameter settings for calling a script written within Warehouse Builder.

Figure 22–5 Activity View for FTP Activity Using a Script

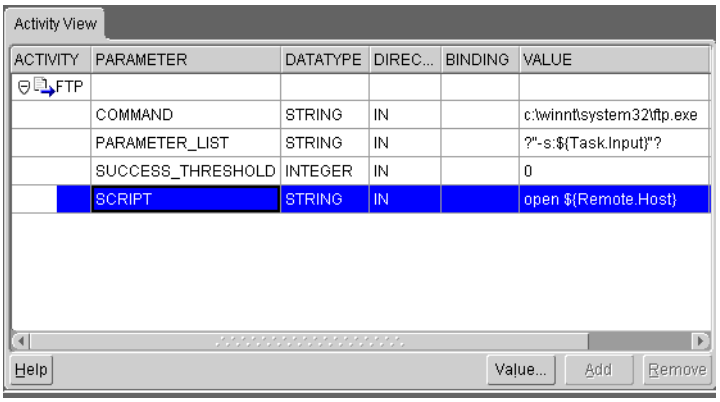
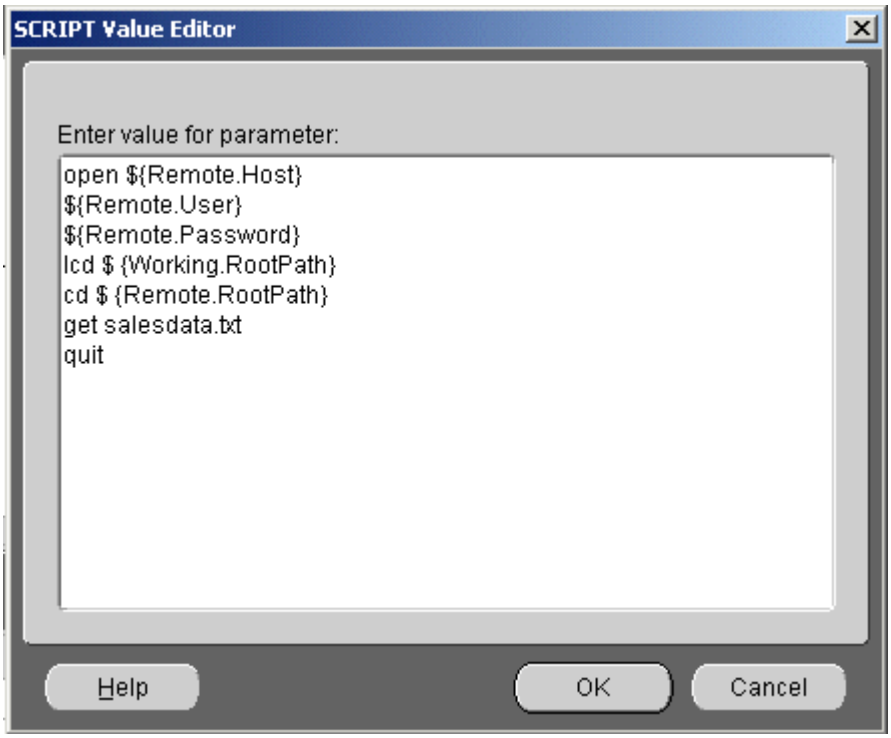


Figure 22–6 shows the first line of the script in the VALUE column of the SCRIPT parameter. To view the entire script, click the **:** button. Warehouse Builder displays the SCRIPT Value editor that you can use to write or copy and paste a FTP script such as the script displayed in Figure 22–6.

Figure 22–6 SCRIPT Value Editor Using Substitution Variables



Notice that the script inFigure 22–6 includes ${Remote.User} and ${Remote.Password}. These are substitution variables. Refer to "Using Substitution Variables" for more details.

Using Substitution Variables

Substitution variables are available only when you choose to write and store the FTP script in Warehouse Builder.

Use substitution variables to prevent having to update FTP activities when server files, accounts, and passwords change. For example, consider that you create 10 process flows that utilize FTP activities to access a file on *salessrv1* under a specific directory. If the file is moved, without the use of substitution variables, you must update each FTP activity individually. With the use of substitution variables, you need only update the location information.

Substitution variables are also important for maintaining password security. When Warehouse Builder executes an FTP activity with substitution variables for the server passwords, it resolves the variable to the secure password you entered for the associated location.

Table 22–10 lists the substitute variables you can enter for the FTP activity. *Working* refers to the machine hosting the Control Center Service, the *local* machine in this case study. Remote refers to the other server involved in the data transfer. You designate which server is remote and local when you configure the FTP activity as described in "Configuring Process Flows" on page 24-13.

Table 22–10 Substitute Variables for the FTP Activity

Variable	Value
${Working.RootPath}	The root path value for the location of the Control Center Service host.
${Remote.Host}	The host value for the location involved in transferring data to or from the Control Center Service host.
${Remote.User}	The user value for the location involved in transferring data to or from the Control Center Service host.
${Remote.Password}	The password value for the location involved in transferring data to or from the Control Center Service host.
${Remote.RootPath}	The root path value for the location involved in transferring data to or from the Control Center Service host.
${Task.Input}	The Working and Remote location are set for the FTP activity when configuring a Process Flow.
${parameter_name}	The values of custom parameters can be substituted into the script and parameter using ${parameter_name} syntax.

All custom parameters are imported into the command's environment space. For example, by defining a custom parameter called PATH it is possible to change the search path used to locate Operating System executables (some JAVA VMs may prevent this).

Calling a Script Outside of Warehouse Builder

If password security is not an issue, you can direct Warehouse Builder to a file containing a script including the FTP commands and the user name and password.

To call a file on the file system, enter the appropriate command in PARAMETERS_LIST to direct Warehouse Builder to the file. For a Windows operating system, enter the following: *?"-s:<file path\file name>"?*

For example, to call a file named *move.ftp* located in a temp directory on the C drive, enter the following: *?"-s:c:\temp\move.ftp"?*

Leave the SCRIPT parameter blank for this method.

Table 22–11 lists the parameters you set for the FTP activity when the FTP script resides in a file on your system.

Table 22–11 FTP Activity Parameters for Script Outside of Warehouse Builder

Parameter	Description
Command	Leave this parameter blank.

Parameter List	Enter the path and name of the file for the FTP script.The Process Flow Editor interprets the first character you type to be the separator. For example, the Process Flow Editor interprets the following entry as two parameters, `/c` and `dir`
	`?/c?dir?`
	Use the backslash as the escape character. For example, the Process Flow Editor interprets the following entry as three parameters, `-l` and `-s` and `/`
	`/-l/-s/\//`

Success Threshold	Designates the FTP command completion status.Type in the highest return value from the operating system that indicates a successful completion. When the operating system returns a higher value, it indicates that the command failed. The default value is `0`.
Script	Leave this parameter blank.

Manual

Use this activity to halt a process flow.

Once the process flow halts, a user must intervene using the Control Center or Repository Browser to resume the process flow.

Consider using this activity to enable you to design a process to restart or recover ETL processes.

The manual activity is similar to the Notification activity except that does not require you to implement Oracle Workflow and therefore does not send an email. To achieve the same results as the notification activity without interacting with Oracle Workflow, consider using the Email activity followed by a manual activity.

Table 22–12 Manual Activity Parameters

Parameter	Description
Performer	The name of the person or group that can resume the process flow.
Subject	Type in the subject of the activity.
Text_body	Type in special instructions to be performed before resuming the process flow.
Priority	Choose a priority. The options are: 1= high, 50=medium, and 99=low.

Mapping

Use the mapping activity to add an existing mapping that you defined and configured in the Mapping Editor.

You can assign multiple incoming transitions to a mapping activity. For outgoing transitions, assign 1 unconditional transition or up to one of each of the unconditional transitions.

When you add a mapping to a process flow, you can view its configuration properties in the Activities panel. The mapping activity in the Process Flow Editor inherits its

properties from the mapping in the Mapping Editor. In the Process Flow Editor, you cannot change a property data type or direction.

You can, however, assign new values that affect the process flow only and do not change the settings for the mapping in the Mapping Editor. For example, if you change the operating mode from set-based to row-based in the Process Flow Editor, the process flow executes in row-based mode. The original mapping retains set-based mode as its operating mode. If want to change the properties for the underlying mapping, see "Configuring Mappings Reference" on page 24-1.

If a mapping contains a Mapping Input Parameter operator, specify a value according to its data type. The Process Flow Editor expects to receive a PL/SQL expression when you add a Mapping Input Parameter operator to a mapping. If the Mapping Input Parameter is a string, enclose the string in double quotes.

If you want to update a process flow with changes you made to a mapping in the Mapping Editor, delete the mapping activity from the process flow and add the mapping activity again.

Table 22–13 and Table 22–14 list the different mapping parameters in PL/SQL and SQL*Loader.

Table 22–13 lists the PL/SQL mapping parameters.

Table 22–13 Mapping parameters for PL/SQL

Parameter	Valid Values
AUDIT_LEVEL	NONE
	STATISTICS
	ERROR_DETAILS
	COMPLETE
BLUK_SIZE	1+
COMMIT_FREQUENCY	1+
MAX_NO_OF_ERRORS	Maximum number of errors allowed after which the mappings will terminate with an error
OPERATING_MODE	SET_BASED
	ROW_BASED
	ROW_BASED_TARGET_ONLY
	SET_BASED_FAIL_OVER_TO_ROW_BASED
	SET_BASED_FAIL_OVER_TO_ROW_BASED_TARGET_ONLY

Table 22–14 lists the SQL*Loader mapping parameters.

Table 22–14 Mapping parameters for SQL*Loader

Parameter	Description
BAD_FILE_NAME	The name of the SQL*Loader "BAD" file
DATA_FILE_NAME	The name of the SQL*Loader "DATA" file
DISCARD_FILE_NAME	The name of the SQL*Loader "DISCARD"file

Notification

This activity enables you to design a process to restart or recover ETL processes.

This activity works in conjunction with Oracle Workflow. To implement notifications in Warehouse Builder you must also implement Workflow notifications in Oracle Workflow. Alternatively, you could use an Email followed by a Manual activity. Oracle Workflow subsystem decides how the message is sent.

To use the notification activity, first define the parameters listed in Table 22–15. Define a conditional outgoing transition based on each response you define. For example, if the value of response_type is *yes,no* and default_response is *yes*, define two outgoing transitions. Right-click each transition and select Condition to view a list of conditions. In this example, you create one outgoing transition with condition set to *yes* and another set to *no*.

Table 22–15 Parameters for the Notification Activity

Parameter	Description
Performer	Type the name of a role defined by the Oracle Workflow administrator.
Subject	Type in the subject of the email.
Text_body	Type in instructions for the performer. Explain how their response affects the process flow and perhaps explain the default action if they do not respond.
Html_body	Use html in addition to or instead of text. Content you enter in html_body is appended to text_body.
Response_type	Type a comma separated list of values from which the performer selects a response. Each entry corresponds to one outgoing transition from the activity.
Default_response	Type the default response.
Priority	Select a priority for the email of either 1 (high), 50 (medium), or 99 (low).
Timeout	The number of seconds to wait for response. If this is set, a #TIMEOUT transition is required.
Response_processor	Oracle Workflow notification response processor function. For more information, see the Oracle Workflow documentation.
Expand_roles	Used for notification voting. Set this value to TRUE or FALSE. When set to TRUE, a notification is sent to each member of a group rather then a single shared message to the group. For more information, see the Oracle Workflow documentation.

Note: Due to Oracle Workflow restriction, only the performer, priority, timeout, and customer parameter values can be changed at runtime.

Notification Message Substitution

Custom parameters can be added to the notification activity to pass and retrieve data from the user through the notification. IN parameters can be substituted into the message using SQL and appropriate syntax. For example, for a custom parameter called NAME, the text `&NAME` will be replaced with the parameter's value. You will also be prompted to enter values for the OUT Parameters.

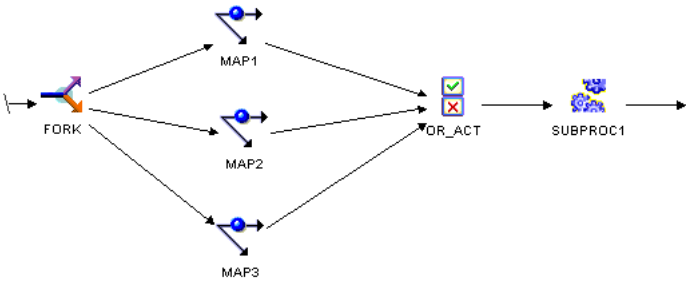
OR

Use the OR activity to launch an activity based on the completion of one of a multiple number of upstream activities. You can assign multiple incoming transitions and only one unconditional outgoing transition to an OR activity.

The OR activity has similar semantics to the AND activity except that the OR activity propagates the SUCCESS, WARNING, or ERROR outcome of the first upstream activity that is completed.

An OR activity in a process flow ensures that downstream activities are triggered only once for each run of a process flow. For example, in Figure 22–7, the SUBPROC1 launches once any one of the three upstream Mapping activities completes.

Figure 22–7 The OR activity in an process flow



The Process Flow Editor enables you to omit the OR activity and assign transitions from each of the three Mapping activities to SUBPROC1 shown in Figure 22–7. However, this logic would launch SUBPROC1 three times within the same run of a process flow. Avoid this by using an OR activity.

Route

Use this activity to route the outcome of an activity to specific results based on a condition you define. This enables you to define exclusive OR and if-the-else scenarios.

A Route activity has no operation and therefore can be used to place a bend in a transition. Like any other activity, you can add outgoing complex condition transitions to the Route activity. But since the activity has no operation, the condition may only refer to the process flow's parameters and variables.

The inclusion of a Route activity can effect the outcome of an AND or OR activity. Since the Route activity has no outcome of its own, it will be considered to have completed as SUCCESS.

This activity does not have any parameters.

Set Status

Use this activity to interject a success, warning, or error status.

You can use set status as a means of overriding the behavior of the AND activity. Recall that if any of the activities immediately preceding an AND return an error, the AND activity resolves to an error. If you want the AND to resolve to success regardless of the result of a preceding activity, insert between that activity and the AND activity a set status.

Sqlplus

Use a sqlplus activity to introduce a script into the process flow.

To paste or type in a script, select the activity on the canvas, select Script in the editor explorer, and then click on Value in the object details. Or, to point to an existing script on a file system, go to parameter_list and type the at sign, @, followed by the full path.

Although you can use this activity to accomplish a broad range of goals, one example is to use a sqlplus activity to control how multiple mappings are committed in a process flow as described in "Committing Mappings Using the Process Flow Editor" on page 8-12.

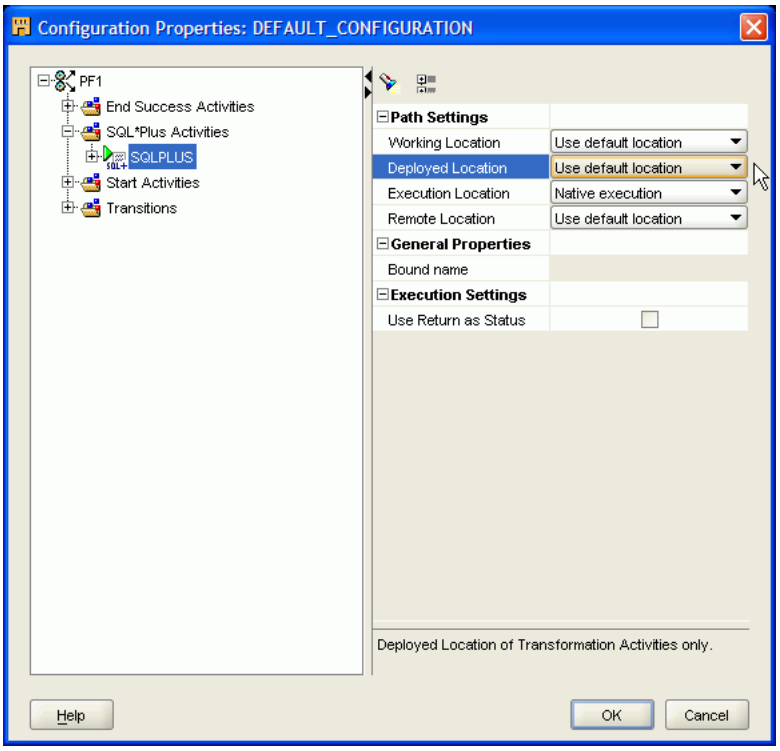
Using Activities in Process Flows

The Process Flow in SQLPlus activity is executed by the configuration item in the Deployed Location.

To set the location that will execute the SQLPlus activity:

1. In Project Explorer, expand the Process Flow module.
2. In the Process Flow module, select **Process Flow**.
3. Right-click Process Flow and select **Configure**.
4. In Configuration Properties window, expand SQL*Plus activity.
5. Select **SQLPLUS**.

Figure 22–8 Configuring the Deployed Location



6. Under Path Settings, set the **Deployed Location** option to the location that will execute the SQLPLUS Activity.

The SQLPlus activity is similar to the User Defined activity with the following differences:

- The COMMAND parameter cannot be specified as it is automatically derived.
- If the ${Task.Input} substitution variable is used then the temporary file that is created will end in .sql.
- It has a different set of substitution variables. The activity should be configured with a Deployed database location.

Table 22–16 SqlPlus Activity Parameters

Parameter	Description
Parameter_List	Type @ followed by the full path of the location of the file containing the script.
Script	As a alternative to typing the path in parameter_list, type or paste in a script.

Using Substitution Variables

The substitution variables are similar to FTP. It uses the following location instead of the remote location as it is connecting to an Oracle Database and not a FTP server:

- Working location as the local location
- Deployed location as the target location

Table 22–17 SqlPlus Substitution Variables

Substitution Variable	Description
${Working.RootPath}	The local working directory
${Task.Input}	A temporary file create from the SCRIPT parameter
${Target.Host}	The target location's host name
${Target.Port}	The target location's post number
${Target.Service}	The target location's service name
${Target.TNS}	The target location's TNS address
${Target.Schema}	The target location's schema name
${Target.User}	The target location's user name
${Target.Password}	The target location's user password
${Target.URL}	The target location's connection descriptor

If the PARAMTER_LIST is empty then one of the following parameter list is used depending on the Deployed location parameters:

- ?${Target.User}/${Target.Password}@${Target.TNS}?@${Task.Input}?
- ?${Target.User}/${Target.Password}@${Target. URL}?@${Task.Input}?
- ?${Target. Schema}/${Target.Password}@${Target.TNS}?@${Task.Input}?
- ?${Target. Schema}/${Target.Password}@${Target. URL}?@${Task.Input}?

SQL *Plus Command

The SQL*Plus command cannot be entered directly to the FTP User Defined activities. It is either loaded from the Warehouse Builder home or its location is predefined by the repository administrator.

The Sql*Plus execution location is determined from the following platform properties in the following order:

1. property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_10g
2. property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_9i
3. property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_8i
4. property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_default

The Oracle home is determined in a similar way from the following platform properties:

1. property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_10g
2. property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_9i
3. property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_8i
4. property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_default

Start

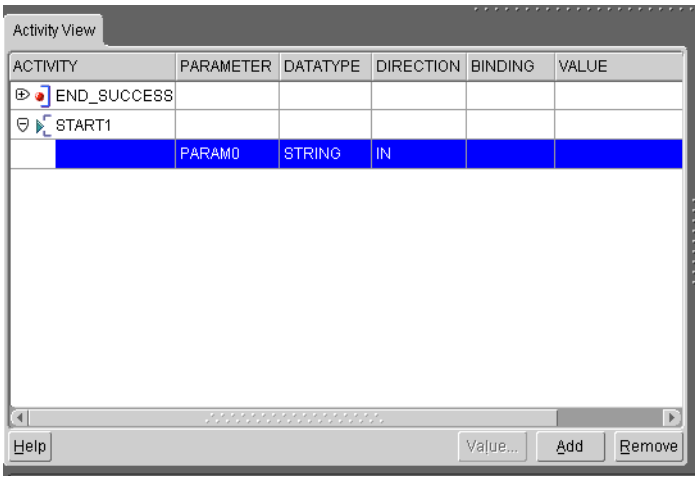
By default, each process flow includes one start activity. You can set input parameters for the start activity that become the input parameters for the complete process flow.

To add parameters to a Start activity:

1. In Project Explorer, double-click the Process Flow to open the Process Editor.
2. In Explorer pane, click **Selected Objects** tab.
3. In Selected Objects tab, expand the **Activities** folder.
4. Select Start activity and click create button (the tiny green "Plus" button at the left top corner) on the Explorer pane.

   The Process Flow Editor adds a parameter under the Start activity.

Figure 22–9 Parameter added to a Start activity



5. In Object Details pane, set the properties for the parameter.

   Change the parameter name and data type as necessary. You cannot alter its direction and binding. The direction is IN, indicating that the parameter is an input parameter only. For value, type the parameter value. You can overwrite this value at runtime.

6. You can now use the parameter as input to other activities in the process flow.

Subprocess

Use a subprocess activity to launch a previously created process flow. From one process flow, you can launch any other process flow that is contained within the same process flow package.

Once you add a subprocess to a process flow, use it in your design similar to any other activity. You can assign multiple incoming transitions. For outgoing transitions, assign either 1 unconditional outgoing transition or up to 3 outgoing conditional transitions.

The END activities within the subprocess apply to the Subprocess only and do not function as a termination point in the process flow.

An important difference between a subprocess and other activities is that you can view the contents of a subprocess, but you cannot edit its contents in the parent process flow. To edit a subprocess, open its underlying process flow from the Project Explorer. With the exception of renaming a process flow, the Process Flow Editor propagates changes from child process flows to its parent process flows.

> **Note:** Use caution when renaming process flows. If you rename a process flow referenced by another process flow, the parent process flow becomes invalid. You must delete the invalid subprocess and add a new subprocess associated with the new name for the child process flow.

To add subprocess activities to a process flow:

1. From the palette in the Process Flow Editor, drag and drop the Subprocess activity icon onto the canvas.

   Warehouse Builder displays a dialog to select and add a process flow as a subprocess.

2. Expand the process flow module and select a process flow from the same process flow package as the parent process flow.

   Warehouse Builder displays the process flow as a subprocess activity on the parent process flow.

3. To view the contents of the subprocess, right-click the subprocess and select **Expand Node.**

   The Process Flow Editor displays the graph for the subprocess surrounded by a blue border.

Transform

When a function transform is dropped onto the canvas the return parameter is created as a new parameter with the same name as the transform. When you add transformations from the Warehouse Builder transformation library to a process flow

using the Transform activity, the Process Flow Editor displays the parameters for the transformation in the Activity panel.

You can specify one or more incoming transitions to launch a transform activity. For outgoing transitions, you can either specify one unconditional transition or one of each of the three conditional transitions.

If you specify conditional outgoing transitions, you can configure the activity to base its status on its return value. For more information on **Use Return as Status,** see "Configuring Process Flows" on page 24-13.

If you want to update a process flow with changes you made to a transformation, delete the transform activity from the process flow and add the transform activity again.

For transforms that are not deployed, such as the public transformations, the activity must be configured with a Deployed location value.

User Defined

The user defined activity enables you to incorporate into a process flow an activity not defined within Warehouse Builder.

You can specify one or more incoming transitions to launch a user defined process activity. For outgoing transitions, you can either specify one unconditional transition or one of each of the three conditional transitions.

If you specify conditional outgoing transitions, you can configure the activity to base its status on it return value. For more information on **Use Return as Status,** see "Configuring Process Flows" on page 24-13.

Table 22–18 lists the parameters you set for the FTP activity.

Table 22–18 User Defined Process Activity Parameters

Parameter	Description
Command	The command to execute the user defined process you defined. Type the path and file name such as `c:\winnt\system32\cmd.exe .`

Parameter List	The list of parameters to be passed to the user defined process. Type the path and file name such as `?/c?c:\\temp\\run.bat`.
	The Process Flow Editor interprets the first character you type to be the separator. For example, the Process Flow Editor interprets the following entry as `/c and dir`.
	`?/c?dir?`
	Use the backslash as the escape character. For example, the Process Flow Editor interprets the following entry as `-l` and `-s and /`.
	`/-l/-s/\//`
	You can also enter the substitution variables listed in Table 22–19.

Success Threshold	Designates the completion status.Type in the highest return value from the operating system that indicates a successful completion. When the operating system returns a higher value, it indicates that the command failed. The default value is `0`.

Table 22–18 (Cont.) User Defined Process Activity Parameters

Parameter	Description
Script	You can enter a script here or type in a filename for a script. If you type in a filename, use the ${Task.Input} variable in the parameter list to pass the filename. To type in or paste text, select **Value** at the bottom of the Activity panel. The Process Flow Editor does not limit you on the amount of text you can enter. Each carriage return in the script is equivalent to pressing the `Enter` key. Therefore, end the script with a carriage return to ensure that the last line is sent.

Table 22–19 lists the substitute variables you can enter for the FTP activity.

Table 22–19 Substitute Variables for the User Defined Process Activity

Variable	Value
${Working.Host}	The host value for the location of the Control Center Service host.
${Working.User}	The user value for the location of the Control Center Service host.
${Working.Password}	The password value for the location of the Control Center Service host.
${Working.RootPath}	The local working directory.
${Task.Input}	A temporary file created from the SCRIPT parameter. Enter the Task.Input variable to direct Warehouse Builder to the script you write in the SCRIPT parameter. For Windows, enter into Parameter_List *?"-s:${Task.Input}"?* and for UNIX, enter into Parameter_List *?"${Task.Input}"?* where the question mark as the separator.

Wait

Use the wait activity to interject a delay in the process flow.

Table 22–20 Wait Activity Parameters

Parameter	Description
Minimum_ Delay	Type in the minimum time to wait. Specify the time in units of seconds.
Until_Date	Specify the date to wait until in the default format for your local region.

While Loop

Use While Loop to execute one or more activities only when a condition you define evaluates to being true.

Typically, you associate a While Loop with Assign activities that enable you to define the while condition. At least one assign activity initializes the data and at least one assign activity increments or modifies the data again to the end of a loop iteration.

When you add a While Loop, the editor also adds an End Loop activity and a transition to the End Loop. Create transitions from the While Loop activity to each activity you want to include in the loop. For each outgoing transition you add, apply

either an EXIT or LOOP condition to the transition by selecting the transition and clicking on Condition in the object details.

To define the while condition that governs whether or not to run the loop, select the While Loop activity, select Condition in the editor Explorer, and then define the condition in the object details.

Table 22–21 While Loop Activity Parameters

Parameter	Description
Condition	Define with a LOOP or EXIT condition.

23

Moving Large Volumes of Data

For optimum performance and manageability, Transportable Modules help design the preprocessing, and ensure fast replication and loading of enterprise data. This chapter contains the following topics:

About Transportable Modules

Oracle Warehouse Builder enables you to build and publish enterprise data warehouses in stages. Similar to the partitioning strategy, creating a data warehouse in stages by following a compartmentalization strategy potentially maximizes both performance and manageability.

Due to the processing overhead and network delays, Warehouse Builder mappings that access remote data through database links can run several magnitudes more slowly than if data were accessed locally. In this case, consider the following two options for speeding up the process:

- Create a Warehouse Builder Transportable Module to copy remote objects (tables, views, materialized views, and so on) from a source database into a target database at very fast speed. The mappings in the target data warehouse can then access data locally.
- Data can be partially processed in source database (to benefit from the local access performance), and then the preprocessed data can be copied, using a Transportable Module, from source to target database for final loading into the data warehouse.

For both strategies, a Transportable Module functions like a shipping service that moves a package of objects from one site to another at the fastest possible speed.

> **Note:** The use of transportable modules requires the Warehouse Builder Enterprise ETL Option.

The fundamental functionality of a Transportable Module is to copy a group of related database objects from one database to another using the fastest possible mechanisms.

Using the Warehouse Builder designer, a Warehouse Builder you first create a Transportable Module, and specify the source database location and the target database location. Then you select the database objects to be included in the

Transportable Module. The metadata of the selected objects are imported from the source database into the Transportable Module and stored in the Warehouse Builder design-time repository. To actually move the data and metadata from source into target, you need to configure the Transportable Module, and deploy it to the target location. At the deployment time, both data and metadata are extracted from the source database and created in the target database.

In the current release, the implementation of the data and metadata movement is accomplished through a combination of Oracle Data Pump, Transportable Tablespace, DBMS_FILE_TRANSFER, binary FTP, local file copy, and Warehouse Builder code generation and deployment. Warehouse Builder users can configure Transportable Modules to influence what implementation technologies will be invoked. As the server technology progresses, faster implementation methods will be added in future releases.

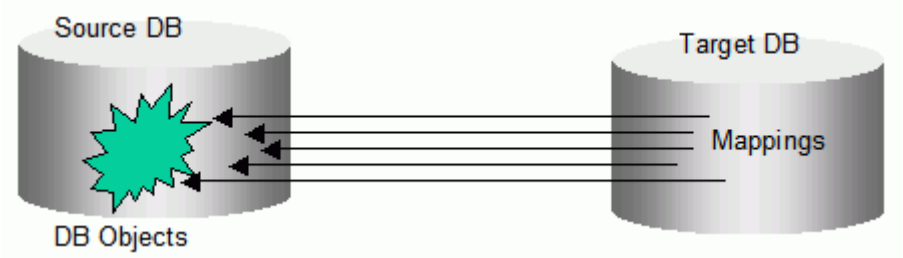
In the current release the following source objects can be added into Transportable Modules:

- Tablespaces
- Schemas
- Tables
- Views
- Sequences
- Materialized Views
- External Tables (including their data)
- PL/SQL Functions
- PL/SQL Procedures
- PL/SQL Packages
- Object Types
- Varrying Array Types
- Nested Table Types

More object types will be available as the technology advances in the future.

The Transportable Modules technology opens a new possibility for building an enterprise wide data warehouse in stages, which offers huge potentials for high performance and manageability. As shown in Figure 23–1, the traditional ETL process extracts data from remote databases through multiple remote accesses using database links.

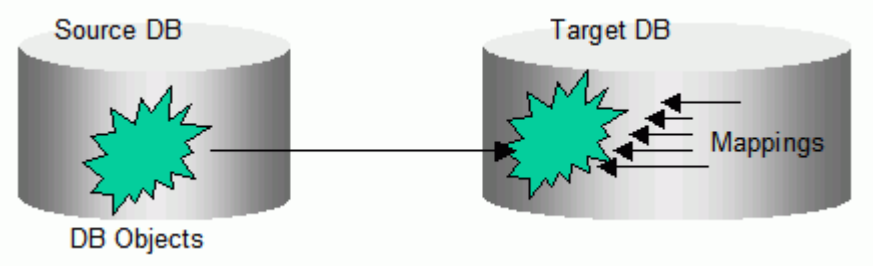
Figure 23–1 Extraction of data from remote databases through multiple remote accesses using database links



Remote accesses using database links suffer significant performance degradation due to serial queries and serial DMLs, plus the network latencies. The performance degradation will appear more wasteful if same source tables are accessed multiple times.

The Transportable Modules technology works as shown in Figure 23–2. In this new architecture, all the source objects needed by the mappings are bundled together and moved to target through just one deployment. The *OWB TM Deployment* uses fast Oracle Data Pump, FTP, and Oracle TTS to achieve very high transportation performance. And this transportation absorbs the cost of the network delays just once. After that, mappings access data locally, which can easily benefit from parallel queries and parallel DMLs. And repeated accesses to the same data increase the performance benefit of Transportable Modules.

Figure 23–2 Warehouse Builder Transportable Modules Deployment



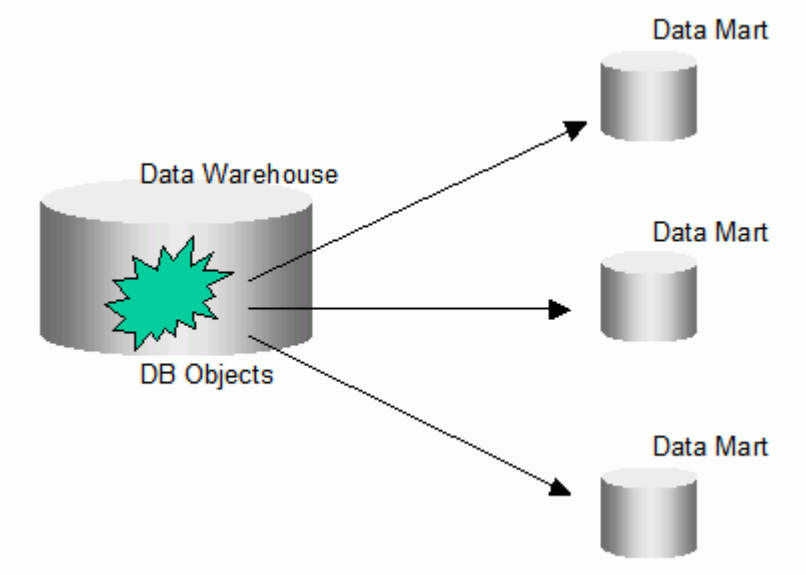
Using the Transportable Modules, data warehouse loadings become more manageable because the source database only needs to be shut down for a short period of time in order for Warehouse Builder Transportable Module to complete the deployment. Users of the source database do not have to wait until the entire data is loaded into the data warehouse. For example, if you are using the transportable tablespace implementation, Warehouse Builder Transportable Modules can copy a tablespace of 20 GB in about five minutes, resulting in a down time of five minutes in the source database.

Data copied into the target database is a snapshot of what was in the source database. This fact can be utilized to create a data versioning mechanism. More advanced Oracle users may create streams on the transported tables for capturing real-time changes from the source. The transported tables may also be exchanged into larger tables as a partition.

Finally, in a multi-departmental enterprise environment, the target database as shown in Figure 23–2 may actually be an Operational Data Store that serves some intermediate reporting and updating purposes, but is still not the final data warehouse yet. This target could in turn serve as a source to the next stage of data collection. The Transportable Modules technology can be repeatedly applied, all the way till the data is finally collected in the data warehouse.

Transportable Modules can also be used for publishing data marts. A data mart is normally a portion of a larger data warehouse for single subject or single departmental access. At times, creating a data mart amounts to copying what has already been collected and processed in the data warehouse. A Warehouse Builder Transportable Module can be created to perform this task, as shown in Figure 23–3. The figure also shows that the same Transportable Module can be used to deploy a data mart to multiple locations.

Figure 23–3 Data Marts in a Data Warehouse



Since a Warehouse Builder Transportable Module deploys a snapshot of the source database objects, the deployment time (tracked by Warehouse Builder runtime deployment service) can serve as the version of the data marts.

About Transportable Modules and Oracle Database Technology

Transportable modules work by leveraging technology in Warehouse Builder plus technology in the Oracle Database. A transportable module replicates parts of a source database into a target database. The parts of the source database that can be replicated include tablespaces, tables, indexes, constraints, and other relational objects.

How the Oracle Database replicates the tablespace depends on the database version. When you transport between two 8*i* databases or between two 9*i* databases, the database invokes the Oracle transportable tablespaces functionality. When you transport between two Oracle 10*g* databases, the database invokes the Oracle data pump functionality.

In the case of 10g databases and Oracle data pump, you can transport tables without also transporting their tablespaces. For example, if your table is 100KB and its tablespace size is 10MB, you can deploy the table without deploying the entire tablespace. Furthermore, only Data Pump gives you the option to copy the entire schema. For 10*g* databases, you specify either data pump or transportable tablespaces during configuration as described in "Configuring a Transportable Module" on page 23-12.

For more information about transportable tablespaces and data pump, see the Oracle Database 10*g* documentation.

Benefits of Using Transportable Modules

Prior to the introduction of transportable modules, the most scalable data transportation mechanisms relied on moving flat files containing raw data. These mechanisms required that data be unloaded or exported into files from the source database, and then these files were loaded or imported into the target database.

Transportable modules entirely bypass the unload and reload steps and gives you access to the Oracle server technologies Transportable Tablespaces and Data Pump.

High Performance Data Extraction

Transportable modules reduce the need for Warehouse Builder mappings to access data remotely. If you have large volumes of data on remote machines, use transportable modules to quickly replicate the sources onto the Oracle target database. Warehouse Builder mappings can then directly access a local copy of the data. Also, because the source is now part of the target, you can perform the ETL operations directly on the source data.

Distribute and Archive Datamarts

Normally a central data warehouse handles ETL processing while dependent data marts are read-only. You can use transportable modules to copy from a read-only data mart to multiple departmental databases. In this way, you can use your central data warehouse to periodically publish new datamarts and then replace old datamarts simply by dropping the old tablespace and importing a new one. This rapid duplication and distribution also enables you to publish and distribute a datamart for daily analytical or business operations.

Archive Sources

You can set your source tablespaces to read-only mode and then export them to a target. All the data files are copied creating a consistent snapshot of the source database at a given time. This copy can then be archived. The advantage of this method is that archived data is restorable both in the source and target databases.

Instructions for Using Transportable Modules

Before You Begin

Ensure that you can connect to source and target databases as a user with the necessary roles and privileges as described in Roles and Privileges Required for Using Transportable Modules on page 23-6.

Ensure that your organization has licensed the Warehouse Builder Enterprise ETL Option.

To use transportable modules in Warehouse Builder, refer to the following sections:

Note to Database Administrators: Step 1 of these instructions require some powerful database roles and privileges. And step 3 requires knowledge of schema passwords. Depending on your preference, you can allow Warehouse Builder developers to complete step 3 or restrict it to database administrators only.

1. Specifying Locations for Transportable Modules on page 23-7

   Be sure to successfully test these connections before proceeding to the next step.

2. Creating a Transportable Module on page 23-8
3. Configuring a Transportable Module on page 23-12
4. Generating and Deploying a Transportable Module on page 23-15
5. Designing Mappings that Access Data Through Transportable Modules on page 23-17
6. Editing Transportable Modules on page 23-17

Roles and Privileges Required for Using Transportable Modules

When creating a Transportable Module source location, the source location user must possess specific roles and/or privileges depending on the version of the source database.

- If the source database is prior to Oracle Database 10*g*, the SYSDBA role is required of the source location user.
- If the source database is Oracle Database 10*g*, the SYSDBA is not required, but the following must be assigned to the source location user.
  - CONNECT role
  - EXP_FULL_DATABASE role
  - ALTER TABLESPACE privilege

When creating a Transportable Module target location, the target location user must possess specific roles and/or privileges depending on the version of the target database.

- If the target database is prior to Oracle Database 10*g*, the SYSDBA role is required of the target location user.
- If the target database is Oracle Database 10*g*, the SYSDBA role is not required but the following must be assigned to the target location user.
  - CONNECT role with admin option
  - RESOURCE role with admin option
  - IMP_FULL_DATABASE role
  - ALTER TABLESPACE privilege
  - EXECUTE_CATALOG_ROLE with admin option
  - CREATE MATERIALIZED VIEW privilege with admin option
  - CREATE ANY DIRECTORY privilege

Note: Transportable Module source and target location users need to be assigned many powerful roles and privileges in order for the Transportable Modules to read objects from the source database and for creating objects in the target database. In a production environment, if necessary, the DBA may choose to create the Transportable Module source and target locations (using the Warehouse Builder Connection Explorer) for the data warehouse developers, and conceal the passwords.

The following is a SQL script for DBA to assign source location users the required roles and privileges in the source database.

```
grant connect to <TM src location user>;

grant exp_full_database,alter tablespace to <TM src location
user>;
```

The following is a SQL script for DBA to assign target location users the required roles and privileges in the target database.

```
grant connect,resource to <TM tgt location user> with admin
option;
```

```
grant imp_full_database,alter tablespace to <TM tgt location
user>;

grant execute_catalog_role to <TM tgt location user> with admin
option;

grant create materialized view to <TM tgt location user> with
admin option;

grant create any directory to <TM tgt location user>;
```

Specifying Locations for Transportable Modules

Before you create a transportable module in Warehouse Builder, first define its source and target locations in the **Connection Explorer.** Each transportable module can have only one source and one target location.

To specify a transportable module location:

1. From the **Connection Explorer**, expand the **Locations** node.
2. Expand the **Databases** node.
3. Right-click either the **Transportable Modules Source Locations** or **Transportable Modules Target Locations** node and select **New.**

   Warehouse Builder displays a dialog for specifying the connection information for the source or target location.
4. The instructions for defining source and target locations are the same except that you do not specify optional FTP connection details for targets. Follow the instructions in "Transportable Module Source Location Information" to specify the connection information and then test the connection.

Transportable Module Source Location Information

Warehouse Builder first uses this connection information to import metadata for the transportable module from your source machine into the Warehouse Builder repository. Then at deployment, Warehouse Builder uses the connection information to move data from the source to the target.

Name

A name for the location of the source or target database.

Description

An optional description for the location.

Username/Password

Warehouse Builder uses the database user name and password to retrieve the metadata of the source objects you want to include in your transportable module. Warehouse Builder also uses this information during deployment to perform transportable tablespace or data pump operations.

To access databases for use with transportable modules, ensure that the user has the necessary database roles and privileges as described in Roles and Privileges Required for Using Transportable Modules on page 23-6.

Host

Host name of the machine where the database is located.

Port

Port number of the machine.

Service

Service name of the machine.

Version

Choose the Oracle Database version from the list.

FTP Username/Password (optional)

Specify an FTP credential if you intend to invoke Oracle Transportable Tablespace as the mechanism for transporting data. If you do not plan to configure the Transportable Tablespace mechanism, the FTP credential is not required.

Even in the case that you will configure to use the Transportable Tablespace, but both source and target databases are located in the same machine, or both source and target machines can access shared disk volumes (which, for instance, is made possible by NFS), you can leave the FTP credential blank. Without the FTP credential, Warehouse Builder will try to perform plain copy of the source files from source directory to target directory.

Test Connection

Click **Test Connection** to validate the connection information. Warehouse Builder attempts to connect to the source database and, if applicable, to the FTP service on the source machine. Warehouse Builder displays a success message only after it validates both credentials.

Creating a Transportable Module

Use the Create Transportable Module Wizard to select the schema objects you want to copy to a target database.

To create a transportable module:

1. From the Project Explorer, expand the **Databases** node.
2. Right-click the **Transportable Modules** node and select **New**.

   The Create Transportable Module Welcome page is displayed.
3. Use the wizard to complete the following tasks:

   Describing the Transportable Module

   Selecting the Source Location

   Selecting the Target Location

   Selecting Tablespaces and Schema Objects to Import

   Reviewing the Transportable Module Definitions

Describing the Transportable Module

In the Name and Description page, type a name and optional description for the transportable module.

Selecting the Source Location

Although you can create a new source location from the wizard page, it is recommended that you define locations for transportable modules before launching the wizard as described in "Transportable Module Source Location Information" on page 23-7

When you select an existing location, the wizard tests the connection and does not allow you to proceed with the wizard until you specify a location with a valid connection.

Selecting the Target Location

Select a target location from the list at the left of the wizard page. If no target locations are displayed, click New and define a target location as described in "Transportable Module Source Location Information" on page 23-7.

Selecting Tablespaces and Schema Objects to Import

Use the Define Contents page to select tablespaces and schema objects to include in the transportable module. In the left panel, Available Database Objects lists all source tablespaces, schemas, and available schema objects. In the right panel, Selected Database Objects displays the objects after you select and shuttle them over.

Expand the tablespaces by clicking the tree node to display the schemas in each tablespace and the objects in each schema. Non-tablespace schema objects such as views and sequences are also listed under their respective schema owners, even though these objects are not stored in the tablespace. To select multiple objects at the same time, hold down the Ctrl key while selecting them. You can include the following types of objects in transportable modules:

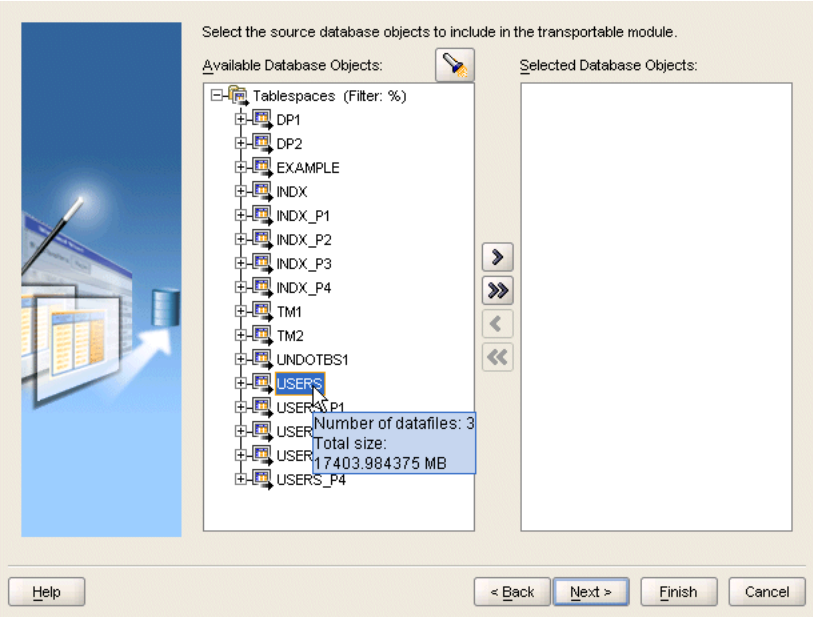
- Tables
- Views
- Materialized Views
- Sequences
- External Tables
- PL/SQL Functions, Procedures, and Packages
- Object Types, Varray Types, and Nested Tables Types

Select the tablespaces and schema objects from the Available Database Objects field and click the arrow buttons in the center to shuttle the objects to the Selected Database Objects field.

Available Database Objects

Because source databases typically contain multiple schemas and a large number of objects, there are several tools to assist you in finding and selecting objects. You can view the number of data files and their total size by placing your mouse over a node. The wizard displays the information in a pop up as shown in Figure 23–4.

Figure 23–4 Viewing the Number of Data files and Total Size

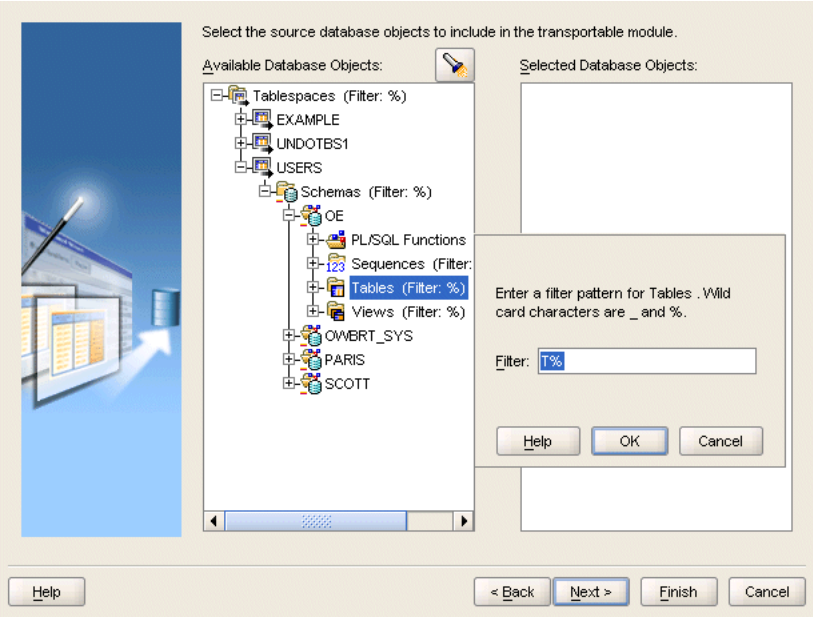


Finding Objects in the Available Database Object List: Double-click the flashlight icon to find source data objects by type or name. In the **Object** field, type a name or a letter by which to filter your search. From the **Type** list, indicate the object type you are searching. Check the appropriate box to perform the search by name or by description.

For example, type 'T%' in the Object field, select tablespaces from the Type field, and click **Find Next.** The cursor on the Available Database Objects navigation tree selects the name of the first tablespace that starts with a 'T'. If that is not the tablespace you want to select, then click **Find Next** to find the next tablespace. During this searching process, the navigation tree expands all the schema names and displays all the tablespaces.

Filtering the Available Database Objects List: As shown in Figure 23–5, you can double-click a schema node or any of the nodes in the schema to type in a filter pattern. For example, if you type T% and click **OK,** the navigation tree displays only those objects that start with the letter T.

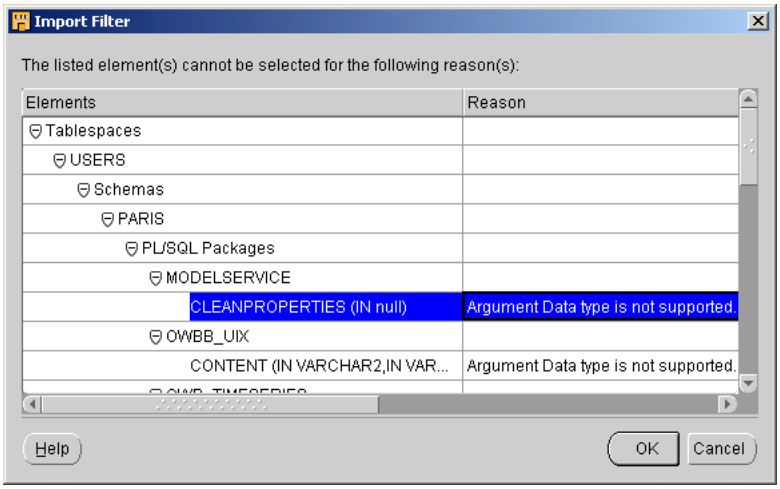
Figure 23–5 Schema Node Selected on Define Contents Page



Objects Not Available For Inclusion in Transportable Modules

If you select items that cannot be included in a transportable module, Warehouse Builder returns a dialog describing which items cannot be included and why.

Figure 23–6 Import Filter Dialog



Reviewing the Transportable Module Definitions

Review the summary information and click **Finish** to import metadata of the selected tablespace and schema objects.

After Warehouse Builder creates the transportable module in your repository, you can locate it on the Project Explorer under the Transportable Modules node. Expand the tree to display the imported definitions.

Warehouse Builder creates separate modules for the separate schemas. The schema names on the Project Explorer mirror the schema names in your source database.

Since the objects contained in a transportable module mirror the source database, you cannot edit these objects using this user interface. If the source database changes, you can re-import the objects. If you want to delete objects from the transportable module, right-click the object and select **Delete.** This action deletes the object from the definition of the transportable module but does not affect the underlying source database.

Configuring a Transportable Module

In the Project Explorer, right-click a transportable module and select **Configure** to configure it for deployment to the target database. You set configuration properties at the following levels:

- Transportable Module Configuration Properties
- Schema Configuration Properties
- Target DataFile Configuration Properties
- Tablespace Configuration Properties

For most use cases, you can accept the default settings for all the configuration properties with the exception of the Password setting. You must specify a password for each target schema. If the schema already exists in the target, specify an existing password. If the schema does not already exist, Warehouse Builder can create the schema with the password you provide.

Depending on your company security polices, knowledge of schema passwords may be restricted to database administrators only. In that case, the database administrator must specify the password for each schema. Alternatively, developers can define new passwords for new schemas if the target has no existing schemas that match source schemas.

Transportable Module Configuration Properties

Set the following runtime properties for the transportable module:

Target OS Type

Select the type of operating system type for the target. For versions prior to Oracle Database 10*g*, the type of operating system on the target machine must be the same as the source machine. For versions Oracle Database 10*g* or higher, you can deploy to any operating system from any operating system.

Work Directory

You should create a directory on the target machine dedicated for the deployment of transportable modules. This dedicated directory stores files generated at runtime including temporary files, scripts, log files, and transportable tablespace data files. If you do not create a dedicated directory and type its full path as the **Work Directory,** Warehouse Builder saves the generated files under the runtime home directory.

What to Deploy

Warehouse Builder enables you to select whether you want to deploy only the tables in your transportable module or all the related catalog objects, such as views and sequences, as well. Select **TABLES_ONLY** if you want to deploy only tables otherwise select **ALL_OBJECTS.**

Use the TABLES_ONLY option to refresh the data in a transportable module. If you previously deployed a transportable module with the ALL_OBJECTS option and want

to replace only the tablespace from the same source, redeploy the transportable module with the TABLES_ONLY option. The deployment drops the existing tablespace in the target, inserts the new one, and then recompiles the previously deployed metadata.

Similarly, if you previously deployed the Transportable Module using Data Pump, then the redeployment will only modify the tables in the Transportable Module.

Transport Tablespace

By default, this setting is enabled and Warehouse Builder transports the tablespaces. If you enable this setting, also specify the settings under Target DataFile Configuration Properties.

If both the source and target databases are Oracle 10*g* or higher, consider disabling this setting. For example, if your table is 100KB and its tablespace size is 10MB, you can deploy the table without deploying the entire tablespace. When you disable **Transport Tablespace,** Warehouse Builder utilizes Oracle Data Pump to deploy the table and you can set the Table Exists Action setting.

Note also that if source or target location is not Oracle Database 10*g*, the Transport Tablespace option is selected by default. In that case, Transportable Tablespace is the only implementation method for data movement. If both source and target locations are Oracle Database 10*g*, then you can deselect Transport Tablespace and therefore utilize Data Pump.

In the case that Transport Tablespace is selected, there are further restrictions, depending on the versions of the source and target locations, as described in Table 23–1. When planning for data replications, take these restrictions into consideration. In general, Oracle10*g*, particularly Oracle10*g* Release 2, is the preferred target database.

Table 23–1 Requirements for Replicating Data Between Database Versions

Source location	Target location
10g	Targeting another Oracle 10*g* location requires both databases must have the same character set and the same national character set. Targeting an Oracle8*i* or 9i location is not possible.
9i	Targeting an Oracle9*i* or 10*g* location requires both databases must have the same character set, the same national character set, and both databases must be on the same operating system platform. Targeting an Oracle8*i* or 9i location is not possible.
8i	Targeting an Oracle8*i*, 9*i*, or 10*g* requires all of the following: Both source and target databases must have the same character set. Both source and target databases must have the same national character set. Both source and target databases must be on the same operating system platform. Both source and target databases must have the same block size. Cannot change schema names during transporting tablespaces. Cannot change tablespace names during transporting tablespaces.

Table 23–1 (Cont.) Requirements for Replicating Data Between Database Versions

Source location	Target location
10g	Targeting another Oracle10*g* location requires both databases must have the same character set and the same national character set. Targeting an Oracle8*i* or 9*i* location is not possible.

Schema Configuration Properties

Set the following schema properties for the transportable module:

Target Schema Name

This property enables you to change the name of the source schema when it is deployed to the target. Select the DEFAULT or click the Ellipsis button to type the new name for your schema in the target and click **OK.** For example, you can change SCOTT to SCOTT1.

Password

For existing schemas, type a valid password for the schema. For schemas to be created, Warehouse Builder creates the schema with the password you provide.

Default Tablespace

Specify the default tablespace to be used when Warehouse Builder creates the target schema. If you leave this setting blank, Warehouse Builder uses the default specified by the target.

Schema Exists Action

Specify what action Warehouse Builder should take if the schema already exists in the target. The default value is skip.

Schema Doesn't Exist Action

Specify what action Warehouse Builder should take if the schema does not already exist in the target. The default value is create.

Table Exists Action

When Transport Tablespace is disabled, use this property to specify what action Warehouse Builder should take if the table already exists in the target. Default value is skip.

Copy Source Schema

When you use data pump by deselecting Transport Tablespace, you can select this option to copy the entire source schema into the target.

Parallel

When you use data pump by deselecting Transport Tablespace, specify the maximum number of processes for the Oracle Database to use to execute the transfer of data.

Target DataFile Configuration Properties

You need to set the following data file properties for the transportable module:

Directory

Indicate the directory where you want the data file to be stored on your target machine. If you leave the directory unspecified, Warehouse Builder stores the data file in the Work Directory.

File Name

Specify the name of the data file to be created in the target machine. You can use this parameter to rename the data file. Accept the DEFAULT to persist the data file name from the source database or click the Ellipsis button to type a new name for the datafile and click **OK.**

Overwrite

If this parameter is selected, Warehouse Builder overwrites the existing data file. Else, Warehouse Builder aborts the deployment if it finds an existing data file.

Tablespace Configuration Properties

When you enable Transport Tablespace, set the following tablespace properties for the transportable module:

Tablespace Name

If you are using an Oracle Database version prior to 10g, then the target tablespace name must be the same as your source tablespace name. For such cases, this field is read-only. If a tablespace with the same name already exists in your target database, then the runtime operation will first drop the existing tablespace and replace it with the new one.

If you are using Oracle Database version 10g or higher, then you can change the target tablespace name.

Drop Existing Tablespace

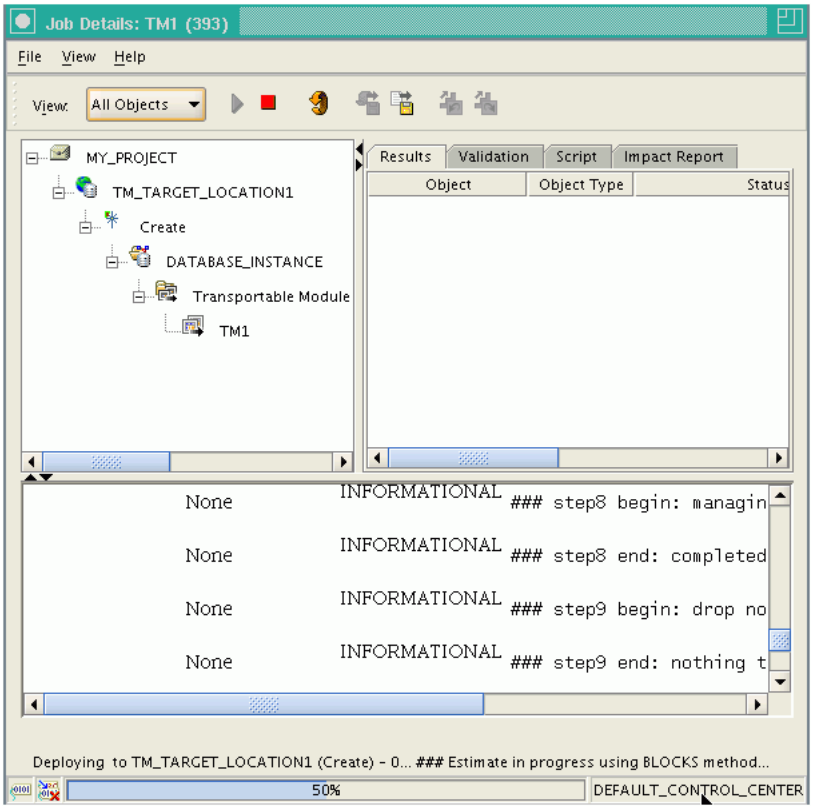
If this setting is selected, Warehouse Builder drops and then recreates the existing tablespace in target. By default, this setting is not selected and prevents you from deleting the tablespace in the target in the event that the tablespace with the same name already exists. In this case, the deployment process stops with an error.

Generating and Deploying a Transportable Module

When you deploy a transportable module, the Control Center displays the transportable module as including all the tables while the other catalog objects such as views are displayed separately. When you select a deploy action for the transportable module, the Control Center sets the associated catalog objects to the same deploy action.

During deployment of a Transportable Module, there are two ways for users to monitor the deployment progress. The first way is by the use of the "Job Details" window, as shown in Figure 23–7. The status line is instantly refreshed with the most up-to-date status. And the message box immediately above the status line shows all the messages logged so far.

Figure 23–7 The Job Details Window



Another way of observing the progress is by watching the log file that the Transportable Module deployment process generates. The Transportable module log file is created in the "Work Directory" that the user has configured. The name of the file is always <The TM Name>.log, for example TM1.log if the name of the Transportable Module is TM1. This file is a plain text file containing the same messages that you can see in the message box in the Job Details window. Example 23–1 shows the contents of a Transportable Module log file.

Currently there are a total of 16 steps. Some steps may be skipped depending on the user configurations, and some steps may contain error messages that Transportable Module considers ignorable, such as failures in creating referential constraints due to referenced tables not found errors. This log file contains important information. It must be carefully examined during and after the Transportable Module deployment completes.

Example 23–1 Log file containing important information

```
step1 begin: making connection to target db ...
step1 end: connected to target
Target ORACLE_HOME = /data/oracle/ora1010
step2 begin: making connection to source db...
step2 end: skipped.
step3 begin: making source tablespaces read only...
step3 end: skipped.
step4 begin: exporting tts...
step4 end: skipped.
step 5 begin: checking for existing datafiles on target...
step5 end: skipped.
step 6 begin: drop existing tablespaces
```

```
step6 end: skipped.
step7 begin: transporting datafiles...
step7 end: skipped.
step8 begin: managing schemas/users ...
step8 end: completed setting up target schemas
step9 begin: drop non-table schema objects...
step9 end: nothing to drop.
step10 begin: converting datafiles...
step10 end: skipped.
step 11 begin: importing tts ...
find or create a useable dblink to source.
step11 end: importing tts is not requested by user.
step 11 end: import tts is successful
step 12 begin: restore source tablespaces original status ...
step12 end: skipped.
step13 end: skipped.
step14 begin: non-tts import ...

Import: Release 10.1.0.4.0 - Production on Tuesday, 04 April, 2006 10:43
Copyright (c) 2003, Oracle.  All rights reserved.

Username:
Connected to: Oracle Database 10g Enterprise Edition Release 10.1.0.4.0 -
Production
With the Partitioning, OLAP and Data Mining options
Starting "TMTGT_U"."SYS_IMPORT_TABLE_02":
TMTGTU/********@(DESCRIPTION=(ADDRESS=(HOST=LOCALHOST)(PROTOCOL=tcp)(PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=ORA1010.US.ORACLE.COM)))
parfile=/home/ygong/tmdir/TM1_imptts.par
Estimate in progress using BLOCKS method...
Processing object type TABLE_EXPORT/TABLE/TBL_TABLE_DATA/TABLE/TABLE_DATA
Total estimation using BLOCKS method: 64 KB
Processing object type TABLE_EXPORT/TABLE/TABLE
. . imported "TMU1"."TA"                                    2 rows
Processing object type TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/REF_CONSTRAINT
ORA-39083: Object type REF_CONSTRAINT failed to create with error:
ORA-00942: table or view does not exist
Failing sql is:
ALTER TABLE "TMU1"."TA" ADD CONSTRAINT "TA_T1_FK" FOREIGN KEY ("C") REFERENCES
"TMU1"."T1" ("C") ENABLE

Job "TMTGT_U"."SYS_IMPORT_TABLE_02" completed with 1 error(s) at 10:44
step14: import has failures.
step14 end: non-tts import completed with warnings
step15 end: create flat file directories skipped.
step16 end: transporting flat files skipped.
```

Designing Mappings that Access Data Through Transportable Modules

Once you successfully deploy a transportable module, you can use the objects in the transportable module in ETL designs. When you add source and target operator to the mapping as described in "Adding Operators that Bind to Repository Objects" on page 6-12, you can select objects from the transportable module folder.

Editing Transportable Modules

A transportable module is located under the Transportable Modules node within the Databases node on the Warehouse Builder Project Explorer.

You can edit a transportable module by right-clicking the name of transportable module from the Warehouse Builder Project Explorer and selecting **Open Editor** from the pop-up menu. Warehouse Builder displays the Edit Transportable Module dialog containing four tabs.

Name

From the Name tab, you can edit the name and description of the transportable module.

Source Location

Warehouse Builder uses this connection information to access the source machine and import the metadata into its repository. Warehouse Builder also uses this information during runtime to move the tablespace data from the source to the target.

The Source Database tab is read-only. Once you have imported tablespace definitions from a source machine, you cannot change the location information.

Tablespaces

The Tablespaces tab displays the tablespaces to be transported and their size. This tab is read-only. You can also view the tablespace size for individual data files in a tablespace. For details, see "Viewing Tablespace Properties" on page 23-18.

Target Locations

Displays the available and selected target locations. You can move a location from Available Locations to Selected Locations, or configure a new location.

Viewing Tablespace Properties

You can view the properties of a tablespace by right-clicking the name of tablespace from the Warehouse Builder Project Explorer and selecting **Open Editor** from the pop-up menu. Warehouse Builder opens the Edit Tablespace. This property sheet displays the size of individual data files in a tablespace.

Reimporting Metadata into a Transportable Module

If your source data has changed since you last created a transportable module in Warehouse Builder, you can reimport the metadata to update your repository definitions. When you open the reimport dialog, Warehouse Builder remembers the source location you specified while creating the transportable module and directly displays your source objects.

To reimport transportable module definitions:

1. From the Warehouse Builder Project Explorer, right-click the **Transportable Modules** name and select **Reimport**.

   The Recreate Transportable Module dialog is displayed.

2. From the Available Database Objects column, select the objects you want to reimport.

   The database objects that have been previously imported into the repository are listed in bold. You can also choose to import new definitions.

3. Use the arrow buttons to shuttle the objects to the Selected Database Objects column and click **OK.**

Warehouse Builder reimports existing definitions and creates new ones. The Transportable Module reflects the changes and updates after the reimport is completed.

24

ETL Objects Configuration

Earlier in the design phase, you defined a logical model for your target system using Warehouse Builder design objects. This chapter includes reference information for assigning physical properties to mappings and process flows. This chapter presents configuration properties in the order they appear in the Warehouse Builder user interface.

This chapter includes:

Configuring Mappings Reference

When you correctly configure mappings, you can improve the ETL performance. Use this section as a reference for setting configuration parameters that govern how Warehouse Builder loads data and optimizes code for better performance.

This section includes the following topics:

Procedure for Configuring Mappings

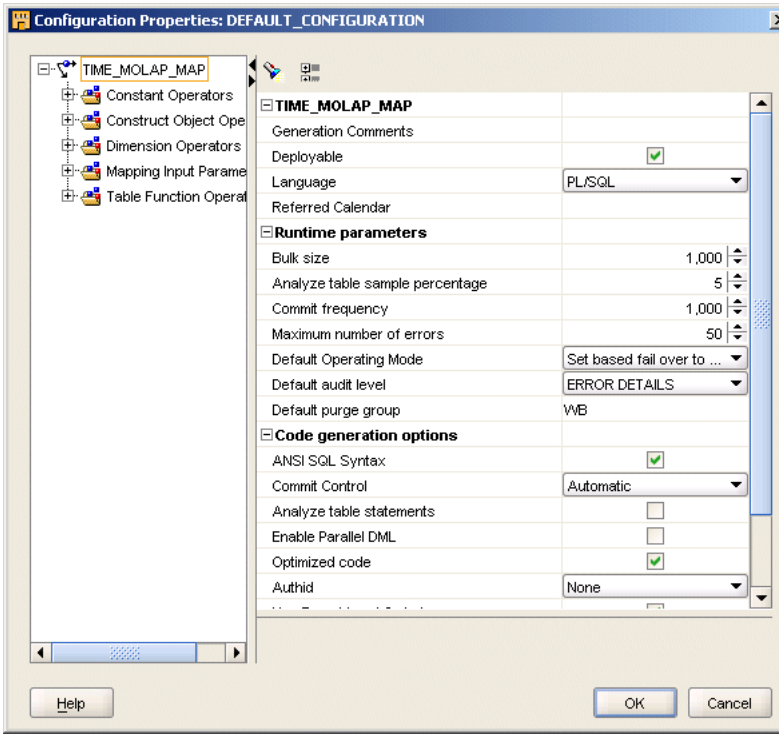
To configure physical properties for a mapping:

1. Right-click a mapping from the Project Explorer and select **Configure** from the pop-up menu.

   Warehouse Builder displays the configuration Properties dialog for a mapping. You can configure properties for the mapping itself and the operators within the mapping.

   Figure 24–1 displays the Configuration Properties dialog.

Figure 24–1 Configuration Properties Dialog For A Mapping



2. Select **Deployable** to enable Warehouse Builder to generate a set of scripts for mapping entities marked as deployable.

   Deployable is enabled by default. If you disable it, Warehouse Builder does not generate scripts for that mapping.

3. Set **Language** to the type of code you want to generate for the selected mapping.

   The options from which you can choose depend upon the design and use of the operators in the mapping. Warehouse Builder sets the correct value depending on the mapping: PL/SQL, SQL*Loader, ABAP (for an SAP source mapping).

4. If you want to schedule the mapping to run based on a previously defined schedule, click **Referred Calendar.**

   For instructions on creating and using schedules, see Chapter 28, "Scheduling ETL Objects".

5. Expand the **Runtime Parameters** to configure your mapping for deployment.

   For a description of each runtime parameter, see "Runtime Parameters" on page 24-3.

6. Expand **Code Generation Options** to enable performance options that optimize the code generated for the mapping.

   For a description of each option, see "Code Generation Options" on page 24-4.

7. Go to the node for each operator in the mapping to set their physical properties.

   For information on configuring sources and targets in a mapping, see "Sources and Targets Reference" on page 24-6. To configure mappings with flat file sources and targets, see "Configuring Flat File Operators" on page 24-10.

Runtime Parameters

When you configure Runtime Parameters for a mapping, you set the default behaviors for the mapping. You can override these parameters when you execute the mapping either in the Control Center, the Process Flow Editor, or Oracle Enterprise Manager.

The Runtime Parameters include the following parameters:

- Bulk Size
- Analyze Table Sample Percentage
- Commit Frequency
- Maximum Number of Errors
- Default Operating Mode
- Default Audit Level
- Default Purge Group

Bulk Size

Use **Bulk Size** to specify the number of rows in each bulk for PL/SQL Bulk Processing. Warehouse Builder uses the Bulk Size parameter only when Bulk Processing Code option is selected and the operating mode is set to row based. For more information, see the *Oracle PL/SQL Reference Guide.*

Analyze Table Sample Percentage

When you select the Analyze Table Statements option, Warehouse Builder estimates when gathering statistics on the target tables. After data is loaded into the target tables, statistics used for cost-based optimization are gathered on each target table. You can set this parameter to the percentage of rows in each target table used for this analysis.

Commit Frequency

Commit frequency applies only to non-bulk mode mappings. Bulk mode mappings commit according to the bulk size.

When you set the **Default Operating Mode** to row based and Bulk Processing Code to false**,** Warehouse Builder uses the **Commit Frequency** parameter when executing the package. Warehouse Builder commits data to the database after processing the number of rows specified in this parameter.

If you select the Bulk Processing Code option, set the Commit Frequency equal to the Bulk Size. If the two values are different, Bulk Size overrides the commit frequency and Warehouse Builder implicitly performs a commit for every bulk size.

Maximum Number of Errors

Use **Maximum Number of Errors** to indicate the maximum number of errors allowed when executing the package. Execution of the package terminates when the number of errors exceeds the maximum number of errors value.

Default Operating Mode

For mappings with a PL/SQL implementation, select a default operating mode. The operating mode you select can greatly affect mapping performance. For details on how operating modes affect performance, see "Set Based Versus Row Based Operating Modes" on page 8-4. You can select one of the following operating modes:

- **Set based:** Warehouse Builder generates a single SQL statement that inserts all data and performs all operations on the data. This increases the speed of Data Manipulation Language (DML) operations. Set based mode offers optimal performance but minimal auditing details.
- **Row based:** Warehouse Builder generates statements that process data row by row. The select statement is a SQL cursor. All subsequent statements are PL/SQL. Since data is processed row by row, the row based operating mode has the slowest performance but offers the most auditing details.
- **Row based (Target Only):** Warehouse Builder generates a cursor select statement and attempts to include as many operations as possible in the cursor. For each target, Warehouse Builder generates a PL/SQL insert statement and inserts each row into the target separately.
- **Set based fail over row based:** Warehouse Builder executes the mapping in set based mode. If an error occurs, the execution fails and Warehouse Builder starts the mapping over again in the row based mode. This mode is recommended for use only in test environments and is not recommended for use in production environments.
- **Set based fail over row based (Target Only):** Warehouse Builder first executes the mapping in set based mode. If an error occurs, the execution fails over to **Row based (Target Only)** mode. This mode is recommended for use only in test environments and is not recommended for use in production environments.

Default Audit Level

Use **Default Audit Level** to indicate the audit level used when executing the package. Audit levels dictate the amount of audit information captured in the runtime schema when the package is run. The audit level settings are:

- **None:** No auditing information is recorded in runtime.
- **Statistics:** Statistical auditing information is recorded in runtime.
- **Error Details:** Error information and statistical auditing information is recorded in runtime.
- **Complete:** All auditing information is recorded in runtime. Running a mapping with the audit level set to **Complete** generates a large amount of diagnostic data which may quickly fill the allocated tablespace.

Default Purge Group

The Default Purge Group is used when executing the package. Each audit record in the runtime schema is assigned to the purge group specified.

Code Generation Options

The Code Generation Options include the following:

- ANSI SQL Syntax
- Commit Control
- Analyze Table Statements
- Enable Parallel DML
- Optimized Code
- Authid

- Use Target Load Ordering
- Error Trigger
- Bulk Processing Code
- Generation Mode

ANSI SQL Syntax

If you select this option, Warehouse Builder generates ANSI SQL syntax. If the option is not selected, the Warehouse Builder generates Oracle SQL syntax.

Commit Control

Automatic: This is the default setting. Warehouse Builder loads and then automatically commits data based on the mapping design. This setting is valid for all mapping types. For multiple targets in a single mapping, Warehouse Builder commits data based on target by target processing (insert, update, delete).

Automatic Correlated: Automatic correlated commit is a specialized type of automatic commit that applies only to PL/SQL mappings with multiple targets. Warehouse Builder considers all targets collectively and commit or rolls back data uniformly across all targets.

The mapping behavior varies according to the operating mode you select. For more information about correlated commit, see "Committing Data from a Single Source to Multiple Targets" on page 8-7.

Manual: Select manual commit control for PL/SQL mappings that you want to interject complex business logic or perform validations before committing data.

You have two options for specifying manual commits. You can define the commit logic within the mapping as described in "Embedding Commit Logic into the Mapping" on page 8-9.

Alternatively, you can commit data in a process flow or from a SQL Plus session as described in "Committing Data Independently of Mapping Design" on page 8-10.

Analyze Table Statements

If you select this option, Warehouse Builder generates code for analyzing the target table after the target is loaded, if the resulting target table is double or half its original size.

Enable Parallel DML

If you select this option, Warehouse Builder enables Parallel DML at runtime. Executing DML statements in parallel improves the response time of data-intensive operations in large databases that are present in a data warehouse.

Optimized Code

Select this option to improve performance for mappings that include the Splitter operator and inserts into multiple target tables. When this option is selected and the mapping is executed by Oracle*9i* or higher, Warehouse Builder generates a single SQL statement (`multi_table_insert`) that inserts data into multiple tables based on same set of source data.

Warehouse Builder performs the multiple table insert only if this parameter is selected and the Oracle target module database is Oracle*9i* or higher. The multiple tables insert is performed only for mappings in set based mode that include a Splitter operator and

no active operators such as an Aggregator or Joiner operator between the Splitter and the target. Also, the multiple insert is available only for tables, not views, materialized views, dimensions, or cubes. Each target table must have fewer than 999 columns. For detailed instructions on how to create a mapping with multiple targets, see "Example: Creating Mappings with Multiple Targets" on page 26-33.

Set this parameter to false for mappings run in row based mode or for mappings executed by Oracle*8i* server. You may also want to set this parameter to false when auditing is required. When this option is selected, Warehouse Builder returns one total SELECT and INSERT count for all targets.

Authid

Specifies the AUTHID option to be used while generating the code. The options you can select are Current_User, Definer, or None.

Use Target Load Ordering

For PL/SQL mappings with multiple targets, you can generate code that defines an order for loading the targets. This is important when a parent child relationship exists between two or more targets in a mapping. The option is selected by default.

Error Trigger

Specify the name of the error trigger procedure in this field.

Bulk Processing Code

If this configuration parameter is selected and the operating mode is set to row based, Warehouse Builder generates PL/SQL bulk processing code. PL/SQL bulk processing improves row-based ETL performance by collecting, processing, and writing rows in bulk, instead of doing it row by row. The size of each bulk is determined by the configuration parameter Bulk Size. Set based mode offers optimal performance, followed by bulk processing, and finally by row based mode. For more information, see the *Oracle PL/SQL Reference Guide.*

Generation Mode

By default, when Warehouse Builder generates code for a mapping, it generates the code for all possible operating modes. That is, if you set Default Operating Mode to set based, Warehouse Builder still generates code for all possible operating modes when **Generation Mode** is set to All Operating Modes. This enables you to switch the operating modes for testing purposes at runtime.

Sources and Targets Reference

For relational and dimensional sources and targets such as tables, views, and cubes, Warehouse Builder displays the following set of properties for each operator:

- Use LCR APIs
- Database Link
- Location
- Conflict Resolution
- Schema
- Partition Exchange Loading
- Hints

- Constraint Management
- SQL*Loader Parameters

Use LCR APIs

By default, this setting is enabled and Warehouse Builder performs DML using LCR APIs if available. If no LCR APIs are available, Warehouse Builder uses the standard DML.

Database Link

This parameter is maintained for backward compatibility only.

In previous version of Warehouse Builder, you could select a database link by name from the drop-down list. Source operators can be configured for schemas and database links, but targets can be configured for schemas only. Sources and targets can reside in different schemas, but they must reside in the same database instance.

Location

This setting specifies the location Warehouse Builder uses to access the source or target operator.

Conflict Resolution

Enable this setting to detect and resolve any conflicts that may arise during DML using the LCR APIs.

Schema

This parameter is maintained for backward compatibility only.

In previous version of Warehouse Builder, you could link the mapping to a particular schema by clicking on the Schema field and typing a name.

Partition Exchange Loading

Use setting in this section to enable partition exchange loading (PEL) into a target table. For specific information on each of these settings and additional information on how to design mappings for PEL, see "Improved Performance Through Partition Exchange Loading" on page 8-24.

Hints

Define loading or extraction hints. Application developers often develop insights into their data. For example, they know that a query runs much faster if a set of tables is joined in one order rather than another. Warehouse Builder can incorporate these insights into the generated SQL code packages as SQL Optimizer Hints.

When you select a hint from the Hints dialog, the hint appears in the Existing Hints field. Type additional text as appropriate in the Extra Text column. The editor includes the hint in the mapping definition as is. There is no validation or checking on this text.

You can define loading hints for mappings that load data in INSERT or UPDATE mode. By default, Warehouse Builder adds commonly used hints such as APPEND and PARALLEL. For all loading modes other than INSERT, the APPEND hint causes no effect and you can choose to remove it.

Hint is available during mapping configuration. To configure the hint:

1. In Project Explorer, expand **Database** folder.

2. In Database, expand the repository module.
3. In Repository module, expand **Mappings**.
4. In Mappings, select the required mapping.
5. Right-click Mapping and select **Configure**.
6. In Configuration Properties window, expand the required operator.
7. Select the operator function.
8. To open the Inline View Hint window, click the ellipses next to the **Inline View Hint** option available in the Configuration Properties window.

For information on optimizer hints and how to use them, see *Oracle9i Designing and Tuning for Performance.*

Constraint Management

Configure the following Constraint Management parameters:

- **Exceptions Table Name:** All rows that violated their foreign key constraints during re-enabling are logged into the specified exceptions table. No automatic truncation of this table is performed either before or after the load. Constraint violations are also loaded into the runtime audit error tables.

  For SQL and PL/SQL loading, if you do not specify an exceptions table, invalid rows load into a temporary table located in the default tablespace and then load into the Runtime Audit error table. The table is dropped at the end of the load.

  If you are using SQL*Loader direct path loading, you must specify an exception table. Consult the SQL*Loader documentation for more information.

- **Enable Constraints:** If you select this option, Warehouse Builder maintains the foreign key constraints on the target table before loading data. If this option is not selected, Warehouse Builder disables the foreign key constraints on target tables before data loading and then re-enables the constraints after loading. Constraint violations found during re-enable are identified in the runtime audit error table and, if specified, in an exceptions table.

  When you disable constraints, loading time is decreased since constraint checking is not performed. However, if exceptions occur for any rows during re-enabling, the constraints for those rows will remain in a non-validated state. These rows are logged in the runtime audit error table by their ROWID. You must manually inspect the error rows to take any necessary corrective action.

  Setting the Enable Constraints to false is subject to the following restrictions:

  – For set based operating mode, the false setting disables foreign key constraints on the targets before loading, and then re-enables the constraints after loading. This property has no effect on foreign key constraints on other tables referencing the target table. If the load is done using SQL*Loader instead of a SQL or PL/SQL package, then a re-enable clause is added to the .ctl file.

  – For set based fail over to row based and set based fail over to row based (target only) operating modes, the false setting disables the foreign key constraints on the targets before loading and then re-enables them if the load succeeds in set based mode. This setting has no effect on foreign keys referencing other tables. If the load fails over to row-based, then loading will repeat in row based mode and all constraints remain enabled.

Note: Constraint violations created during re-enabling will not cause the load to fail from set based over to row based mode.

- For row based or row based (target only) operating modes, all foreign key constraints remain enabled even if the property is set to false.
- For the TRUNCATE/INSERT DML type, the false setting disables foreign key constraints on other tables referencing the target table before loading, and then re-enables the constraints after loading, regardless of the default operating mode.

SQL*Loader Parameters

When you have a table operator that contains inputs from a flat file, you need to configure the following SQL*Loader Parameters properties:

- **Partition Name:** Indicates that the load is a partition-level load. Partition-level loading enables you to load one or more specified partitions or subpartitions in a table. Full database, user, and transportable tablespace mode loading does not support partition-level loading. Because incremental loading (incremental, cumulative, and complete) can be done only in full database mode, partition-level loading cannot be specified for incremental loads. In all modes, partitioned data is loaded in a format such that partitions or subpartitions can be selectively loaded.
- **Sorted Indexes Clause:** Identifies the indexes on which the data is presorted. This clause is allowed only for direct path loads. Because data sorted for one index is not usually in the right order for another index, you specify only one index in the SORTED INDEXES clause. When the data is in the same order for multiple indexes, all indexes can be specified at once. All indexes listed in the SORTED INDEXES clause must be created before you start the direct path load.
- **Singlerow:** Intended for use during a direct path load with APPEND on systems with limited memory, or when loading a small number of records into a large table. This option inserts each index entry directly into the index, one record at a time. By default, SQL*Loader does not use SINGLEROW to append records to a table. Index entries are stored in a temporary area and merged with the original index at the end of the load. Although this method achieves better performance and produces an optimal index, it requires extra storage space. During the merge, the original index, the new index, and the space for new entries all simultaneously occupy storage space. With the SINGLEROW option, storage space is not required for new index entries or for a new index. Although the resulting index may not be as optimal as a freshly sorted one, it takes less space to produce. It also takes more time because additional UNDO information is generated for each index insert. This option is recommended when the available storage is limited. It is also recommended when the number of records to be loaded is small compared to the size of the table. A ratio of 1:20 or less is considered small.
- **Trailing Nullcols:** Sets SQL*Loader to treat any relatively positioned columns that are not present in the record as null columns.
- **Records To Skip:** Invokes the SKIP command in SQL*Loader. SKIP specifies the number of logical records from the beginning of the file that should not be loaded. By default, no records are skipped. This parameter continues loads that have been interrupted for some reason. It is used for all conventional loads, for single-table direct loads, and for multiple-table direct loads when the same number of records are loaded into each table. It is not used for multiple-table direct loads when a different number of records are loaded into each table.

- **Database File Name:** Specifies the names of the export files to import. The default extension is .dmp. Because you can export multiple export files, you may need to specify multiple filenames to be imported. You must have read access to the imported files. You must also have the IMP_FULL_DATABASE role.

Configuring Flat File Operators

The Configuration Properties dialog contains additional settings for Mapping Flat File operators, depending on how the operators are used in the mapping.

- **Flat File Operators as a Target:** Warehouse Builder generates a PL/SQL deployment code package. For information on configuring the parameters associated with a Mapping Flat File operator used as a target, see "Flat File Operators as a Target" on page 24-10.
- **Flat File Operator as a Source:** Warehouse Builder generates SQL*Loader scripts. For information on the parameters associated with a Mapping Flat File operator used as a source, see "Flat File Operator as a Source" on page 24-11.

Flat File Operators as a Target

To configure properties unique to mappings with flat file targets:

1. Select a mapping from the Project Explorer, select **Design** from the menu bar, and select **Configure.**

   Or, right-click the mapping you want to configure and select **Configure** from the pop-up menu.

   Warehouse Builder displays the Configuration Properties dialog.

2. Choose the parameters you want to configure and click the space to the right of the parameter name to edit its value.

   For each parameter, you can either select an option from a list, type a value, or click the Ellipsis button to display another properties dialog.

3. Select the **Deployable** option to enable Warehouse Builder to generate a set of scripts for mapping objects marked as deployable. If this option is not selected for a mapping, Warehouse Builder does not generate scripts for that mapping.

4. Set **Language** to the type of code you want to generate for the selected mapping. The options you can choose from depend upon the design and use of the operators in the mapping. Depending on the mapping, you can select from PL/SQL, ABAP (for an SAP source mapping), or SQL*Loader.

5. Specify the location to deploy the mapping.

6. Under **Runtime Parameters**, set the **Default Operating Mode** to **Row based (target only)**. This type of mapping will not generate code in any other default operating mode. For a description of each runtime parameter, see "Runtime Parameters" on page 24-3.

7. Set the **Code Generation Options** as described in "Code Generation Options" on page 24-4.

8. Set the Sources and Targets Reference as described in "Sources and Targets Reference" on page 24-6.

9. For **Access Specification,** specify the name of the flat file target in **Target Data File Name.** For the **Target Data File Location,** specify a target file located on the

machine where you installed the Warehouse Builder Runtime Platform. Select **Output as XML file** if you want the output to be in an xml file.

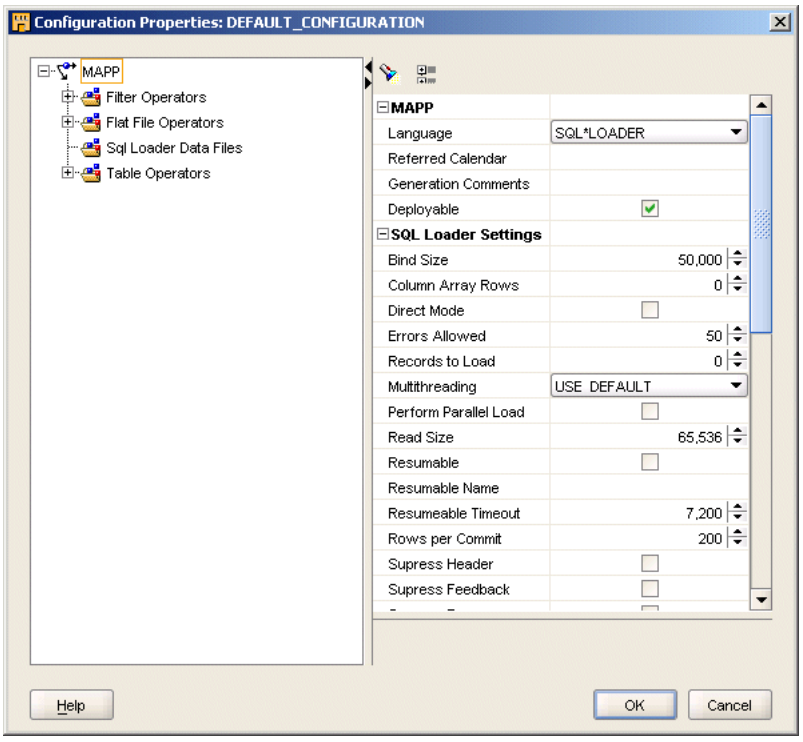
Flat File Operator as a Source

To configure a mapping with a flat file operator as a source:

1. Select a mapping from the Project Explorer, select **Design** from the menu bar, and select **Configure.** Or, right-click the mapping you want to configure and select **Configure** from the pop-up menu.

   The Configuration Properties dialog appears as shown in Figure 24–2.

Figure 24–2 Configuration Properties Dialog (Flat File Source)



2. Choose the parameters you want to configure and click the space to the right of the parameter name to edit its value.

   For each parameter, you can specify whether you want the parameter to be selected, select an option from a list, type a value, or click the Ellipsis button to display another properties dialog.

3. Select the **Deployable** option to generate SQL*Loader script.

4. Specify the **Log File Location** and **Log File Name.**

5. Select **Continue Load.**

   If SQL*Loader runs out of space for data rows or index entries, the load is discontinued. If **Continue Load** option is selected, Warehouse Builder attempts to continue discontinued loads.

6. In **Nls Characterset,** specify the character set to place in the CHARACTERSET clause.

7. Select **Direct Mode** to indicate that a direct path load will be done. If this option is not set, a conventional load will be done. In general, direct mode is faster.
8. Select **Operation Recoverable** to indicate that the load is recoverable. If this option is not selected, the load is not recoverable and records are not recorded in the redo log.
9. Configure the following parameters that affect the OPTIONS clause in the SQL *Loader scripts Warehouse Builder generates for mappings with flat file sources.

   Perform Parallel Load: If this option is selected, direct loads can operate in multiple concurrent sessions.

   Errors Allowed: If the value specified is greater than 0, then the ERRORS = n option is generated. SQL*Loader terminates the load at the first consistent point after it reaches this error limit.

   Records To Skip: If the value specified is greater than 0, then the SKIP = n option is generated. This value indicates the number of records from the beginning of the file that should not be loaded. If the value is not specified, no records are skipped.

   Records To Load: If the value specified is greater than 0, then the LOAD = n option will be generated. This value specifies the maximum number of records to load. If a value is not specified all of the records are loaded.

   Rows Per Commit: If the value specified is greater than 0, then the ROWS = n option is generated. For direct path loads, the value identifies the number of rows to read from the source before a data is saved. For conventional path loads, the value specifies the number of rows in the bind array.

   Read Size: If the value specified is greater than 0, then the READSIZE = n option is generated. The value is used to specify the size of the read buffer.

   Bind Size: If the value specified is greater than 0, then the BINDSIZE = n option is generated. The value indicates the maximum size in bytes of the bind array.

   Read Buffers: If the value specified is greater than 0, then the READBUFFERS n clause is generated. READBUFFERS specifies the number of buffers to use during a direct path load. Do not specify a value for READBUFFERS unless it becomes necessary.

   Preserve Blanks: If this option is selected, then the PRESERVE BLANKS clause is generated. PRESERVE BLANKS retains leading white space when optional enclosure delimiters are not present. It also leaves the trailing white space intact when fields are specified with a predetermined size.

   Database File Name: This parameter enables you to specify the characteristics of the physical files to be loaded. The initial values for these parameters are set from the properties of the flat file used in the mapping.

   If this parameter is set to a non-blank value, then the FILE= option is generated. The value specified is enclosed in single quotes in the generated code.

   Control File Location and **Control File Name:** The control file name necessary for audit details.

   For more information on each SQL*Loader option and clause, see *Oracle Database Utilities 10g*.
10. Expand the **Runtime Parameters** to configure your mapping for deployment.

    Audit: Select this option to perform an audit when the package is executed.

Default Purge Group: The Default Purge Group is used when executing the package. Each audit record in the runtime schema is assigned to the purge group specified.

11. Expand **Sources and Targets Reference** to set the physical properties of the operators in the mapping as described in "Sources and Targets Reference" on page 24-6.

Configuring Process Flows

To configure a process flow module:

1. Right-click the process flow module and select **Configure.**

   Warehouse Builder displays the Configuration Properties sheet for the process flow module.

2. Set the properties for **Evaluation Location** and **Identification Location.**

   Evaluation Location is the location from which this process flow is evaluated.

   Identification Location provides the location where the generated code will be deployed to.

To configure a process flow package:

1. Right-click the process flow package and select **Configure**.

   Warehouse Builder displays the Configuration Properties sheet for the process flow module.

2. Set the properties for **Referred Calendar** and **Generation Comments.**

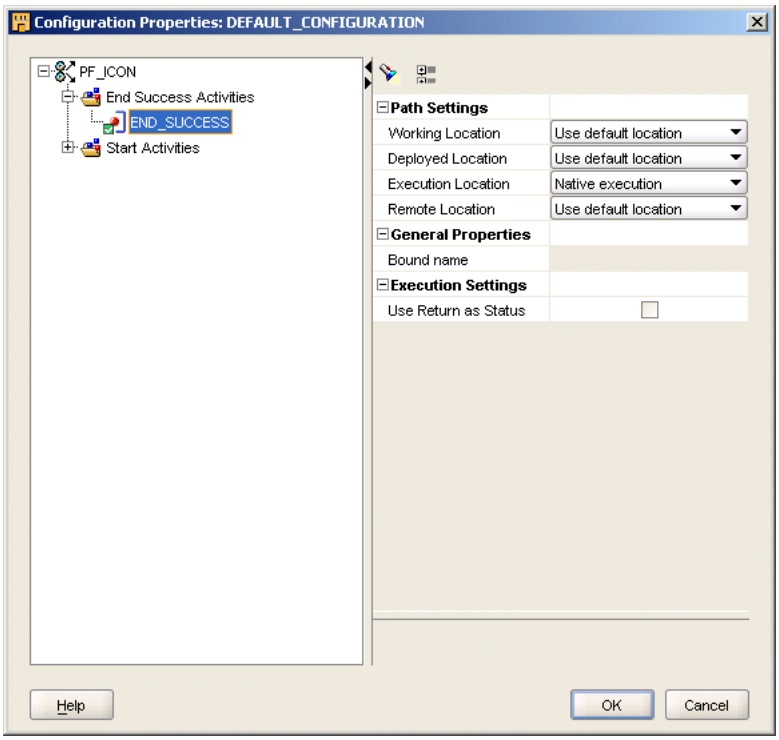
   Referred Calendar provides the schedule to associate with this package.

   Generation Comments provides additional comments for the generated code.

Click any of the activities of a package to view its properties.

Figure 24–3 displays the Configuration Properties of an activity.

Figure 24–3 Activities Configuration Properties Sheet



Under **Path Settings,** set the following properties for each activity in the process flow:

Execution Location: The location from which this activity is executed. If you configured Oracle Enterprise Manager, you can select an OEM agent to execute the process flow.

Remote Location: The remote location for FTP activities only.

Working Location: The working location for FTP, FILE EXISTS and External Process activities only.

Deployed Location: The deployment location. This setting applies to transformation activities only. For activities referring to pre-defined transformations, you must change the setting from **Use Default Location** and specify a valid location.

Under **General Properties,** you can view the bound name which is the name of the object that the activity represents in the process flow. Only mapping, transformation, and subprocess activities have bound names.

Under **Execution Settings,** select the option **Use Return as Status.**

This setting governs the behavior for activities that return NUMBER in their output. These activities include the FTP, User Defined, and Transform activities. When you select **Use Return as Status**, the Process Flow Editor assigns the outgoing transition conditions based on the following numerical return values for the activity:

1 = Success Transition

2 = Warning Transition

3 = Error Transition

25

Source and Target Operators

This chapter provides details on how to use operators as sources and targets in a mapping and includes the following topics:

Using Source and Target Operators

This section describes how to set properties for source and target operators, and also provides details on how to use each operator.

Defining a mapping involves the following general steps:

1. Creating a Mapping
2. Adding Operators
3. Editing Operators
4. Connecting Operators
5. Setting Mapping Properties
6. Setting Properties for Oracle Source and Target Operators
7. Configuring Mappings Reference
8. Debugging a Mapping

List of Source and Target Operators

The list of source and target operators are:

Using Oracle Source and Target Operators

Oracle source and target operators refer to operators that are bound to Oracle data objects in the Warehouse Builder repository. Use these operators in a mapping to load data into or source data from Oracle data objects.

Setting Properties for Oracle Source and Target Operators

The properties panel in the Mapping Editor displays the properties of the selected operator. When you select an object on the canvas, the editor displays its associated properties in the Properties panel along the left side. It contains the following categories of parameters for source and target operators:

- **Operator Name:** Under the operator name, you can set Primary Source, Target Load Order, and the Loading Type. Depending upon the type of target, you can set different values for the Loading Type as described in Loading Types for Oracle Target Operators and Loading Types for Flat Files.
- **Conditional Loading:** You can set Target Filter for Update, Target Filter for Delete, and Match By Constraint.
- **Keys (read-only):** You can view the Key Name, Key Type, and Referenced Keys. If the operator functions as a source, the key settings are used in conjunction with the join operator. If the operator functions as a target, the key settings are used in conjunction with the Match By Constraint parameter.
- **File Properties:** Under the file properties, you can view the Bound Name.
- **Error Table:** You can set the Error Table Name, Roll up Errors, and Select Only Errors from this Operator. This section of properties is displayed only for the following mapping operators: Table, View, Materialized View, External Table, and Dimension.

Primary Source

For Oracle Application Embedded Data Warehouse (EDW) users, refer to EDW documentation. For all other users, disregard this parameter.

Loading Types for Oracle Target Operators

Select a loading type for each target operator using the Loading Type property.

For all Oracle target operators, except for dimensions and cubes, select one of the following options.

- **CHECK/INSERT:** Warehouse Builder checks the target for existing rows. If there are no existing rows, Warehouse Builder inserts the incoming rows into the target.

- **DELETE:** Warehouse Builder uses the incoming row sets to determine which of the rows on the target to delete.
- **DELETE/INSERT:** Warehouse Builder deletes all rows in the target and then inserts the new rows.
- **INSERT:** Warehouse Builder inserts the incoming row sets into the target. Insert fails if a row already exists with the same primary or unique key.
- **INSERT/UPDATE:** For each incoming row, Warehouse Builder performs the insert operation first. If the insert fails, an update operation occurs. If there are no matching records for update, the insert is performed. If you select INSERT/UPDATE and the Default Operating Mode on page 24-3 is row based, you must set unique constraints on the target. If the operating mode is set based, Warehouse Builder generates a MERGE statement.
- **None:** Warehouse Builder performs no operation on the target. This setting is useful for testing. Extraction and transformations run but have no effect on the target.
- **TRUNCATE/INSERT:** Warehouse Builder truncates the target and then inserts the incoming row set. If you choose this option, Warehouse Builder cannot roll back the operation even if the execution of the mapping fails. Truncate permanently removes the data from the target.
- **UPDATE:** Warehouse Builder uses the incoming row sets to update existing rows in the target. If no rows exist for the specified match conditions, no changes are made.

  If you set the configuration property **PL/SQL Generation Mode** of the target module to Oracle 10*g* or Oracle 10*g*R2, Warehouse Builder updates the target in set based mode. The generated code includes a MERGE statement without an insert clause. For modules configured to generate 9*i* and earlier versions of PL/SQL code, Warehouse Builder updates the target in row based mode.
- **UPDATE/INSERT:** For each incoming row, Warehouse Builder performs the update first. If you select UPDATE/INSERT and the Default Operating Mode on page 24-3 for the target is set-based, Warehouse Builder generates a MERGE statement.

For dimensions and cubes, the Loading Type property has he following options: Load and Remove. Use load to load data into the dimension or cube. Use Remove to remove data from the dimension or cube.

Loading Types for Flat File Targets

Configure **SQL*Loader parameters** to define SQL*Loader options for your mapping. The values chosen during configuration directly affect the content of the generated SQL*Loader and the runtime control files. SQL*Loader provides two methods for loading data:

- **Conventional Path Load:** Executes an SQL INSERT statement to populate tables in an Oracle Database.
- **Direct Path Load:** Eliminates much of the Oracle Database overhead by formatting Oracle data blocks and writing the data blocks directly to the database files. Because a direct load does not compete with other users for database resources, it can usually load data at or near disk speed.

  Certain considerations such as restrictions, security, and backup implications are inherent to each method of access to database files. See Oracle9*i Database Utilities* for more information.

When designing and implementing a mapping that extracts data from a flat file using SQL*Loader, you can configure different properties affecting the generated SQL*Loader script. Each load operator in a mapping has an operator property called Loading Types. The value contained by this property affects how the SQL*Loader INTO TABLE clause for that load operator is generated. Although SQL*Loader can append, insert, replace, or truncate data, it cannot update any data during its processing. Table 25–1 lists the INTO TABLE clauses associated with each load type and their affect on data in the existing targets.

Table 25–1 Loading Types and INTO TABLE Relationship

Loading Types	INTO TABLE Clause	Affect on Target with Existing Data
INSERT/UPDATE	APPEND	Adds additional data to target.
DELETE/INSERT	REPLACE	Removes existing data and replaces with new (DELETE trigger fires).
TRUNCATE/INSERT	TRUNCATE	Removes existing data and replaces with new (DELETE trigger fires).
CHECK/INSERT	INSERT	Assumes target table is empty.
NONE	INSERT	Assumes target table is empty.

Target Load Order

This property enables you to specify the order in which multiple targets within the same mapping are loaded. Warehouse Builder determines a default load order based on the foreign key relationships. You can overrule the default order.

Target Filter for Update

If the condition evaluates to true, the row is included in the update loading operation.

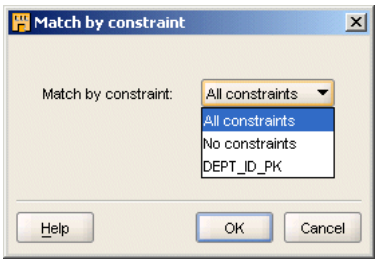
Target Filter for Delete

If evaluated to true, the row is included in the delete loading operation.

Match By Constraint

When loading target operators with the UPDATE or the DELETE conditions, you can specify matching criteria. You can set matching and loading criteria manually or choose from several built-in options. Use Match By Constraint to indicate whether unique or primary key information on a target overrides the manual matching and loading criteria set on its attributes. When you click the property Match By Constraint, Warehouse Builder displays a drop down list containing the constraints defined on that operator and the built-in loading options, as shown in Figure 25–1.

Figure 25–1 Match By Constraint Options for Operators



If you select **All Constraints**, all manual attribute load settings are overruled and the data is loaded as if the load and match properties of the target attributes were set as displayed in Table 25–2.

Table 25–2 All Constraints Target Load Settings

Load Setting	Key Attribute	All Other Attributes
Load Column when Updating Row	NO	YES
Match Column when Updating Row	YES	NO
Match Column when Deleting Row	YES	NO

If you select **No Constraints,** all manual load settings are honored and the data is loaded accordingly.

If you select a constraint previously defined for the operator, all manual attribute load settings are overruled and the data is loaded as if the load and match properties of the target were set as displayed in Table 25–4.

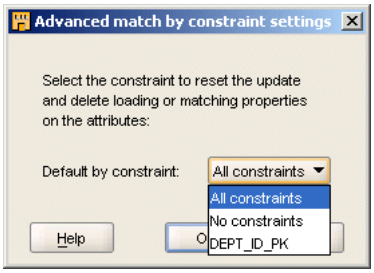
Table 25–3 Target Load Settings for a Selected Constraint

Load Setting	Selected Key Attributes	All Other Attributes
Load Column when Updating Row	NO	YES
Match Column when Updating Row	YES	NO
Match Column when Deleting Row	YES	NO

Reverting Constraints to Default Values

If you made changes at the attribute level and you want to default all settings, click **Advanced.** Warehouse Builder displays a drop down box list containing the loading options as shown in Figure 25–2. Warehouse Builder defaults the settings based on the constraint type you select.

Figure 25–2 Advanced Settings for Match By Constraint



For example, if you want to reset the match properties for all key attributes, click **Advanced,** select **No Constraints,** and click **OK.** Warehouse Builder overwrites the manual load settings and loads the data based on the settings displayed in Table 25–4.

Table 25–4 Default Load Settings for Advanced No Constraints

Load Setting	All Key Attributes	All Other Attributes
Load Column when Updating Row	YES	YES
Match Column when Updating Row	NO	NO

Table 25–4 (Cont.) Default Load Settings for Advanced No Constraints

Load Setting	All Key Attributes	All Other Attributes
Match Column when Deleting Row	NO	NO

Alternatively, if you click **Advanced** and select **All Constraints,** Warehouse Builder overwrites the manual load settings and loads the data based on the settings displayed in Table 25–5

Table 25–5 Default Load Settings for Advanced All Constraints

Load Setting	All Key Attributes	All Other Attributes
Load Column when Updating Row	NO	YES
Match Column when Updating Row	YES	NO
Match Column when Deleting Row	YES	NO

Bound Name

The name used by the code generator. If an operator is currently bound and synchronized, then this property is read-only. If an operator is not yet bound, you can edit the bound name within the Mapping Editor before you synchronize it to a repository object.

Key Name

Name of the primary, foreign, or unique key.

Key Columns

Local columns that define this key. Each key column is comma-delimited if the operator contains more than one key column.

Key Type

Type of key, either primary, foreign, or unique.

Referenced Keys

If the operator contains a foreign key, **Referenced Keys** displays the primary key or unique key for the referenced object.

Error Table Name

The name of the error table that stores the invalid records during a load operation.

Roll up Errors

Select **Yes** to roll up records selected from the error table by the error name. Thus all errors generated by a particular input record will be rolled up into a single record with the error names concatenated in the error name attribute.

Select Only Errors from this Operator

Rows selected from the error table will contain only errors created by this operator in this map execution

Setting Attribute Properties

For each attribute in a source and target operator, parameters are categorized into the following types:

- **Attribute Properties:** Under the attribute properties, you can view the Bound Name property.
- **Loading Properties:** The operators for tables, dimensions, cubes, views, and materialized views have a Loading Properties category. This category contains the following settings: Load Column When Inserting Row, Load Column When Updating Row, Match Column When Updating Row, Update: Operation, and Match Column When Deleting Row.
- **Data Type Information:** The data type properties are applicable to all operators. They include Data Type, Precision, Scale, Length, and Fractional Seconds Precision.

Bound Name

Name used by the code generator to identify this item. By default, it is the same name as the item. This is a read-only setting when the operator is bound.

Data Type

Data type of the attribute.

Precision

The maximum number of digits this attribute will have if the data type of this attribute is a number or a float. This is a read-only setting.

Scale

The number of digits to the right of the decimal point. This only applies to number attributes.

Length

The maximum length for a CHAR, VARCHAR, or VARCHAR2 attribute.

Fractional Seconds Precision

The number of digits in the fractional part of the datetime field. It can be a number between 0 and 9. This property is used only for TIMESTAMP data types.

Load Column When Inserting Row

This setting prevents data from moving to a target even though it is mapped to do so. If you select **Yes** (default), the data will reach the mapped target.

Load Column When Updating Row

This setting prevents the selected attribute data from moving to a target even though it is mapped to do so. If you select **Yes** (default), the data reaches the mapped target attribute. If all columns of a unique key are not mapped, then the unique key is not used to construct the match condition. If no columns of a unique key are mapped, Warehouse Builder displays an error. If a column (not a key column) is not mapped, then it is not used in loading.

Match Column When Updating Row

This setting updates a data target row only if there is a match between the source attribute and mapped target attribute. If a match is found, then an update occurs on the row. If you set this property to **Yes** (default), the attribute is used as a matching attribute. If you use this setting, then all the key columns must be mapped. If there is only one unique key defined on the target entity, use constraints to override this setting.

Update: Operation

You can specify an update operation to be performed when Warehouse Builder locates a matching row and updates the target. An update operation is performed on the target attribute using the data of the source attribute. Table 25–6 lists the update operations you can specify and describes the update operation logic.

Table 25–6 Update Operations

Operation	Example	Result If Source Value = 5 and Target Value = 10
=	TARGET = SOURCE	TARGET = 5
+=	TARGET = SOURCE + TARGET	TARGET = 15 (5 + 10)
-=	TARGET = TARGET - SOURCE	TARGET = 5 (10 - 5)
=-	TARGET = SOURCE - TARGET	TARGET = negative 5 (5 - 10)
||=	TARGET = TARGET || SOURCE	TARGET = 105 (10 concatenated with 5)
=||	TARGET = SOURCE || TARGET	TARGET = 510 (5 concatenated with 10)

Match Column When Deleting Row

Deletes a data target row only if there is a match between the source attribute and mapped target attribute. If a match is found, then a delete occurs on the row. If you set this property to **Yes** (default), the attribute is used as a matching attribute. Constraints can override this setting.

Constant Operator

The Constant operator enables you to define constant values. You can place Constants anywhere in any PL/SQL or ABAP mapping.

Note: For SQL*Loader mappings, use a Data Generator Operator instead.

The Constant operator produces a single output group that contains one or more constant attributes. Warehouse Builder initializes constant at the beginning of the execution of the mapping.

For example, use a constant operator to load the value of the current system date into a table operator. In the Expression Builder, select the public transformation `SYSDATE` from the list of pre-defined transformations. For more information on public transformations, see Chapter 9, "Using Oracle Warehouse Builder Transformations".

To define a constant operator in a PL/SQL or ABAP mapping:

1. Drop a Constant operator onto the Mapping Editor canvas.
2. Right-click the Constant operator and select **Open Editor**.

   The Constant Editor dialog is displayed.

3. On the Output tab, click the **Add** button to add an output attribute.

   You can modify the name and the data type of the attribute.

4. Click **OK** to close the Constant Editor dialog.

5. In the Mapping Editor, select an output attribute on the Constant operator.

   The Properties panel of the Mapping Editor displays the properties of the output attribute.

6. Click the Ellipsis button to the right of the **Expression** field.

   The Expression Builder dialog is displayed. Use this dialog to write an expression for the constant.

   The length, precision, and scale properties assigned to the output attribute must match the values returned by the expressions defined in the mapping. For VARCHAR, CHAR, or VARCHAR2 data types, enclose constant string literals within single quotes, such as, `'my_string'`.

Construct Object Operator

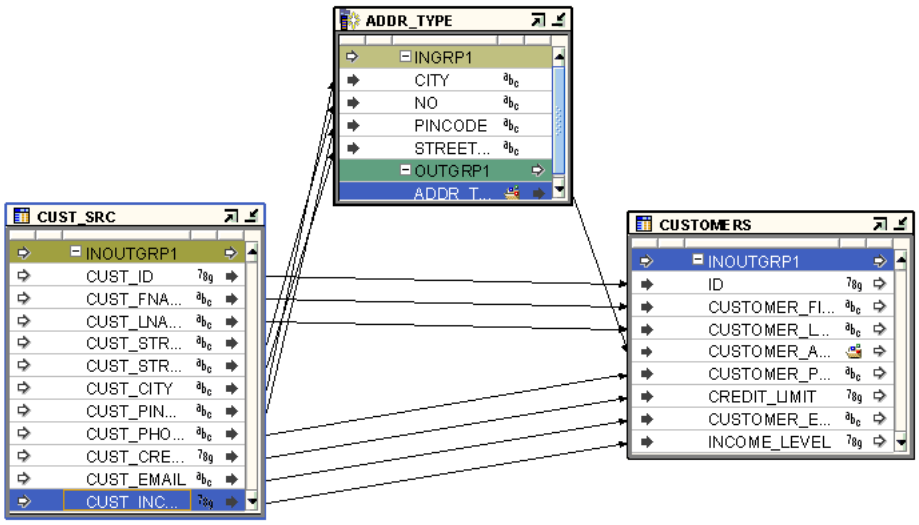
The Construct Object operator enables you to create SQL object data types (object types and collection types), PL/SQL object types, and cursors in a mapping by using the individual attributes that they comprise.

For example, you can use a Construct Object operator to create a SQL object type that is used to load data into a table that contains a column whose data type is an object type. You can also use this operator to create the payload that loads data into an advanced queue. This operator also enables you to construct a SYS.REFCURSOR object.

The Construct Object operator has one input group and one output group. The input group represents the individual attributes that comprise the object type. The output of the Construct Object operator is an object type that is created using the individual attributes. In a mapping, the data type of the output attribute of the Construct Object operator should match the target attribute to which it is being mapped.

Figure 25–3 displays a mapping that uses a Construct Object operator. The source table `CUST_SRC` uses separate attributes to store each component of the customer address. But the target table `CUSTOMERS` uses an object type to store the customer address. To load data from the `CUST_SRC` table into the `CUSTOMERS` table, the customer address should to an object type whose signature matches that of the customer address in `CUSTOMERS`. The Construct Object operator takes the individual attributes, from `CUSTOMERS_SRC`, that store the customer address as input and constructs an object type. The Construct Object operator is bound to the user-defined data type `CUST_ADDR` stored in the Warehouse Builder repository.

Figure 25–3 Construct Object Operator in a Mapping



To define a Construct Object operator in a mapping:

1. Drag and drop a Construct Object operator onto the Mapping Editor canvas.
2. Use the Add Construct Object dialog to create or select an object. For more information on these options, see Adding Operators that Bind to Repository Objects on page 6-12.
3. Map the individual source attributes that are used to construct the object to the input group of the Construct Object operator.
4. Map the output attribute of the Construct Object operator to the target attribute. the data type of the target attribute should be an object type.

   Note that the signatures of the output attribute of the Construct Object operator and the target attribute should be the same.

Cube Operator

You use the Cube operator to source data from or load data into cubes.

The Cube operator contains a group with the same name as the cube. This group contains an attribute for each of the cube measures. It also contains the attributes for the surrogate identifier and business identifier of each dimension level that the cube references. Additionally, the Cube operator displays one group for each dimension that the cube references.

You can bind a Cube operator to a cube defined in any Oracle module in the current project. You can also synchronize the cube operator and update it with changes made to the cube to which it is bound. To synchronize a Cube operator, right-click the Cube operator on the Mapping Editor canvas and select **Synchronize**.

To create a mapping that contains a Cube operator:

1. Drag and drop a Cube operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Cube dialog.
2. Use the Add Cube dialog to create or select a cube. For more information on these options, see "Adding Operators that Bind to Repository Objects" on page 6-12.

Alternatively, you can perform steps 1 and 2 as one step. In the Mapping Editor, navigate to the Selection Tree tab of the Explorer window. Select the cube and drag and drop it onto the Mapping Editor canvas.

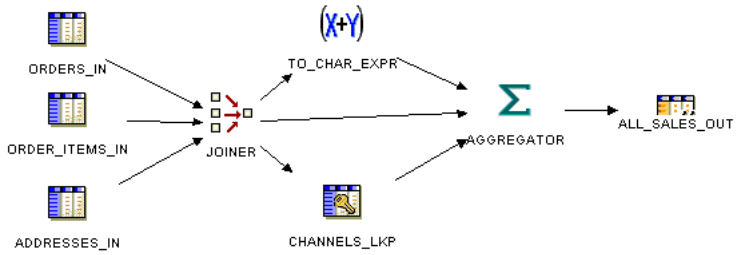
3. Map the attributes from the Cube operator to the target or map attributes from the source operator to the Cube operator.

When you load a cube, you map the data flow from the source to the attribute that represents the business identifier of the referencing level. Warehouse Builder performs a lookup and then stores the corresponding surrogate ID in the cube table. Fro example, when you map the attributes from the dimension operator to the cube operator, Warehouse Builder creates a Key Lookup operator in cases where it is needed to lookup the surrogate identifier of the dimension.

Note that if there is a possibility of the lookup condition returning multiple rows, you must ensure that only one row is selected out of the returned rows. You can do this by using the Deduplicator operator or Filter operator.

Figure 25–4 displays a mapping that uses the Cube operator as a target. Data from three source tables is joined using a Joiner operator. An Aggregator operator is used to aggregate the joined data with the data from another source table. The output of the Aggregator operator is mapped to the Cube operator.

Figure 25–4 Mapping that Loads a Cube



Cube Operator Properties

The cube operator has the following properties that you can use to load a cube.

- **Target Load Order:** This property determines the order in which multiple targets within the same mapping are loaded. Warehouse Builder determines a default order based on the foreign key relationships. You can use this property to overrule the default order.
- **Solve the Cube:** Select YES for this property to aggregate the cube data while loading the cube. This increases the load time, but decreases the query time. The data is first loaded and then aggregated.
- **Incremental Aggregation:** Select this option to perform incremental loads. This means that if the cube has been solved earlier, subsequent loads will only aggregate the new data.
- **AW Staged Load:** If true, the set-based AW load data is staged into a temporary table before loading into the AW.
- **AW Truncate Before Load:** Indicates whether all existing cube values should be truncated before loading the cube. Setting this property to YES truncates existing cube data.

Data Generator Operator

Use a single Data Generator operator to introduce constants or sequences into a SQL*Loader mapping. Each SQL*Loader mapping can contain a maximum of one Data Generator operator.

> **Recommendation:** For PL/SQL mappings use a Constant Operator or Sequence Operator instead of a Data Generator.

For mappings with flat file sources and targets, the Data Generator operator connects the mapping to SQL*Loader to generate the data stored in the database record.

The following functions are available:

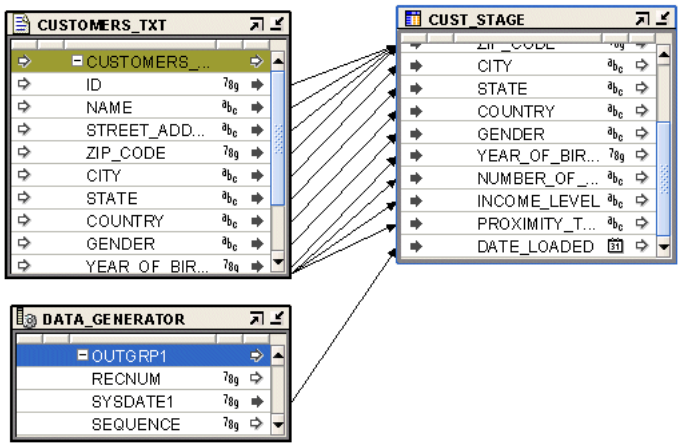
- RECNUM
- SYSDATE
- SEQUENCE

Warehouse Builder can generate data by specifying only sequences, record numbers, system dates, and constants as field specifications. SQL*Loader inserts as many records as are specified by the LOAD keyword.

The Data Generator operator has one output group with predefined attributes corresponding to Record Number, System Date, and a typical Sequence. You can create new attributes but not alter the predefined attributes.

Figure 25–5 shows a mapping that uses the Data Generator operator to obtain the current system date. The data from a flat file CUSTOMERS_TXT is loaded in to a staging table CUST_STAGE. The staging table contains an additional attribute for the date the data was loaded. The SYSDATE attribute of the Data Generator operator is mapped to the DATE_LOADED attribute of the staging table CUST_STAGE.

Figure 25–5 Data Generator in a Mapping



To define a Data Generator in a SQL *Loader mapping:

1. Drop a Data Generator operator onto the Mapping Editor canvas.
2. Right-click the operator and select **Open Details**.

   The DATA_GENERATOR Editor is displayed.

3. Select the Output Attributes tab of the DATA_GENERATOR Editor.

   Warehouse Builder displays the pre-defined output attributes RECNUM, SYSDATE1, and SEQUENCE.

4. On the Output Attributes tab, define the properties and type an optional description for the predefined output attributes.

5. Click **OK** to close the DATA_GENERATOR Editor.

6. On the operator in the mapping canvas, select the RECNUM attribute.

   Warehouse Builder displays the properties of this attribute in the Properties panel of the Mapping Editor.

7. In the Expression field, click the Ellipsis button to open the Expression Builder and define an expression.

8. Repeat steps 6 and 7 for the SEQUENCE attribute.

Setting a Column to the Data File Record Number

Use the RECNUM keyword to set an attribute to the number of the records that the record was loaded from. Records are counted sequentially from the beginning of the first data file, starting with record 1. RECNUM increments as each logical record is assembled. It increments for records that are discarded, skipped, rejected, or loaded. For example, if you use the option SKIP=10, the first record loaded has a RECNUM of 11.

Setting a Column to the Current Date

A column specified with SYSDATE gets the current system date, as defined by the SQL language SYSDATE function.

The target column must be of type CHAR or DATE. If the column is of type CHAR, the date is loaded in the format dd-mon-yy. If the system date is loaded into a DATE column, then you can access it in the time format and the date format. A new system date/time is used for each array of records inserted in a conventional path load and for each block of records loaded during a direct path load.

Setting a Column to a Unique Sequence Number

The SEQUENCE keyword ensures a unique value for a column. SEQUENCE increments for each record that is loaded or rejected. It does not increment for records that are discarded or skipped.

The combination of column name and the SEQUENCE function is a complete column specification. Table 25–7 lists the options available for sequence values.

Table 25–7 Sequence Value Options

Value	Description
column_name	The name of the column in the database to which the sequence is assigned.
SEQUENCE	Specifies the value for a column.
integer	Specifies the beginning sequence number.
COUNT	The sequence starts with the number of records already in the table plus the increment.
MAX	The sequence starts with the current maximum value for the column plus the increment.

Table 25–7 (Cont.) Sequence Value Options

Value	Description
incr	The value that the sequence number is to increment after a record is loaded or rejected.

If records are rejected during loading, Warehouse Builder preserves the sequence of inserts despite data errors. For example, if four rows are assigned sequence numbers 10, 12, 14, and 16 in a column, and the row with 12 is rejected, Warehouse Builder inserts the valid rows with assigned numbers 10, 14, and 16, not 10, 12, 14. When you correct the rejected data and reinsert it, you can manually set the columns to match the sequence.

Dimension Operator

You use the Dimension operator to source data from or load data into dimensions and slowly changing dimensions.

The Dimension operator contains one group for each level in the dimension. The groups use the same name as the dimension levels. The level attributes of each level are listed under the group that represents the level.

You cannot map a data flow to the surrogate identifier attribute or the parent surrogate identifier reference attribute of any dimension level. Warehouse Builder automatically populates these columns when it loads a dimension.

You can bind and synchronize a Dimension operator with a dimension stored in the repository. To avoid errors in the generated code, ensure that the repository dimension is deployed successfully before you deploy the mapping that contains the Dimension operator. To synchronize a Dimension operator with the repository dimension, right-click the dimension on the Mapping Editor canvas and select **Synchronize**.

To use a Dimension operator in a mapping:

1. Drag and drop a Dimension operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Dimension dialog.

2. Use the Add Dimension dialog to select a dimension.

   Alternately, you can combine steps 1 and 2 into one single step. In the Mapping Editor, navigate to the Selection Tree tab of the Explorer window. Select the dimension and drag and drop it onto the Mapping Editor canvas.

3. Map the attributes from the Dimension operator to the target or map attributes from the source to the Dimension operator.

Dimension Operator Properties

Use the Properties panel of the Mapping Editor to set options that define additional details about loading or removing data from a dimension or slowly changing dimension.

You can set properties at the following three levels: operator, group that represents each level in the dimension, and level attribute.

The Dimension operator has the following properties.

Loading Type Represents the type of operation to be performed on the dimension. The options you can select are as follows:

- **LOAD:** Select this value to load data into the dimension or slowly changing dimension.
- **REMOVE:** Select this value to delete data from the dimension or slowly changing dimension.

While loading or removing data, Warehouse Builder performs a lookup to determine if the source record exists in the dimension. The matching is performed by the natural key identifier. If the record exists, a REMOVE operation removes existing data. A LOAD operation updates existing data and then loads new data.

Note that when you remove a parent record, the child records will have references to a non-existent parent.

Target Load Order Specifies the order in which multiple targets within the same mapping are loaded. Warehouse Builder determines a default order based on the foreign key relationships. Use this property to overrule the default order.

Default Effective Time of Open Record This property is applicable for Type 2 SCDs only. It represents the default value set for the effective time of the current record. In cases where data is loaded into the Type 2 SCD on a daily basis, you can set this property to SYSDATE.

Default Expiration Time of Open Record This property is applicable for Type 2 SCDs only. It represents a date value that is used as the expiration time of a newly created open record for all the levels in the dimension. The default value is NULL.

Note that, even when you set the value of this property, Warehouse Builder ignores this setting. You must set the Default Expiration Time of Open Record for each level in the Type 2 Slowly Changing Dimension. This property will be implemented in future releases.

Type 2 Extract/Remove Current Only This property is applicable only for Type 2 SCDs. Use this property to specify which records are to be extracted or removed. You can set the following values for this property:

- **YES:** When you are extracting data from the Type 2 SCD, only the current record is extracted. When you are removing data from a Type 2 SCD, only the current record is closed (expiration date is set either to SYSDATE or to the date defined in the Default Expiration Time of Open Record property).

  Note that in the case of a Type 2 SCD that uses a snowflake implementation, you cannot remove a record if it has child records that have a Type 2 trigger.
- **NO:** When you are extracting data from a Type 2 SCD, all the records, including historical records, that match the natural identifier are extracted from the dimension.

  When you are removing data from the Type 2 SCD, all records, including historical records, that match the natural identifier are closed. However, the records will not be removed from the dimension. Any child records of the closed record will remain linked to a closed parent.

AW Truncate Before Load This property is applicable for MOLAP dimensions only. It indicates whether all existing dimension data should be truncated before loading fresh data. Set this property to YES to truncate any existing dimension data before you load fresh data.

AW Staged Load This property is applicable for MOLAP dimensions only. Select this option to stage the set-based load data into a temporary table before loading into the analytic workspace.

Each group in the Dimension operator represents a dimension level. You can set the following properties for each dimension level:

- **Extracting Type:** Represents the extraction operation to be performed when the dimension is used as a source. Select **Extract Current Only (Type 2 Only)** to extract current records only from a Type 2 SCD. This property is valid only for Type 2 SCDs. Select **Extract All** to extract all records from the dimension or SCD.
- **Default Expiration Time of Open Record:** This property is applicable for Type 2 SCDs only. It represents a date value that is used as the expiration time of a newly created open record. The default value is NULL.

Dimension Operator as a Source

You can source data stored in a repository dimension or SCD by using a Dimension operator in a mapping.

Sourcing Data From Dimensions To source data stored in a dimension, create a mapping that contains a Dimension operator. Ensure that the Dimension operator is bound to the repository dimension that contains the source data. Then map the attributes from the dimension levels to the target operator. See Figure 25–4 for and example of a mapping that sources data from a dimension.

Sourcing Data From Type 2 Slowly Changing Dimensions The Dimension operator enables you to source data stored in a Type 2 SCD. Since a Type 2 SCD stores multiple versions of a single record, you must specify which version of the record you want to use.

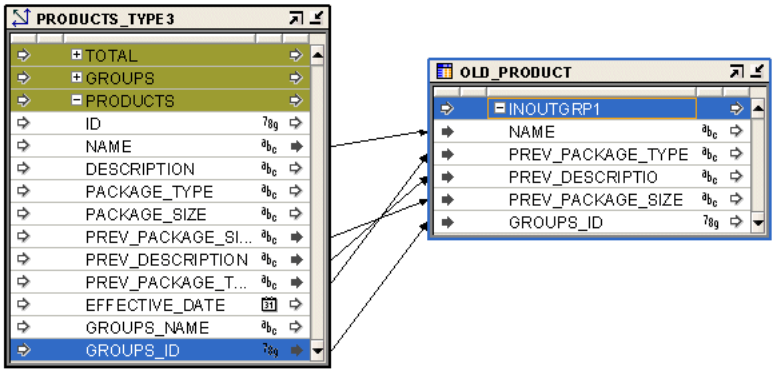
Use the **Extracting Type** property of the group that represents each dimension level to specify the version of the level record from which you want to source data. To load all versions of a particular level record select **Extract All**. To retrieve only the latest version of a level record, set the Extracting Type property to **Extract Current Only (Type 2 Only)**.

To set the properties of a dimension level, select the group that represents the level in the Dimension operator. The Group Properties panel displays the properties of the level.

Sourcing Data From Type 3 Slowly Changing Dimensions To source data from a Type 3 SCD, create a mapping that contains a Dimension operator. Bind the Dimension operator to the Type 3 SCD in the repository from which you want to source data. A Type 3 SCD uses separate attributes to store historic values of versioned attributes. Thus, to source historic data, map the attributes that represent the previous values of versioned attributes to the target. To source current data, map the attributes that store the level attributes to the target.

Figure 25–6 displays a mapping that sources the historic data records from the `PRODUCTS` Type 3 dimension. In this example, to source historic data, use the `PREV_DESCRIPTION`, `PREV_PACKAGE_TYPE`, or `PREV_PACKAGE_SIZE` attributes. To source current data, use `DESCRIPTION`, `PACKAGE_TYPE`, or `PACKAGE_SIZE`.

Figure 25–6 Mapping that Sources Data from a Type 3 SCD



Dimension Operator as a Target

You use a Dimension operator as a target in a mapping to load data into dimensions and SCDs. Define a data flow from the operators that represent the source objects to the dimension or SCD.

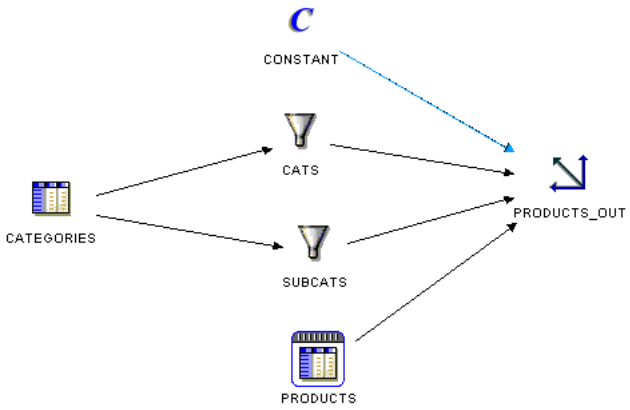
Warehouse Builder loads data into the dimension starting from the highest level.

> **Note:** You cannot map a data flow to the surrogate identifier or the parent surrogate identifier reference of a level.

Loading Data Into Dimensions Before loading records into a dimension, Warehouse Builder checks if a record with the same business identifier as the one being loaded exists in the dimension. If a similar record does not exist, the record from the source is added to the dimension. If a similar record exists, Warehouse Builder updates the current record to reflect the new attribute values.

Figure 25–7 displays a mapping that loads data into the `PRODUCTS` dimension, represented by the operator `PRODUCTS_OUT`. The source data is stored in two tables, `CATEGORIES` and `PRODUCTS`. The `CATEGORIES` table stores both the category and subcategory information. So the data from this table is filtered using two Filter operators `CATS` and `SUBCATS`. The filtered data is then mapped to the `CATEGORIES` level and the `SUBCATEGORIES` dimension levels. The `TOTAL` level of the dimension is loaded using a Constant operator. The data from the `PRODUCTS` table is mapped directly to the `PRODUCTS` level in the dimension.

Figure 25–7 Loading the Products Dimension



When you define a data flow to load a dimension, Warehouse Builder creates an in-line pluggable mapping that actually loads data into the dimension. To view this pluggable mapping, select the Dimension operator on the Mapping Editor canvas and click the Visit Child Graph icon on the graphical toolbar.

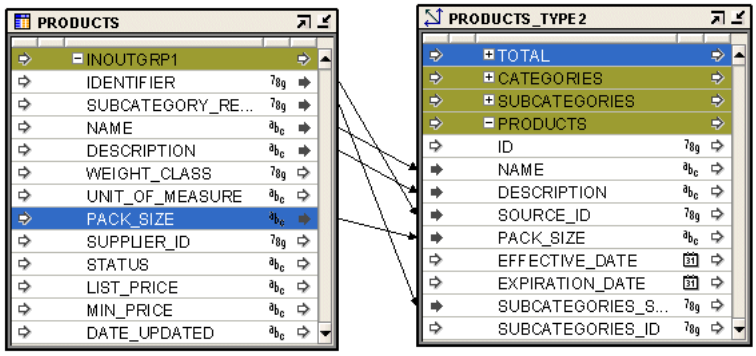
Loading Data Into Type 2 SCDs Before loading records into the Type 2 SCD, Warehouse Builder first checks if a record with the same business identifier exists in the Type 2 SCD. If the record does not exist, it is added to the Type 2 SCD. If the record already exists, Warehouse Builder performs the following steps:

- Marks the existing record as closed by setting the value specified in the property **Default Expiration Time of Open Record**.
- Creates a new record using the changed attribute values. If the effective date input for the level is not mapped, the effective time and expiration time are set using the **Default Effective Time of Current Record** and the **Default Expiration Time of Current Record** properties of the operator. If the effective date input for the level is mapped, then the effective time of the new record is set to the value that is obtained from the effective date input data flow. The effective date input, if connected, represents the actual effective date of each individual new record.

  The expiration date of the old record is set using the Default Expiration Time of Current Record property, regardless of the input connections

To load data into a slowly changing dimension, you map the source attributes to the attributes in the Type 2 SCD. Figure 25–8 displays a mapping that loads data into the PRODUCTS level of a Type 2 SCD.

Figure 25–8 Loading Data Into a Type 2 SCD



In this mapping, the effective time of a record is loaded from the source. You can also choose to set this to a default value, such as SYSDATE, using the **Default Effective Time of Current Record** property.

In addition to this, you need to store the closure date of historical records. In the mapping shown in Figure 25–8, the expiration time attribute is not loaded from the source. It is set to a default value using the **Default Expiration Time of Current Record** property.

To set default values for the expiration time or effective time for a level record:

1. In the Dimension operator, select the group that represents the level for which default values for the effective time and expiration time attributes should be set. For example, in the mapping shown in Figure 25–8, select the PRODUCTS group on the Dimension operator.

   The Group properties panel displays the properties of the selected level.

2. Use **Default Effective Time of Current Record** to specify the value to be stored as the effective time of the record being loaded.

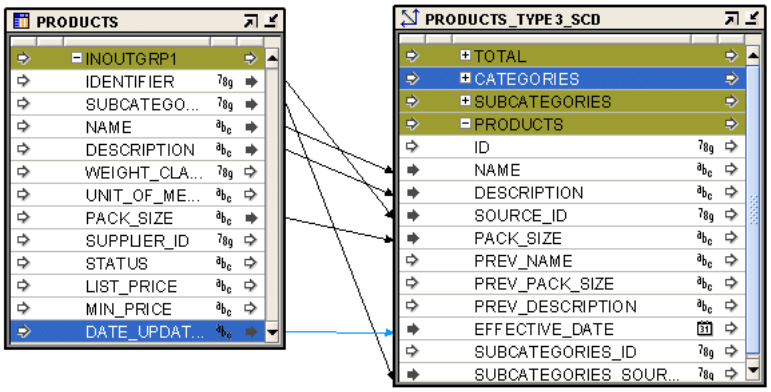
3. Use **Default Expiration Time of Current Record** to specify the value to be stored as the effective time of the record being loaded.

Loading Data Into Type 3 SCDs Before loading data into a Type 3 SCD, Warehouse Builder checks if a record with the same business identifier exists in the Type 3 SCD. If the record does not exist, it is added. If the record already exists, Warehouse Builder performs the following steps:

- Moves the values of the versioned attributes to the attributes that store the previous values of versioned attributes.
- Updates the record with the values from the source record.

Figure 25–9 displays a mapping that loads data into the PRODUCTS level of the Type 3 SCD. In this mapping, the effective time of the current record is loaded from the source. You can also use the **Default Effective Time of Current Record** property to set a default value for effective time of a level record.

Figure 25–9 Loading a Type 3 SCD



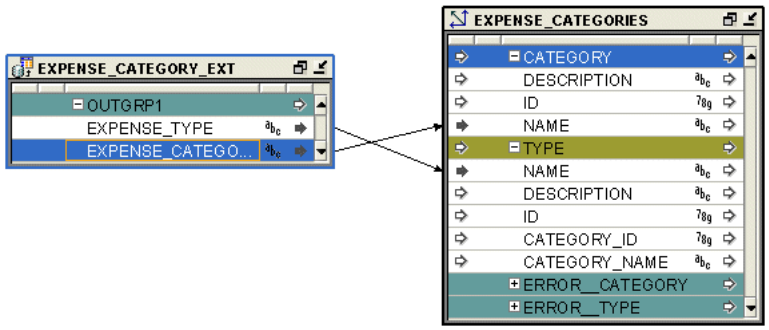
You cannot map a data flow to the attributes that represent the previous values of versioned attributes. For example, in the mapping shown in Figure 25–9, you cannot map an attribute to the `PREV_PACK_SIZE` and `PREV_DESCRIPTION` attributes of the `PRODUCTS` level.

External Table Operator

The External Table operator enables you to source data stored in external tables in the Warehouse Builder repository. You can then load the external table data into another repository object or perform transformations on the data. For example, you can source data stored in an external table, transform the data using mapping operators, and then load the data into a dimension or a cube.

Figure 25–10 displays a mapping that uses the External Table operator. The External Table operator `EXPENSE_CATEGORY_EXT` is bound to the external table of the same name in the repository. The data stored in this external table is used to load the dimension `EXPENSE_CATEGORIES`.

Figure 25–10 External Table Operator in a Mapping



To create a mapping that contains an External Table operator:

1. Drag and drop an External Table operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add External Table dialog.

2. Use the Add External Table dialog to create or select an external table. For more information on these options, see Adding Operators that Bind to Repository Objects on page 6-12.

3. Map the attributes from the output group of the External Table operator to the target operator or the intermediate transformation operator.

Expand Object Operator

The Expand Object operator enables you to expand an object type and obtain the individual attributes that comprise the object type.

You can bind and synchronize an Expand Object operator with a repository object type. To avoid generation errors in the mapping, ensure that you deploy the repository object type before you deploy the mapping.

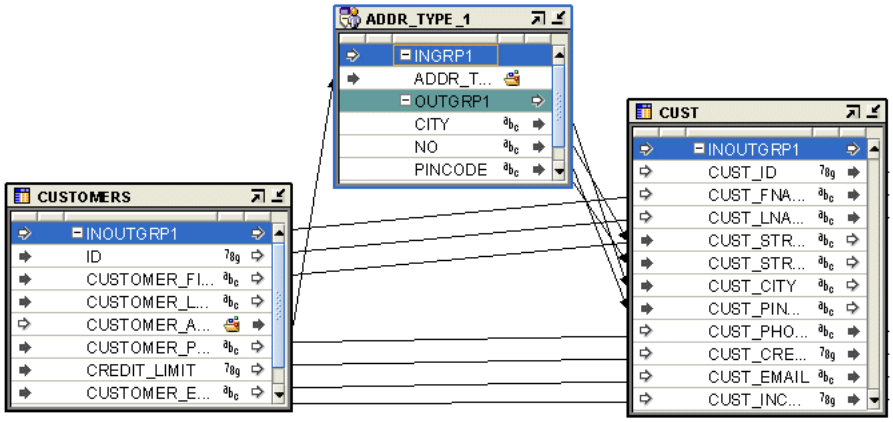
The Expand Object operator has one input group and one output group. The input group represents the object type that you want to expand in order to obtain its individual attributes. When you bind an Expand Object operator to a repository object, the output group of the operator contains the individual attributes that comprise the object type.

To successfully deploy a mapping that contains an Expand Object operator, ensure the following conditions are satisfied.

- The schema that contains the source tables must be on the same instance as the warehouse schema.
- The warehouse schema is granted the SELECT privilege on the source tables.
- The warehouse schema is granted the EXECUTE privilege on all the object types and nested tables used in the Expand Object operator.

Figure 25–11 displays a mapping that uses an Expand Object operator. The source table CUSTOMERS contains a column CUSTOMER_ADDRESS of data type ADDR_TYPE, a SQL object type. But the target table CUST contains four different columns, of Oracle built-in data types, that store each component of the customer address. To obtain the individual attributes of the column CUSTOMER_ADDRESS, create an Expand Object operator that is bound to the object type ADDR_TYPE. You then map the CUSTOMER_ADDRESS column to the input group of an Expand Object operator. The output group of the Expand Object operator contains the individual attributes of the column CUSTOMER_ADDRESS. Map these output attributes to the target operator.

Figure 25–11 Expand Operator in a Mapping



To define an Expand Object operator in a mapping:

1. Drag and drop an Expand Object operator onto the Mapping Editor canvas.
2. Use the Add Expand Object dialog to create or select an object. For more information on these options, see Adding Operators that Bind to Repository Objects on page 6-12.
3. Map the source attribute that needs to be expanded to the input group of the Expand Object operator.

   Note that the signature of the input object type should be same as that of the Expand Object operator.
4. Map the output attributes of the Expand Object operator to the target attributes.

Mapping Input Parameter Operator

You can introduce information external to Warehouse Builder as input into a mapping using a Mapping Input Parameter.

For example, you can use a Mapping Input Parameter operator to pass SYSDATE to a mapping that loads data to a staging area. Use the same Mapping Input Parameter to pass the timestamp to another mapping that loads the data to a target.

When you a generate mapping, Warehouse Builder creates a PL/SQL package. Mapping input parameters become part of the signature of the main procedure in the package.

The Mapping Input Parameter has a cardinality of one. It creates a single row set that can be combined with another row set as input to the next operator.

The names of the input attributes become the names of the mapping output parameters. The parameters can be used by connecting the attributes of the Mapping Input Parameters operator within the mapping editor. You can have only one Mapping Input Parameter in a mapping.

When you define the Mapping Input Parameter, you specify a data type and an optional default value.

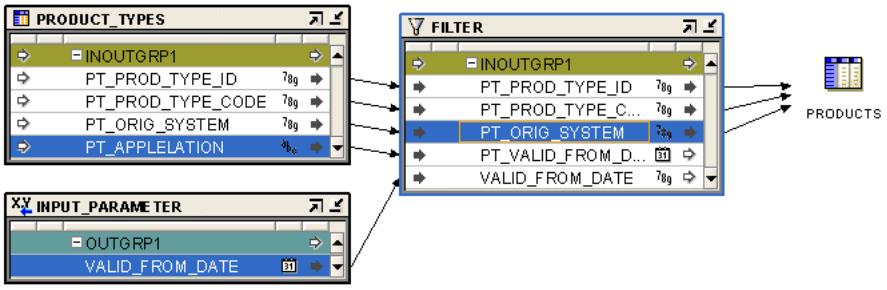
To define a Mapping Input Parameter operator in a mapping:

1. Drag and drop a Mapping Input Parameter operator onto the Mapping Editor canvas.
2. Right-click the Mapping Input Parameter operator and select **Open Details**.

   The INPUT_PARAMETER Editor is displayed.
3. Select the Output tab and click **Add** to add output attributes.

   You can rename the attributes and define the data type and other attribute properties.
4. Click **OK** to close the INPUT_PARAMETER Editor.
5. Connect the output attribute of the Mapping Input Parameter operator to an attribute in the target operator as shown in Figure 25–12.

Figure 25–12 Mapping Editor Showing A Mapping Input Parameter



Mapping Output Parameter Operator

Use a single Mapping Output Parameter operator to send values out of a PL/SQL mapping to applications external to Warehouse Builder.

A Mapping Output Parameter operator is not valid for a SQL*Loader mapping. When you generate mapping, Warehouse Builder creates a PL/SQL package. Mapping Output Parameters become part of the signature of the main procedure in the package.

The Mapping Output Parameter operator has only one input group and no output groups. You can have only one Mapping Output Parameter operator in a mapping. Only attributes that are not associated with a row set can be mapped into a Mapping Output Parameter operator. For example, constant, input parameter, output from a pre-mapping process, or output from a post process can all contain attributes that are not associated with a row set.

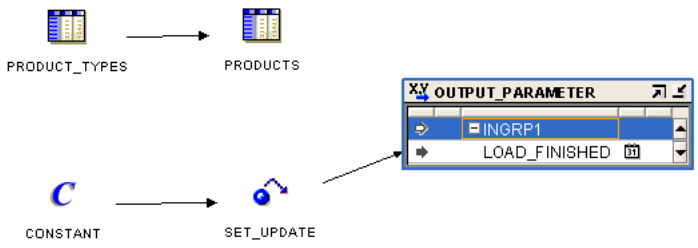
To define a Mapping Output Parameter operator in a mapping:

1. Drag and drop a Mapping Output Parameter operator onto the Mapping Editor canvas.
2. Right-click the Mapping Output Parameter operator and select **Edit.**
3. Select the Input Attributes tab and click **Add** to add input attributes.

   You can rename the attributes and define the data type and other attribute properties.

   Figure 25–13 displays an example of a Mapping Output Parameter operator used in a mapping.

Figure 25–13 Mapping Editor Showing An Output Parameter Operator



Materialized View Operator

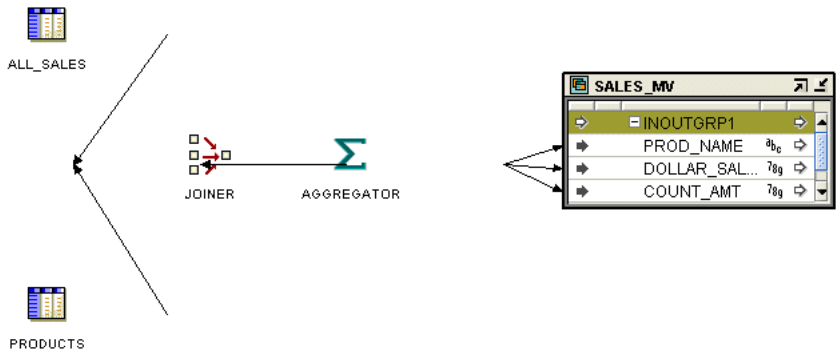
The Materialized View operator enables you to source data from or load data into a materialized view stored in the Warehouse Builder repository.

For example, you can use the data stored in a materialized view to load a cube. The Materialized View operator has one Input/Output group called INOUTGRP1. You cannot add additional groups to this operator, but you can add attributes to the existing Input/Output group.

You can bind and synchronize a Materialized View operator to a repository table. The repository materialized view must be deployed before the mapping that contains the Materialized View operator is generated to avoid errors in the generated code package.

Figure 25–14 displays a mapping that uses a Materialized View operator. The data from the two source tables PRODUCTS and ALL_SALES is joined using a Joiner operator. This data is then aggregated using an Aggregator operator. The aggregated data is used to load the materialized view SALES_MV.

Figure 25–14 Mapping that Contains a Materialized View Operator



To create a mapping that contains a Materialized View operator:

1. Drag and drop a Materialized View operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Materialized View dialog.

2. Use the Add Materialized View dialog to create or select a materialized view. For more information on these options, see "Adding Operators that Bind to Repository Objects" on page 6-12.

3. Map the attributes of the Materialized View operator.

   If you are using the materialized view operator as a target, connect the source attributes to the Materialized View operator attributes. If you are using the materialized view as a source, connect the Materialized View operator attributes to the target.

Sequence Operator

A Sequence operator generates sequential numbers that increment for each row.

For example, you can use the Sequence operator to create surrogate keys while loading data into a dimension table. You can connect a Sequence to a target operator input or to the inputs of other types of operators. You can combine the sequence outputs with outputs from other operators.

Because sequence numbers are generated independently of tables, the same sequence can be used for multiple tables. Sequence numbers may not be consecutive because the same sequence can be used by multiple sessions.

This operator contains an output group containing the following output attributes:

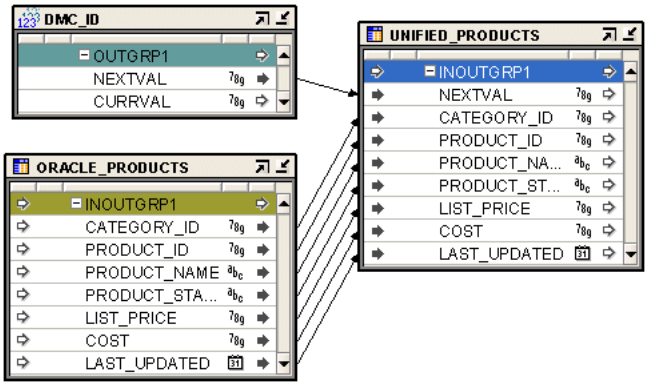
- **CURRVAL:** Generates from the current value.
- **NEXTVAL:** Generates a row set of consecutively incremented numbers beginning with the next value.

You can bind and synchronize Sequences to a repository sequence in one of the modules. The repository sequence must be generated and deployed before the mapping containing the Sequence is deployed to avoid errors in the generated code package. See "Adding Operators that Bind to Repository Objects" on page 6-12 for more information.

Generate mappings with sequences using row based mode. Sequences are incremented even if rows are not selected. If you want a sequence to start from the last number, then do not run your SQL package in set based or in set based with failover operating modes. See "Runtime Parameters" on page 24-3 for more information on configuring mode settings.

Figure 25–15 shows a mapping that uses a Sequence operator to automatically generate the primary key of a table. The NEXTVAL attribute of the Sequence operator is mapped to an input attribute of the target. The other input attributes from the source table are mapped directly to the target.

Figure 25–15 Mapping Sequence Operator in a Mapping



To define a Mapping Sequence operator in a mapping:

1. Drag and drop the Sequence operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Sequence dialog.

2. Use the Add Sequence dialog to create or select a sequence. For more information on these options, see "Adding Operators that Bind to Repository Objects" on page 6-12.

3. Connect the required output attribute from the Sequence operator to a target attribute.

Table Operator

The Table operator enables you to source data from and load data into tables stored in the Warehouse Builder repository.

You can bind and synchronize a Table operator to a repository table. The repository table must be deployed before the mapping that contains the Table operator is generated to avoid errors in the generated code package.

Figure 25–15 displays a mapping that uses Table operators as both source and target. Figure 25–5 displays a mapping that uses the Table operator as a target.

To define a Table operator in a mapping:

1. Drag and drop a Table operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Table dialog.

2. Use the Add Table dialog to create or select a table. For more information on these options, see "Adding Operators that Bind to Repository Objects" on page 6-12.

3. Map the attributes of the Table operator.

   If you are using the table as a target, connect the source attributes to the Table operator attributes. If you are using the table as a source, connect the Table operator attributes to the target.

Varray Iterator Operator

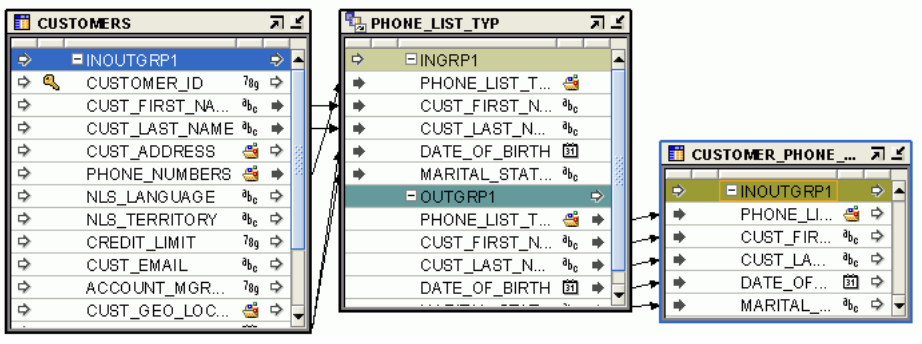
When you have an object of type nested table or varray, you can use the Varray Iterator operator to iterate through the values in the table type.

This operator accepts a table type attribute as the source and generates a value that is of the base element type defined in the nested table or varray type. If the operator is bound, reconciliation operations are performed on the operator. Reconciliation operations are not supported for unbound Varray Iterator operators.

You can create the Varray Iterator operator as either a bound operator or an unbound operator. You cannot synchronize or validate an unbound Varray Iterator operator. It has only one input group and one output group. You can have an input group with attributes of other data types. However, there must be at least one table type attribute.

The attributes of the output group are a copy of the attributes of the input group. The only difference is that instead of the table type to which the operator is bound (which is one of the input group attributes), the output group will have an attribute that is the same as the base element type of the input group. The input group is editable. The output group is not editable.

Figure 25–16 Varray Iterator Operator in a Mapping



To define a Varray Iterator operator in a mapping:

1. Drag and drop a Varray Iterator operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Varray Iterator dialog.

2. From the **Add Varray Iterator** dialog, choose either an unbound operator or a bound operator.

- If you select the unbound operator, then a Varray Iterator operator with no attributes is created. You will have to create these attributes manually.
- If you select the bound operator, then you must choose one of the available nested table or varray types shown in the tree. The output attribute is the same as the base element.

3. Click **OK.**
4. Map the attributes of the Varray Iterator operator.

For an unbound operator, right-click the unbound Varray Iterator operator on the Mapping Editor canvas and then select **Open Details.** This opens the Varray Iterator editor dialog. You can add attributes to the input group by using the **Add** button. You can only change the data type of the attributes in the output group.

View Operator

The View operator enables you to source data from or load data into a view stored in the Warehouse Builder repository

You can bind and synchronize a View operator to a repository view. The repository view must be deployed before the mapping that contains the View operator is generated to avoid errors in the generated code package.

To define a View operator in a mapping:

1. Drag and drop a View operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add View dialog.

2. Use the Add View dialog to create or select a view. For more information on these options, see "Adding Operators that Bind to Repository Objects" on page 6-12.
3. Map the attributes of the View operator.

   If you are using the view as a target, connect the source attributes to the View operator attributes. If you are using the view as a source, connect the View operator attributes to the target.

Using Remote and non-Oracle Source and Target Operators

You can bind a target operator in a mapping to an object in a remote Oracle location or a non-Oracle Database location such as SQL Server or DB2. Such operators are referred to as non-Oracle targets. You use database links to access these targets. The database links are created using the locations. SAP targets are not supported, in that Warehouse Builder will not generate ABAP to populate SAP.

There are certain restrictions to using remote or non-Oracle targets in a mapping as described in the following sections:

- Limitations of Using non-Oracle or Remote Targets
- Warehouse Builder Workarounds for non-Oracle and Remote Targets

Limitations of Using non-Oracle or Remote Targets

The following limitations apply when you use a remote or non-Oracle target in a mapping:

- You cannot set the Loading Type property of the target operator to TRUNCATE/INSERT.

This results in a validation error when you validate the mapping.

- For non-Oracle targets, setting the Loading Type property of the target operator to INSERT/UPDATE produces the same result as setting the loading type to INSERT.
- The RETURNING clause is not supported in a DML statement.

  The RETURNING clause enables you to obtain the ROWIDs of the rows that are loaded into the target using row-based mode. These ROWIDs are recorded by the runtime auditing system. But in the case of a remote or non-Oracle target, the RETURNING clause is not generated and nulls are passed to the runtime auditing system for the ROWID field.
- In set-based mode, you cannot load data from an Oracle Database into a remote or non-Oracle target. All other modes, including set-based failover, are supported.

  When you set the Operating Mode property of the target operator to Set-based, a runtime error occurs.

Warehouse Builder Workarounds for non-Oracle and Remote Targets

When you use a remote or non-Oracle target in a mapping, Warehouse Builder uses default workarounds for certain restricted activities. These workarounds are listed for your information only. You need not explicitly do anything to enable these workarounds.

The default workarounds that Warehouse Builder uses for a remote or a non-Oracle target are as follows:

- When you set the loading type of a target to INSERT/UPDATE or UPDATE/INSERT in Oracle9*i* and to UPDATE in Oracle10*g*, Warehouse Builder generates a MERGE to implement this mapping in set-based mode. But a MERGE statement cannot be run against remote or non-Oracle targets. Thus, when you use a remote or non-Oracle target in a mapping, Warehouse Builder generates code without a MERGE statement. The generated code is the same as that generated when the PL/SQL generation mode is set to Oracle8*i*.
- For set-based DML statements that reference a database sequence that loads into a remote or non-Oracle target, the GLOBAL_NAMES parameter to must be set to TRUE. When Warehouse Builder generates code for a mapping, it sets this parameter to TRUE if the mapping contains a remote or non-Oracle target.
- For a multi-table insert to a remote or non-Oracle target, the Warehouse Builder code generator generates an insert statement for each table instead of a multi-table insert statement.
- While performing bulk inserts on a remote or non-Oracle Database, Warehouse Builder does not generate bulk processing code. Instead, it generates code that processes one row at a time. This means that the Generate Bulk property of the operator is ignored.

Note: The loading types used for remote or non-Oracle targets are the same as the ones used for other Oracle target operators. For more information about the loading type property, see Loading Types for Oracle Target Operators on page 25-2.

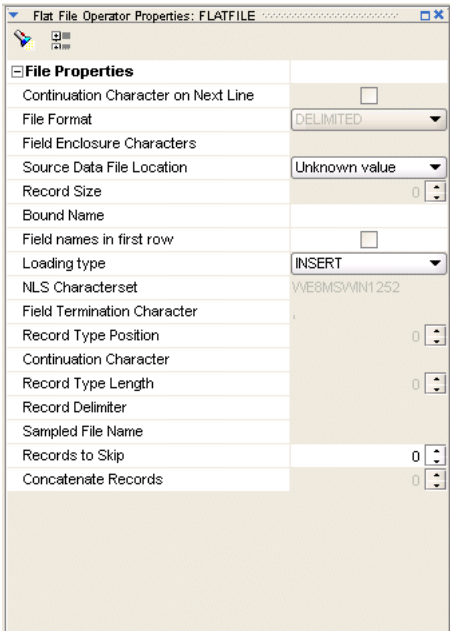
Using Flat File Source and Target Operators

The Flat File operator enables you to use a flat file as a source or target in a mapping.

Setting Properties for Flat File Source and Target Operators

You can set properties for a flat file operator as either a source or target. You can set Loading Types for Flat Files and the Field Names in the First Row setting. All other settings are read-only and depend upon how you imported the flat file. The operator properties window displays as shown in Figure 25–17.

Figure 25–17 Properties Window for Flat File Operator



Loading Types for Flat Files

Select a loading type from the drop-down list:

- **Insert:** Creates a new target file. If a target file already exists, then it is replaced with a new target file.
- **Update:** Creates a new target file. If a target file already exists, then it is appended.
- **None:** No operation is performed on the data target. This setting is useful for test runs where all transformations and extractions are run but have no effect on the target.

Field Names in the First Row

Set this property to **True** if you want to write the field names in the first row of the operator or **False** if you do not.

Flat File Operator

You can use a flat file operator as either a source or target.

However, the two are mutually exclusive within the same mapping. There are differences in code generation languages for flat file sources and targets. Subsequently, mappings can contain a mix of flat files, relational objects, and transformations, but with the restrictions discussed further in this section.

You have the following options for flat file operators:

- Using Previously Imported Flat Files
- Importing and Binding New Flat Files into Your Mapping
- Defining New Flat File Sources or Targets in Mappings

Flat File Source Operators

You can introduce data from a flat file into a mapping using either a flat file operator or an external table operator. If you are loading large volumes of data, loading to a flat file enables you to use the DIRECT PATH SQL*Loader option, which results in better performance.

If you are not loading large volumes of data, you can benefit from many of the relational transformations available in the external table feature. See "External Table Operators versus Flat File Operators" on page 14-3 for more information.

As a source, the flat file operator acts as the row set generator that reads from a flat file using the SQL*Loader utility. Do not use a flat file source operator to map to a flat file target or to an external table. When you design a mapping with a flat file source operator, you can use the following operators:

- Filter Operator on page 26-12
- Constant Operator on page 25-8
- Data Generator Operator on page 25-12
- Sequence Operator on page 25-24
- Expression Operator on page 26-11
- Transformation Operator on page 26-37
- Other relational target objects, excluding the External Table operator.

Note: If you use the Sequence, Expression, or Transformation operators, you cannot use the SQL*Loader Direct Load setting as a configuration parameter.

When you use a flat file as a source in a mapping, remember to create a connector from the flat file source to the relational target for the mapping to deploy successfully.

Flat File Target Operators

A mapping with a flat file target generates a PL/SQL package that loads data into a flat file instead of loading data into rows in a table.

Note: A mapping can contain a maximum of 50 flat file target operators at one time.

You can use an existing flat file with either a single or multiple record types. If you use a multiple-record-type flat file as a target, you can only map to one of the record types. If you want to load all of the record types in the flat file from the same source, you can drop the same flat file into the mapping as a target again and map to a different record type. For an example of this, see "Using Direct Path Loading to Ensure Referential Integrity in SQL*Loader Mappings" on page 8-21. Alternatively, create a separate mapping for each record type you want to load.

To create a new flat file target, see "Creating a New Flat File Target" on page 6-3.

26

Data Flow Operators

The Mapping Editor provides a set of pre-built mapping operators. These operators enable you to define common transformations that specify how data moves from the source to the target.

This chapter provides details on how to use operators in a mapping to transform data. Some operators have wizards that assist you in designing the mapping. And some operators allow you to launch the Expression Builder as an aide to writing SQL expressions.

This chapter includes the following topics:

List of Data Flow Operators

The list of data flow operators is as follows:

Operator Wizards

For operators that require you to make numerous design decisions, Warehouse Builder provides wizards that guide you in defining the operator. Each wizard begins with a welcome page that provides an overview of the steps you must perform. And each wizard concludes with a summary page listing your selections. Use **Next** and **Back** to navigate through the wizard. To close an operator wizard, click **Finish** on the any of the wizard pages.

Warehouse Builder provides wizards for the following operators:

- Key Lookup Operator
- Match-Merge Operator, see Using the Match-Merge Operator
- Name and Address Operator, see Using the Name and Address Operator in a Mapping
- Pivot Operator
- Unpivot Operator

Once you become proficient with defining an operator, you may prefer to disable the wizard and use the Operator Editor instead. To launch the Operator Editor, right-click the operator on the Mapping Editor and select **Open Details**. The Operator Editor displays the same content as the wizard except in tab format rather than wizard pages.

Whether you are using an operator wizard or the Operator Editor, complete the following pages for each operator:

- Operator Wizard General Page
- Operator Wizard Groups Page
- Operator Wizard Input and Output Pages
- Operator Wizard Input Connections

Operator Wizard General Page

Use the General page to specify a name and optional description for the operator. By default, the wizard assigns the operator type as the name. For example, the default name for a new pivot operator is "Pivot".

Operator Wizard Groups Page

Edit group information on the Groups tab.

Each group has a name, direction, and optional description. You can rename groups for most operators but cannot change group direction for any of the operators. A group can have one of these directions: Input, Output, Input/Output.

Depending on the operator, you can add and remove groups from the Groups tab. For example, you add input groups to Joiners and output groups to Splitters.

Operator Wizard Input and Output Pages

The Operator Editor displays a tab for each type of group displayed on the Groups tab. Each of these tabs displays the attribute name, data type, length, precision, scale and optional description.

Depending on the operator, you may be able to add, remove, and edit attributes. The Mapping Editor greys out properties that you cannot edit. For example, if the data type is NUMBER, you can edit the precision and scale but not the length.

Operator Wizard Input Connections

Use the Input Connections page to copy and map attributes into the operator. The attributes you select become mapped members in the input group. The Available Attributes panel displays a list of all the operators in the mapping.

To complete the Input Connections page for an operator:

1. Select complete groups or individual attributes from the Available Attributes panel.

   To search for a specific attribute or group by name, type the text in **Search for** and select **Go.** To find the next match, select **Go** again.

   Hold the **Shift** key down to select multiple groups or attributes. If you want to select attributes from different groups, you must first combine the groups with a Joiner or Set operator.

2. Use the left to right arrow button between the two panels to move your selections to the Mapped Attributes panel.

   You can use the right to left arrow to remove groups or attributes from the input connections list. Warehouse Builder removes the selection from the input group and removes the mapping line between the source operator and the current operator.

The Expression Builder

Some of the data flow operators require that you create expressions. An expression is a statement or clause that transforms data or specifies a restriction. These expressions are portions of SQL that are used inline as part of a SQL statement. Each expression belongs to a type that is determined by the role of the data flow operator. You can create expressions using Expression Builder, or by typing them into the expression field located in the operator or attribute property windows.

Opening the Expression Builder

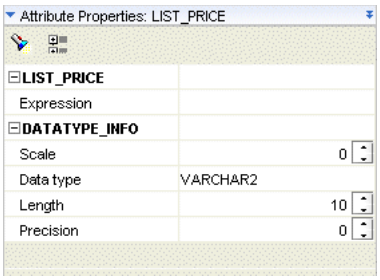
You can open the Expression Builder from the <Operator Name> Properties panel of the Mapping Editor for operators such as filters, joiners, and aggregators.

You can open the Expression Builder from the Attribute Properties panel of the Mapping Editor in the operators such as expressions, data generators, splitters, and constants.

To open the Expression Builder:

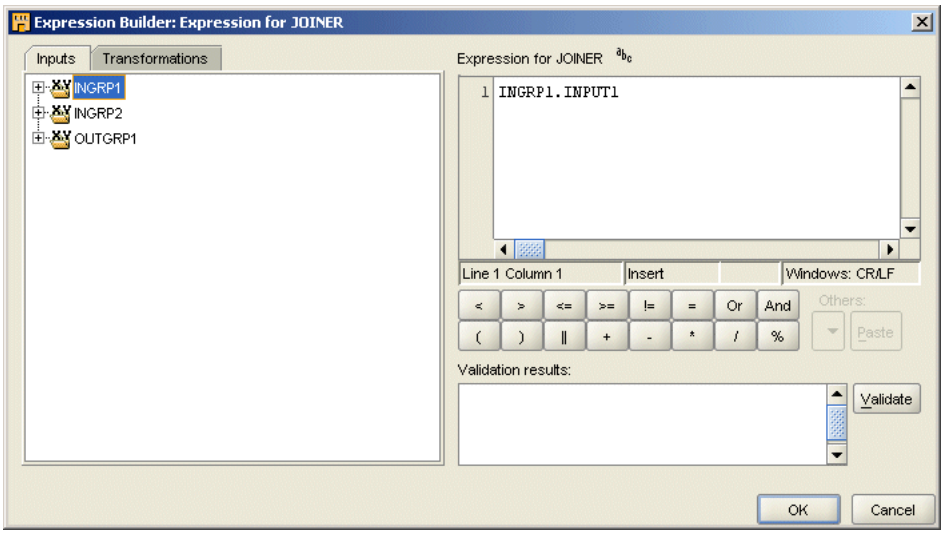
1. From the Properties panel of the operator or an attribute, click the Ellipsis button in the Expression field. Figure 26–1 shows the Attribute Properties panel.

Figure 26–1 Attribute Property Window



The Expression Builder displays as shown in Figure 26–2.

Figure 26–2 Expression Builder Interface



2. Create an expression by:
   - Typing text into the Expression field.
   - Dragging items from the Inputs and Transformations tabs on the left panel and dropping them into the **Expression** field on the right.
   - Double clicking on items from the Inputs and Transformations tabs on the left panel.
   - Clicking arithmetic operator buttons available under the **Expression** field.
3. Click **Validate.**

   This verifies the accuracy of the Expression syntax.
4. Click **OK** to save the expression and close the Expression Builder.

The Expression Builder User Interface

The Expression Builder contains the following parts:

- In the left panel, the navigation tree displays two tabs:
  - **Inputs Tab:** A list of input parameters.
  - **Transformations Tab:** A list of predefined functions and procedures located in the Oracle Library, the Global Shared Library, and a custom Transformation

Library. For more information about transformations, see Chapter 9, "Using Oracle Warehouse Builder Transformations".

- **Expression Field:** At the top of the right panel is the **Expression** field. Use this field to type and edit expressions.
- **Arithmetic Operator Buttons:** Below the **Expression** field are buttons for arithmetic operators. Use these buttons to build an expression without typing. The arithmetic operators available vary by the type of data flow operator that is active.
- **Others:** A drop-down list of available SQL clauses that are appropriate for the active expression type.

  Beginning in Oracle9*i*, the `CASE` function is recommended over the `DECODE` function because the `CASE` function generates both SQL and PL/SQL while `DECODE` is limited to SQL. If you use the `DECODE` function in an expression, Warehouse Builder promotes it to `CASE` where appropriate during code generation. This enables you to deploy the `DECODE` functionality in all operating modes (such as setbased or rowbased) and transparently across Oracle Database releases (8.1, 9.0 and higher).

  For example, Warehouse Builder converts the function

  ```
  DECODE (T1.A, 1, 'ABC', 2, 'DEF', 3, 'GHI', 'JKL')
  ```

  to the following:

  ```
  CASE T1.A WHEN 1 THEN 'ABC'
  WHEN 2 THEN 'DEF'
  WHEN 3 THEN 'GHI'
  ELSE 'JKL'
  ```

- **Validate Button:** Use this button to validate the current expression in the Expression Builder. Validation ensures that all mapping objects referred to by the expression have associated repository objects. The expressions you create with the Expression Builder are limited to the operator inputs and to any transformations available in a project. This limitation protects the expression from becoming invalid because of changes external to the operator. If the deployment database is different from the design repository, it may not accept the expression. If this happens, the expression may be valid but incorrect against the database. In this case, expression errors can only be found at deployment time.
- **Validation Results Field:** At the bottom of the right panel is the **Validation Results** field. After you select the **Validate** button to the right of this field, this field displays the validation results.

Aggregator Operator

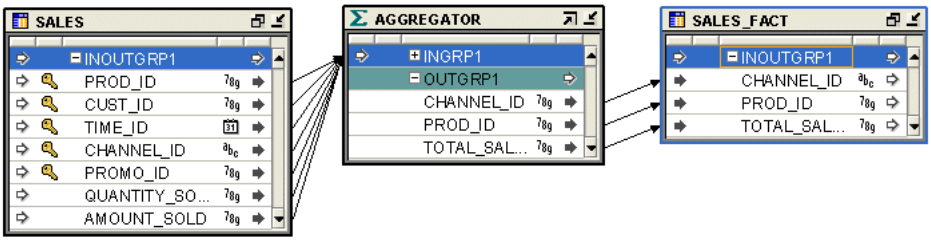
The Aggregator operator calculates data aggregations, such as summations and averages, on the input data. It provides an output row set that contains the aggregated data.

The Aggregator operator has one input group and one output group. For the output group, define a GROUP BY clause that specifies the attributes over which the aggregation is performed. You can optionally specify a HAVING clause that restricts the aggregated data. Each attribute in the output group has the same cardinality. The number of rows in the output row set is less than or equal to the number of input rows.

You can use a single aggregator operator to perform multiple aggregations. Although you can specify a different aggregation function for each attribute in the output group of an aggregator, each aggregator supports only one GROUP BY and one HAVING clause.

Figure 26–3 shows a mapping that uses the Aggregator operator to aggregate the total sales over channels and products. Use the Expression property of the output attribute to specify that the aggregate function to be applied to the attribute `TOTAL_SALES` is SUM. Use the Group By property of the Aggregator operator to specify that the sales are aggregated over channel ID and product ID. The output of the Aggregator operator is mapped to the target table `SALES_FACT`.

Figure 26–3 Aggregator Operator in a Mapping



To define an Aggregator operator in a mapping:

1. Drag and drop an **Aggregator** operator onto the Mapping Editor canvas.
2. On the canvas, connect source attributes to the input group of the Aggregator operator.
3. Right-click the Aggregator operator and select **Open Details**.

   Warehouse Builder displays the Aggregator Editor.
4. On the Output Attributes tab, click **Add** to add an output attribute.

   Warehouse Builder adds an output attribute with the data type NUMBER. You can change both the name and the data type of this attribute.

   In the example displayed in Figure 26–3, you add an output attribute and rename it to TOTAL_SALES.
5. Click **OK** to close the Aggregator Editor.
6. Define expressions for each output attribute by using the Properties Inspector window of the attribute. For detailed instructions, see "Aggregate Function Expression" on page 26-8.

   In the example displayed in Figure 26–3, you define the expression as SUM(amount_sold).
7. Define a Group By clause and an optional Having clause for the operator. For detailed instructions, see "Group By Clause" on page 26-7 and "Having Clause" on page 26-7.
8. Map the attributes in the output group of the Aggregator operator to the input group of the target.

Group By Clause

The Group By clause defines how to group the incoming row set to return a single summary row for each group. An ordered list of attributes in the input group specifies how this grouping is performed. The default GROUP BY clause is NONE.

To define the Group By Clause:

1. Select the Aggregator operator on the Mapping Editor canvas.

   The Aggregator Properties panel displays the properties of the Aggregator operator.

2. Click the Ellipsis button to the right of the Group By Clause property.

   The Group By Clause dialog is displayed.

3. Move the attributes from the Available Attributes list to the GROUP BY Attributes list.

   In the example displayed in Figure 26–3, move the input attributes `channel_id` and `prod_id` from the Available Attributes list to the Group By Attributes list.

4. Click **OK.**

Having Clause

The Having clause is a boolean condition that restricts the groups of rows returned in the output group to those groups for which this condition is true. If this clause is not specified, all summary rows for all groups are returned in the output group.

To define the Having Clause:

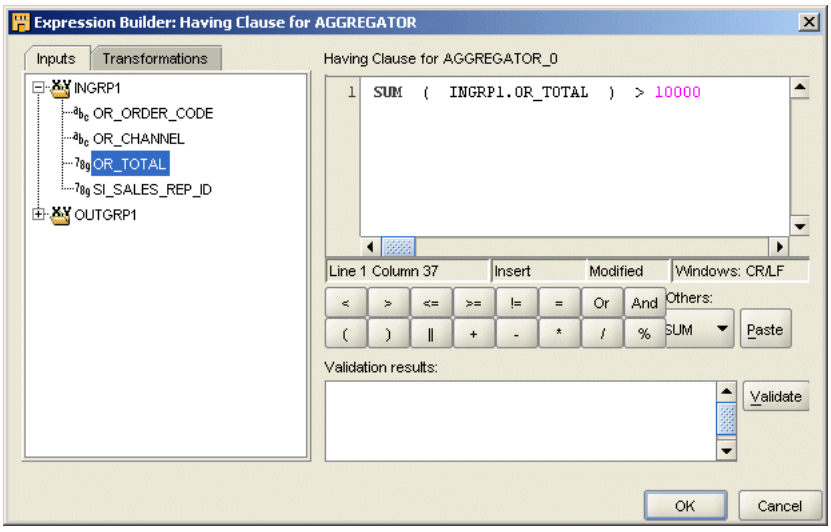
1. Select the Aggregator operator on the mapping canvas.

   The Aggregator Properties panel displays the properties of the Aggregator operator.

2. Click the Ellipsis button to the right of the Having Clause property.

   The Expression Builder dialog for the Having Clause displays as shown in Figure 26–4.

Figure 26–4 Having Clause Dialog



3. Create an expression for the Having Clause of the Aggregator operator. For example Figure 26–4 shows a sample Having Clause expression.
4. Click **OK** to close the Expression Builder.
5. Map the attributes you edited from the output group of the Aggregator operator to the attributes in the target.

Aggregate Function Expression

The Expression property of an attribute defines the aggregation functions to be performed on the attribute. For each ungrouped output attribute, select whether the aggregation expression should be a DISTINCT or ALL result. ALL is the default setting. For example,

- ALL: `Select AVG(ALL sal) from emp;`
- DISTINCT: `Select AVG(DISTINCT sal) from emp;`

A DISTINCT result removes all duplicate rows before the average is calculated.

An ALL result returns an average value on all rows.

If no aggregation function is necessary, specify NONE for the function. Specifying NONE on the attribute aggregation automatically adds the attribute to the resulting GROUP BY function.

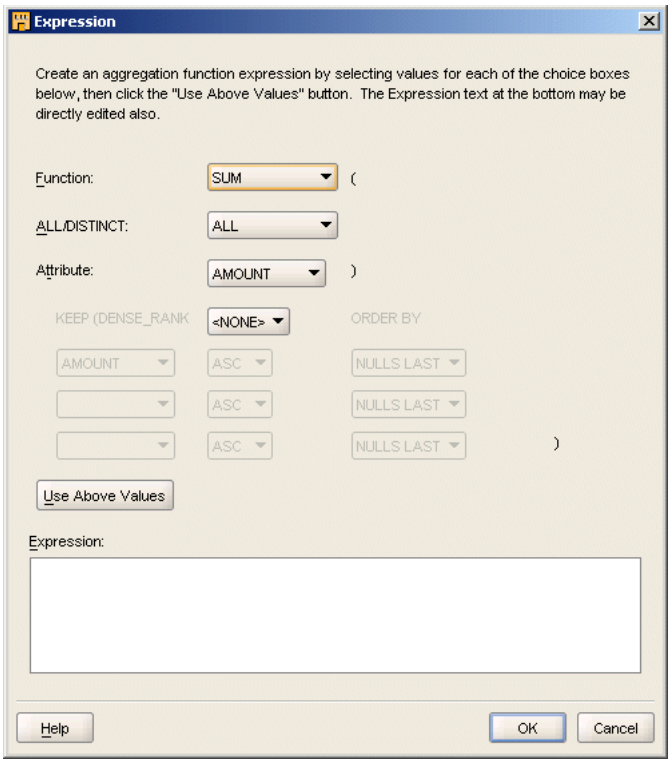
To define expressions for output attributes:

1. In the Aggregator operator on the mapping canvas, select the output attribute for which you want to define an aggregate function.

   The Attribute Properties panel displays the properties of the selected output attribute.
2. Click the Ellipsis button to the right of the **Expression** property.

   The Expression dialog displays as shown in Figure 26–5.

Figure 26–5 Expression Dialog



3. Select an aggregate function from the Function drop-down list.

   The aggregate functions you can select in Warehouse Builder are as follows: AVG, COUNT, MAX, MIN, NONE, STDDEV, STDDEV_POP, STDDEV_SAMP, SUM, VAR_POP, VAR_SAMP, and VARIANCE.

   In the example displayed in Figure 26–3, you select SUM as the aggregate function.

4. Select either ALL or DISTINCT as the aggregation expression.

5. Select the attribute that should be aggregated from the Attribute drop-down list.

   In the example displayed in Figure 26–3, you select the attribute `amount_sold` from the drop-down list.

6. Click **OK.**

Anydata Cast Operator

Anydata Cast operator enables you to convert an object of type Sys.AnyData to either a primary type or to a user defined type. The Anydata Cast operator accepts an Anydata attribute as a source and transforms the object to the desired type.

The Anydata Cast operator is used with user defined data types and primitive data types. The cast operator acts as a filter. The number of attributes in the output group is n+1 where n is the number of attributes in the input group. This operator has one input group and one output group. The input group is editable. The output group is not editable. In an output group, you can only rename the attributes and change the data type of only the cast target. You cannot change the data type of any other output group attribute.

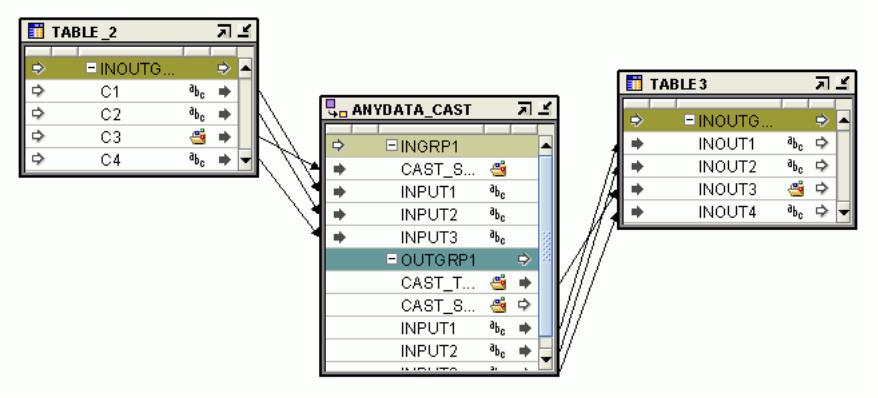
You can connect attributes to the input group. Each output group gets a copy of the input group attributes, including the Anydata attributes. You must choose an Anydata attribute of the input group as the source of the Cast operation.

If you change the data type to which you are going to cast the Anydata attribute, then you must:

1. Edit the output group attribute that is the target of the Cast operation
2. Change the data type of the attribute.

Because the Cast operator is unbound, it will not support any reconciliation operations.

Figure 26–6 Anydata Cast in a Mapping



To define a Anydata Cast operator in a mapping:

1. Drop an Anydata Cast operator onto the Mapping Editor canvas.

   The AnyData Cast dialog is displayed. The tree inside the dialog has one parent node that will open to display the primary data types (other than Anydata). Each of the other parent nodes will correspond to the modules.

2. Select the target type for casting and click **Finish**.
3. Right-click the ANYDATA CAST operator and select **Open Details**.

   Warehouse Builder displays the ANYDATA_CAST Editor.

4. On the Input Attributes tab, click **Add** and specify the attribute name, data type, and other properties.
5. Click **OK** to close the Operator Editor.
6. Map the attributes of the output group of the Anydata Cast operator to the target.

Deduplicator Operator

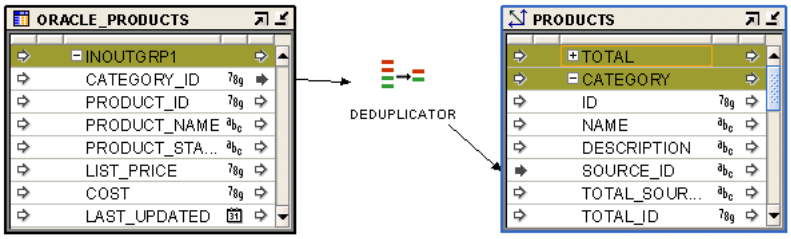
The Deduplicator enables you to remove duplicate data in a source by placing a DISTINCT clause in the select code represented by the mapping.

For example, when you load data from a source table into a dimension, the higher levels within a dimension may be duplicated in the source. Figure 26–7 displays a mapping that uses the Deduplicator operator to remove duplicate values in the source while loading data into the `PRODUCTS` dimension. The source table contains duplicate

values for category ID because more than one products may belong to the same category. The Deduplicator operator removes these duplicates and loads distinct values of category ID into the PRODUCTS dimension.

All rows that are required by the target must pass through the deduplicator. No row set can bypass the deduplicator and hit the target directly.

Figure 26–7 Deduplicator in a Mapping



To remove duplicates:

1. Drop the Deduplicator operator onto the Mapping Editor canvas.
2. Connect the attributes from the source operator to the input/output group of the Deduplicator operator.
3. Connect the attributes from the Deduplicator operator group to the attributes of the target operator.

Expression Operator

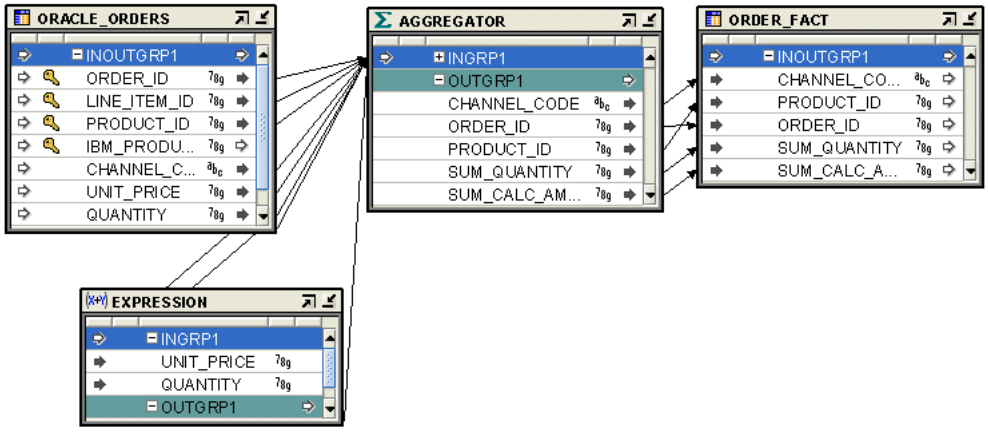
Use the Expression operator to write SQL expressions that define non-procedural algorithms for one output parameter of the operator.

The expression text can contain combinations of input parameter names, variable names, and library functions. Use the Expression operator to transform the column value data of rows within a row set using SQL-type expressions, while preserving the cardinality of the input row set. To create these expressions, open the Attribute properties window for the output attribute and then open the Expression Builder.

By default, the Expression operator contains one input group and one output group.

Figure 26–8 shows a mapping that uses the Expression operator. The transaction table ORACLE_ORDERS contains order details such as product ID, unit price, and quantity sold. The ORDERS_FACT table contains an aggregation of the total sales amount across channels, products, and orders. The Expression operator is used to compute the amount of sale for each product by multiplying the unit price by the quantity sold. The Aggregator operator aggregates the sale amounts over channel code, product ID, and order ID before loading the target table.

Figure 26–8 Expression Operator in a Mapping



Do not use the expression operator to write aggregation functions. Use the Aggregator operator. See "Aggregator Operator" on page 26-5.

To define an Expression operator in a mapping:

1. Drag and drop an Expression operator onto the Mapping Editor canvas.
2. Right-click the Expression operator and select **Open Details**.

   Warehouse Builder displays the Expression Editor.
3. On the Output Attributes tab, click **Add** and specify the attribute name, data type, and other properties.
4. Click **OK** to close the Operator Editor.
5. From the Expression operator, select the output attribute.

   The Attribute Properties panel window displays the properties of the selected output attribute.
6. Click the Ellipsis button to the right of the Expression field.

   The Expression Builder is displayed. Define an expression in the Expression Builder.

   For code generation, the input attributes are replaced by the input attribute names in the expression template.
7. Connect the Expression output attribute to the appropriate target attribute.

Filter Operator

You can conditionally filter out rows using the Filter operator.

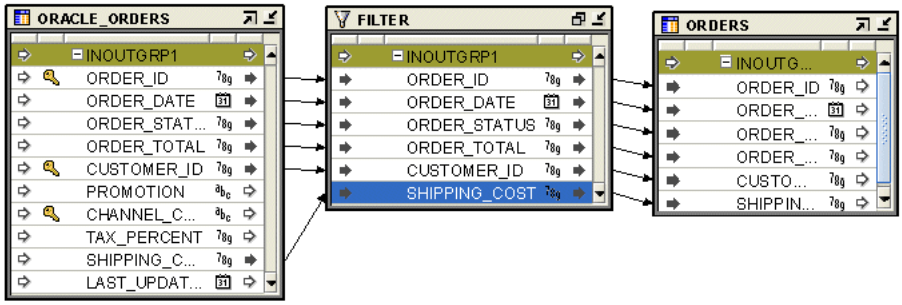
You connect a source operator to the Filter operator, apply a filter condition, and send a subset of rows to the next operator. The Filter operator filters data from a source to a target by placing a WHERE clause in the code represented by the mapping. You specify the filter condition using the Expression Builder. The filter condition can be based on all supported data types and can contain constants.

A Filter operator has only one input/output group that can be connected to both a source and target row set. The resulting row set is a filtered subset of the source row set based on a boolean filter condition expression. All rows that are required at the target must pass through the filter operator. No row set can bypass the filter and hit the target directly.

For a mapping that contains a Filter operator, Warehouse Builder generates code that displays the filter condition expression as a WHERE clause for set-based view mode. The filter input names in the original filter condition are replaced by actual column names from the source table, qualified by the source table alias.

Figure 26–9 shows the mapping that uses the Filter operator to move selected data to the target table. The ORACLE_ORDERS table contains orders data. Use the Filter Condition property of the Filter operator to move only the booked orders which were last updated on the current system date into the ORDERS table.

Figure 26–9 Filter in a Mapping



To define a Filter operator in a mapping:

1. Drag and drop the **Filter** operator onto the Mapping Editor canvas.
2. Connect source attributes to the input/output group of the Filter operator.
3. Select the Filter operator header.

   The Filter Properties panel of the Mapping Editor displays the properties of the Filter operator.
4. Click the Ellipsis button the right of the Filter Condition property.

   Warehouse Builder displays the Expression Builder dialog for the filter condition.
5. Define a filter condition expression using the Expression Builder.
6. Click **OK** to close the Expression Builder.
7. Connect the Filter operator outputs to the input/output group in the target.

Joiner Operator

You use the Joiner operator to join multiple row sets from different sources with different cardinalities, and produce a single output row set.

The Joiner operator uses a boolean condition that relates column values in each source row set to at least one other row set. The Joiner operator results in a WHERE clause in the generated SQL query. Warehouse Builder supports inner joins, outer joins, self joins, equijoins, and non-equijoins. When run on Oracle9*i*, Warehouse Builder supports full outer joins. For more information on joins refer the *Oracle SQL Reference*.

Note: Operators placed between data sources and a Joiner can generate complex SQL or PL/SQL.

If the input row sets are related through foreign keys, that relationship is used to form a default join condition. You can use this default condition or you can modify it. If the sources are not related through foreign keys, then you must define a join condition.

If two tables in a join query do not have a join condition specified, Warehouse Builder returns the Cartesian product of the two tables and combines each row of one table with each row of the other table.

If the default foreign keys result in duplicate WHERE clauses, the Joiner operator will remove the duplicate clauses. This can happen if the join condition references several foreign keys. For example, if table T1 has a foreign key FK1 point to unique key UK1 in table T2 and table T2 has a foreign key FK2 pointing to unique key UK2 in T1, the resulting join condition

```
T1.A = T2.A AND T1.B = T2.B /*All instances of FK1 -> UK1 are reduced to one WHERE
clause*/ AND
T2.B = T1.B AND T2.C = T1.C /*All instances of FK2 -> UK2 are reduced to one
E-Business Suite clause*/
```

is generated by the Joiner operator as

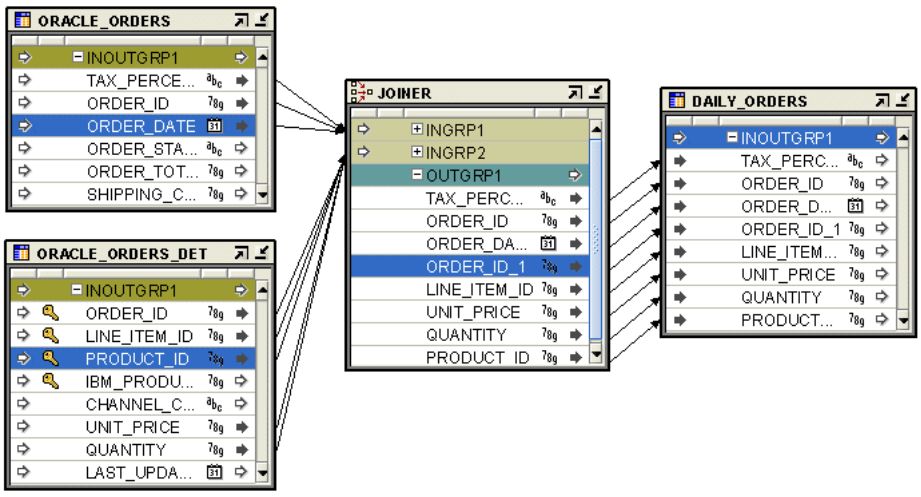
```
T2.A = T2.A AND T1.B = T2.B AND T2.C = T1.C
```

If you define a join condition before you map attributes to the input group of the Joiner operator, the generated code treats the join condition as a literal. Since the attributes are not yet mapped to the Joiner operator, the code generator does not recognize these attributes. To avoid this problem, it is recommended that you first map the input groups of the Joiner operator and then define the join condition.

The join condition is defined in a PL/SQL context. For SAP sources, Warehouse Builder can generate ABAP code by interpreting the PL/SQL join condition in the ABAP context. ABAP can only join over defined foreign key relationships.

Figure 26–10 shows a mapping that contains a Joiner operator. The two source tables ORACLE_ORDERS and ORACLE_ORDER_LINES are joined to combine the data from these tables into one table. The output of the Joiner operator is passed to the target table DAILY_ORDERS.

Figure 26–10 Joiner in a Mapping



To define a Joiner operator in a mapping:

1. Drag and drop the Joiner operator onto the Mapping Editor canvas.

2. Connect an output group from the first source to the Joiner input group INGRP1.

   The output attributes are created with data types matching the corresponding input data types.

3. Connect a group from the second source operator to the INGRP2 group of the Joiner operator.

4. Select the Joiner operator header.

   The Joiner Properties panel of the Mapping Editor displays the properties of the Joiner operator.

5. Click the Ellipsis button to the right of the Join Condition property.

   The Expression Builder dialog is displayed.

6. Define the join condition.

7. Click **OK** to close the Expression Builder.

8. Map the attributes of the output group of the Joiner operator to the target.

Joiner Restrictions

Do not include aggregation functions in a join condition.

A Joiner can have an unlimited number of input groups but only one output group.

The order of input groups in a joiner is used as the join order. The major difference between ANSI join and an Oracle join is that ANSI join must clearly specify join order. An Oracle join does not require it.

```
SELECT ...
      FROM T1 FULL OUTER JOIN T2 ON (T1.A=T2.A)
                          JOIN T3 ON (T2.A=T3.A);
```

If you create input groups in another order, such as T1, T3, T2. Warehouse Builder will generate the following:

```
SELECT ...
FROM T1 JOIN T3 ON (1=1)
        JOIN T2 ON (T1.A=T2.A and T2.A=T3.A);
```

When T1 and T3 are joined, there is no join condition specified. Warehouse Builder fills in a condition 1=1 (essentially a boolean true) and the two conditions you specified are used to join T2.

The filter condition is applied after join. For example, consider the following join:

```
Input1.c --- +
Input2.c --- +---> Joiner
Input3.c --- +
```

with the following conditions:

- **Condition 1:** Input1.c (+) = Input2.c (+)
- **Condition 2:** Input2.c = Input3.c
- **Condition 3:** Input1.c is null

The first two conditions are true joins while the third is a filter condition. If ANSI code is to be generated, Warehouse Builder interprets the statement as

```
select ...
```

```
from Input1 full outer join Input2 on (Input1.c = Input2.c)
join Input3 on (Input2.c = Input3.c)
WHERE Input1.c is not null;
```

Specifying a Full Outer Join

If your target warehouse is based on Oracle9*i* or a later version, the Warehouse Builder Joiner operator also supports the full outer join. To specify a full outer join condition, you must place the (+) sign on both sides of a relational operator. The relational operator is not restricted to equality. You can also use other operators such as, >, <, !=, >=, <= .

```
T1.A (+) = T2.B (+)
```

The results of the full outer join are as follows:

- Rows from sources T1 and T2 that satisfy the condition T1.A = T2.B.
- Rows from source T1 that do not satisfy the condition. Columns corresponding with T2 are populated with nulls.
- Rows from source T2 that do not satisfy the condition. Columns corresponding with T1 are populated with nulls.

When using the Oracle SQL syntax for partial outer join such as T1.A = T2.B (+), if you place a (+) sign on both sides of the relational operator, it is invalid Oracle SQL syntax. However, Warehouse Builder translates any condition with the double (+) sign into ANSI SQL syntax. For example,

```
SELECT ...
FROM T1 FULL OUTER JOIN T2 ON (T1.A = T2.B);
```

When using full outer join, keep in mind the following:

- Do not specify a full outer join condition for versions earlier than Oracle9*i*.
- The ANSI join syntax is generated only if you specify a full outer join condition in the joiner. Otherwise, the following Oracle proprietary join syntax is generated:

  ```
  SELECT ...
  FROM T1, T2
  WHERE T1.A = T2.B;
  ```

- You can specify both full outer join and join conditions in the same joiner. However, if both conditions are specified for the same sources, the stronger join type is used for generating code. For example, if you specify:

  ```
  T1.A(+) = T2.A(+) and T1.B = T2.B
  ```

  Warehouse Builder will generate a join statement instead of a full outer join because `T1.B = T2.B` is stronger than the full outer join condition between T1 and T2.
- You cannot specify a full outer join and partial outer join condition in the same joiner. If you specify a full outer join, then you cannot specify a partial outer join anywhere in the join condition. For example, `T1.A (+) = T2.A (+) and T2.B = T3.B (+)` is not valid.

Creating Full Outer Join Conditions

In an equijoin, key values from the two tables must match. In a full outer join, key values are matched and nulls are created in the resulting table for key values that cannot be matched. A left or a right outer join retains all rows in the specified table.

In Oracle8*i*, you create an outer join in SQL using the join condition variable (+):

```
SELECT ...
FROM A, B
WHERE A.key = B.key (+);
```

This example is a left outer join. Rows from table A are included in the joined result even though no rows from table B match them. To create a full outer join in Oracle8*i*, you must use multiple SQL statements.

The Expression Builder allows the following syntax for a full outer join:

```
TABLE1.COL1 (+) = TABLE2.COL2 (+)
```

This structure is not supported by Oracle8*i*. Oracle Database is ANSI SQL 1999 compliant. The ANSI SQL 1999 standard includes a solution syntax for performing full outer joins. The code generator translates the preceding expression into an ANSI SQL 1999 full outer join statement, similar to:

```
SELECT ...
FROM table1 FULL OUTER JOIN table2 ON (table1.col1 = table2.col2)
```

Because the full outer join statement complies to ANSI SQL 1999, it is only valid if the generated code is deployed to an Oracle9*i* database. Specifying a full outer join to an Oracle8*i* database results in a validation error.

A full outer join and a partial outer join can be used together in a single SQL statement, but it must in an AND or an AND/OR condition. If a full outer join and partial outer join are used in the OR condition, an unexpected AND condition will result. For example,

```
SELECT ...
FROM table1 FULL OUTER JOIN table2 ON (A = B or C = D)
```

is evaluated by Oracle Server as `A (+) = B (+) AND C = D`.

To use a full outer join in a mapping:

1. Follow steps one through four on page 26-14 for adding a Joiner operator.
2. Click the Ellipsis button to the right of the Join Condition property to define an expression for the full outer join using the Expression Builder.
3. Click **OK** to close the Expression Builder.

Key Lookup Operator

Use the Key Lookup operator to lookup data from a table, view, cube, or dimension. For example, use the Key Lookup operator when you define a mapping that loads a cube or when you define surrogate keys on the dimension. For more information about surrogate identifiers, see "Defining Levels" on page 4-31.

The key that you look up can be any unique value. It need not be a primary or unique key, as defined in an RDBMS. The Key Lookup operator reads data from a lookup table using the key input you supply and finds the matching row. This operator

returns a row for each input key. You can have multiple Key Lookup operators in the same mapping.

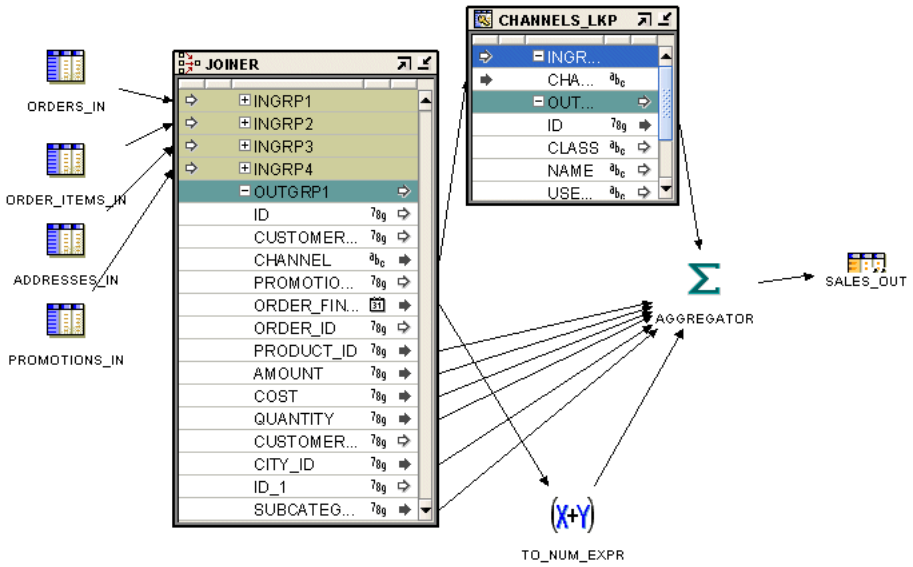
The output of the Key Lookup operator corresponds to the columns in the lookup object. To ensure that only a single lookup row is found for each key input row, use keys in your match condition.

Each output attribute for the key lookup has a property called DEFAULT VALUE. The DEFAULT VALUE property is used instead of NULL in the outgoing row set if no value is found in the lookup table for an input value. The generated code uses the NVL function. The Key Lookup always results in an outer-join statement.

The table, view, or dimension from which the data is being looked up is bound to the Key Lookup operator. You can synchronize a Key Lookup operator with the repository object to which it is bound. But you cannot synchronize the repository object with the Key Lookup operator. For more information on synchronizing operators, refer "Synchronizing Operators based on Repository Objects" on page 6-31.

Figure 26–11 shows a mapping that is used to load a cube. Data from four source tables is joined using a Joiner operator. But the data in the source tables only contains a channel name. To load the cube, we need the value of the surrogate identifier. A key lookup operator is used to lookup the surrogate identifier of the CHANNELS dimension and then load this value into the cube.

Figure 26–11 Key Lookup in a Mapping



Using the Key Lookup Operator

You have the following options for using a Key Lookup operator:

- **Define a new Key Lookup operator:** Drag a Key Lookup operator from the Palette onto the mapping. The Mapping Editor launches a wizard.
- **Edit an existing Key Lookup operator:** Right-click the Key Lookup operator and select **Open Details**.

Whether you are using the operator wizard or the Operator Editor, complete the following pages:

- General

- Groups
- Input Connections
- Lookup
- Type 2 History Lookup
- No-match Rows

General

Use the General page to specify a name and optional description for the key lookup operator. By default, the wizard names the operator "Key_Lookup".

Groups

Use the Groups page to specify one input and one output group.

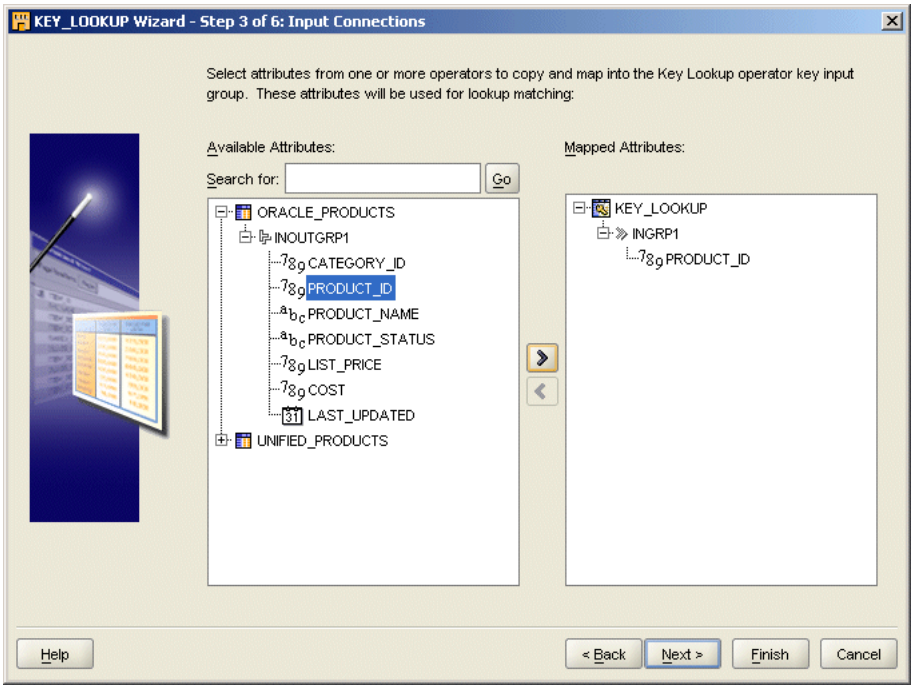
In a Key Lookup operator, the input group represents the data from the source that is contained across multiple attributes. The output group represents that data transformed into rows.

You can rename and add descriptions to the input and output groups. Since each Key Lookup operator must have exactly one input and one output group, the wizard prevents you from adding or removing groups or changing group direction.

Input Connections

Use the Input Connections page to copy and map attributes into the Key Lookup operator. The attributes you select are used to perform a lookup on the lookup object and find matching rows. The left side of the page displays a list of all the operators in the mapping. Figure 26–12 shows an attribute from the table ORACLE_PRODUCTS table selected as input for the key lookup operator.

Figure 26–12 Input Connections Page of the Key Lookup Operator



To complete the Input Connections page for a Key Lookup operator:

1. Select complete groups or individual attributes from the left panel.

   To search for a specific attribute or group by name, type the text in **Search for** and select **Go.** To find the next match, select **Go** again. You can select multiple attributes by pressing the Shift key.

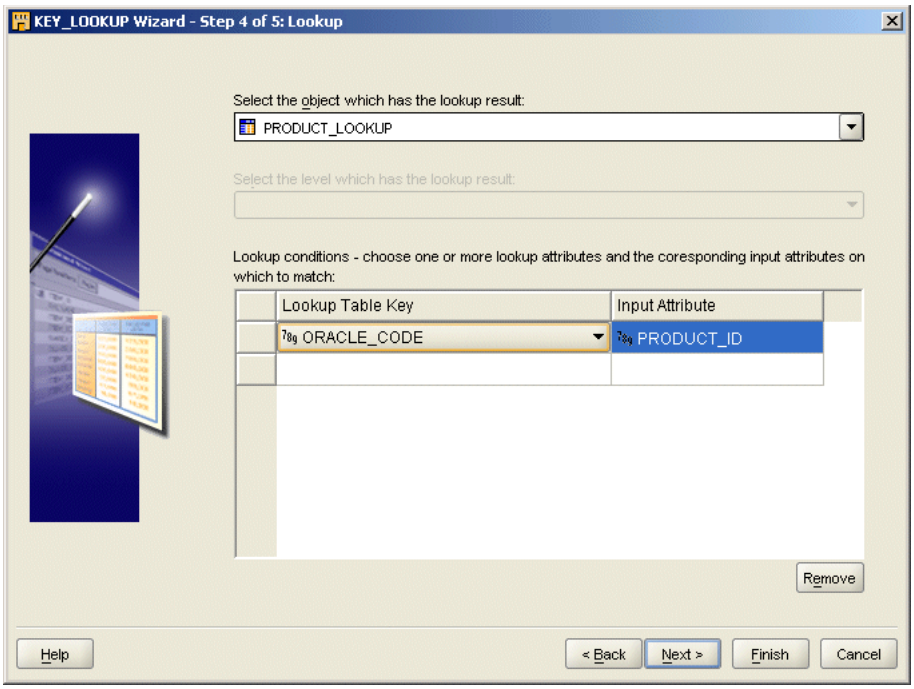
2. Use the right arrow button in the middle of the page to move your selections to the right side of the wizard page.

   Use the left arrow to remove groups or attributes from the input connections list. Warehouse Builder removes the selection from the input group and removes the mapping line between the source operator and the pivot operator.

Lookup

Use the Lookup page to provide details about the object on which the lookup is being performed. This object is referred to as the lookup result. You can perform a lookup on any table, view, or dimension that belongs to the current project. Figure 26–13 displays the Lookup page for the mapping shown in Figure 26–11.

Figure 26–13 Lookup Page of the Key Lookup Operator



To provide the lookup details, select values for the following:

- Object that has the lookup result

  Use the **Select the object which has the lookup result** drop-down list to select the object on which you want to perform the lookup. This drop-down list displays all the modules in the current project. Expand a module node to display the objects in the module. Select the object that contains the lookup result from this list.

- Level that has the lookup result

  When you select a dimension as the lookup result, you must specify which level within the dimension contains the lookup result. The **Select the level which has**

the lookup result drop-down list displays the levels in the selected dimension. Select the dimension level that contains the lookup result.

- Lookup Condition

  Specify the condition used to match the input attributes with the records in the lookup result. The **Lookup Table key** drop-down list displays the unique constraints in the lookup result. The **Input Attribute** drop-down list displays the attributes you selected on the Input Connections page. Use these drop-down lists to specify the attributes used to match records.

  If you select a dimension level for the lookup, the options displayed are the surrogate and business identifier of the dimension level and the primary key of the database table that stores the dimension data.

Type 2 History Lookup

Use this page only if you selected a Type 2 SCD as the lookup result on the Lookup page. When the lookup result is a Type 2 SCD, you must specify which version of a particular record is to be used as a lookup. The options you can choose are as follows:

- **Use the most current record**

  This option returns the current record that corresponds to the attribute being looked up using the lookup condition. The current record is the one with the latest timestamp.

- **Specify the historic date as a constant value**

  This option returns the record that contains the constant value that is specified using the **Date** and **Time** drop-down lists.

- **Choose an input attribute that holds the historic value**

  This option enables you return records that pertain to a date and time that is contained in one of the input attributes. Use the **Input Attribute** drop-down list to select the attribute that contains the historic value.

No-match Rows

Use the No-match Rows page to indicate the action to be taken when there are no rows that satisfy the lookup condition specified on the Lookup page. Select one of the following options:

- **Return no row**

  This option does not return any row when no row in the lookup result satisfies the matching condition.

- **Return a row with the following default values**

  This option returns a row that contains default values when the lookup condition is not satisfied by the lookup result. Use the table below this option to specify the default values for each lookup column.

Pivot Operator

The pivot operator enables you to transform a single row of attributes into multiple rows.

Use this operator in a mapping when you want to transform data that is contained across attributes instead of rows. This situation can arise when you extract data from non-relational data sources such as data in a crosstab format.

Example: Pivoting Sales Data

The external table SALES_DAT, shown in Figure 26–14, contains data from a flat file. There is a row for each sales representative and separate columns for each month. For more information on external tables, see "Using External Tables" on page 14-26.

Figure 26–14 SALES_DAT

ID	Reg	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
0675	4	10.5	11.4	9.5	8.7	7.4	7.5	7.8	9.7				
0676	3	9.5	10.5	10.3	7.6	8.0	7.8	8.7	8.9				
0679	3	8.7	7.4	7.5	7.8	9.7	10.3	7.6	8.0				
0683	2	9.5	10.5	10.3	9.5	8.7	7.4	7.8	8.7				
0684	1	11.4	9.5	8.7	7.4	7.5	10.3	9.5	8.7				
0687	1	9.5	8.7	7.4	7.8	8.7	7.4	7.5	7.8				
0690	1	8.7	7.4	7.8	8.7	11.4	9.5	8.7	7.4				

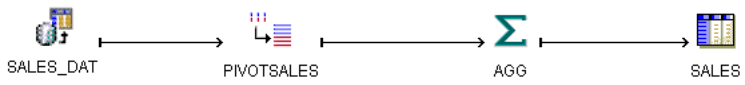
Table 26–1 shows a sample of the data after Warehouse Builder performs a pivot operation. The data that was formerly contained across multiple columns (M1, M2, M3...) is now contained in a single attribute (Monthly_Sales). A single ID row in SALES_DAT corresponds to 12 rows in pivoted data.

Table 26–1 Pivoted Data

REP	MONTH	MONTHLY_ SALES	REGION
0675	Jan	10.5	4
0675	Feb	11.4	4
0675	Mar	9.5	4
0675	Apr	8.7	4
0675	May	7.4	4
0675	Jun	7.5	4
0675	Jul	7.8	4
0675	Aug	9.7	4
0675	Sep	NULL	4
0675	Oct	NULL	4
0675	Nov	NULL	4
0675	Dec	NULL	4

To perform the pivot transformation in this example, create a mapping like the one shown in Figure 26–15.

Figure 26–15 Pivot Operator in a Mapping



In this mapping, Warehouse Builder reads the data from the external table once, pivots the data, aggregates the data, and writes it to a target in set based mode. It is not

necessary to load the data to a target directly after pivoting it. You can use the pivot operator in a series of operators before and after directing data into the target operator. You can place operators such as filter, joiner, and set operation before the pivot operator. Since pivoted data in Warehouse Builder is not a row-by-row operation, you can also execute the mapping in set based mode.

The Row Locator

In the pivot operator, the row locator is an output attribute that you create to correspond to the repeated set of data from the source. When you use the pivot operator, Warehouse Builder transforms a single input attribute into multiple rows and generates values for a row locator. In this example, since the source contains attributes for each month, you can create an output attribute named 'MONTH' and designate it as the row locator. Each row from SALES_DAT then yields 12 rows of pivoted data in the output.

Table 26–2 shows the data from the first row from SALES_DAT after Warehouse Builder pivots the data with 'MONTH' as the row indicator.

Table 26–2 Data Pivoted By Row Indicator

REP	MONTH	MONTHLY_ SALES	REGION
0675	Jan	10.5	4
0675	Feb	11.4	4
0675	Mar	9.5	4
0675	Apr	8.7	4
0675	May	7.4	4
0675	Jun	7.5	4
0675	Jul	7.8	4
0675	Aug	9.7	4
0675	Sep	NULL	4
0675	Oct	NULL	4
0675	Nov	NULL	4
0675	Dec	NULL	4

Using the Pivot Operator

You have the following options for using a pivot operator:

- **Define a new pivot operator:** Use the Pivot Wizard to add a new pivot operator to a mapping. Drag a pivot operator from the Palette onto the mapping. The Mapping Editor launches the Pivot Wizard.
- **Edit an existing pivot operator:** Use the Pivot Editor to edit a pivot operator you previously created. Right-click the operator and select **Open Details**. The Mapping Editor opens the Pivot Editor.

Whether you are using the Pivot Wizard or the Pivot Editor, complete the following pages:

- General
- Groups

- Input Connections
- Input Attributes
- Output Attributes
- Pivot Transform

General

Use the General page to specify a name and optional description for the pivot operator. By default, the wizard names the operator "Pivot".

Groups

Use the Groups page to specify one input and one output group.

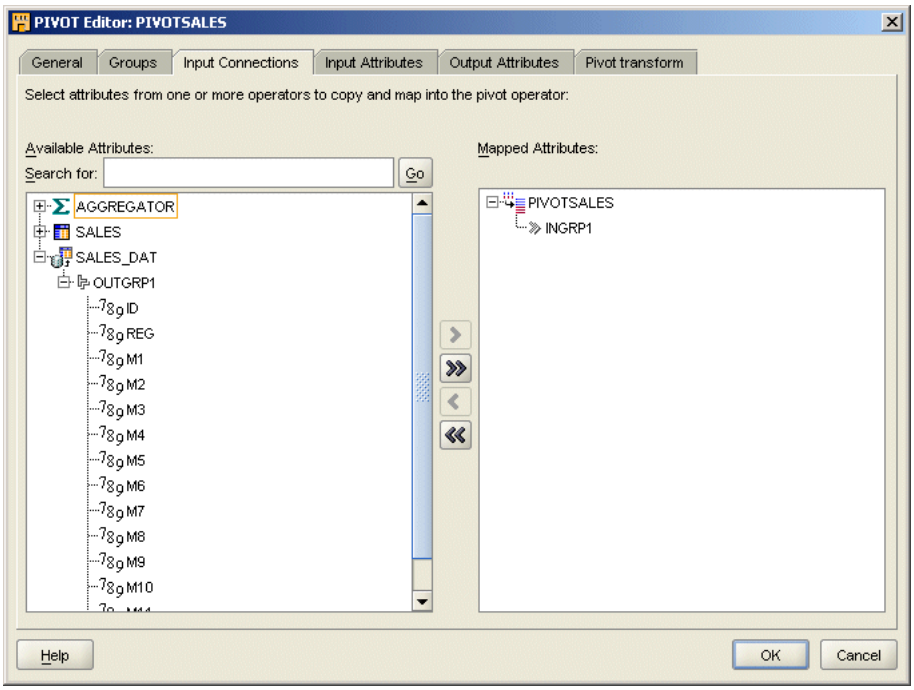
In a pivot operator, the input group represents the data from the source that is contained across multiple attributes. The output group represents that data transformed into rows.

You can rename and add descriptions to the input and output groups. Since each pivot operator must have exactly one input and one output group, the wizard prevents you from adding or removing groups or changing group direction.

Input Connections

Use the Input Connections page to copy and map attributes into the pivot operator. The attributes you select become mapped to the pivot input group. The left side of the page displays a list of all the operators in the mapping. Figure 26–16 shows a group from the external table SALES_DAT selected as input for the pivot operator.

Figure 26–16 Pivot Operator Input Connections Tab



To complete the Input Connections page for a Pivot operator:

1. Select complete groups or individual attributes from the left panel.

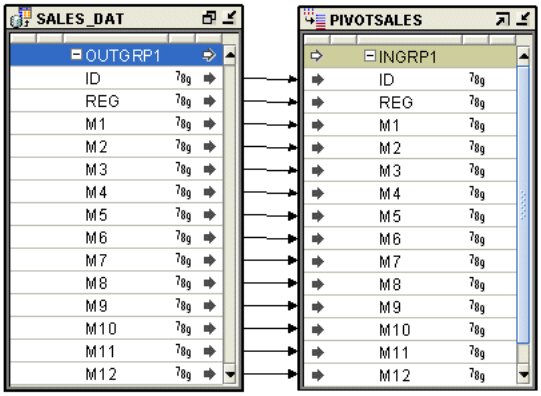
To search for a specific attribute or group by name, type the text in **Search for** and select **Go.** To find the next match, select **Go** again.

Press the Shift key to select multiple attributes. If you want to select attributes from different groups, you must first combine the groups with a Joiner or Set operator.

2. Use the right arrow button in the middle of the page to move your selections to the right side of the wizard page.

   Use the left arrow to remove groups or attributes from the input connections list. Warehouse Builder removes the selection from the input group and removes the mapping line between the source operator and the pivot operator. Figure 26–17 shows a group from `SALES_DAT` copied and mapped into the `PIVOTSALES` operator.

Figure 26–17 Attributes Copied and Mapped into Pivot In Group



Input Attributes

Use the Input Attributes page to modify the attributes you selected in the Input Connections tab or wizard page.

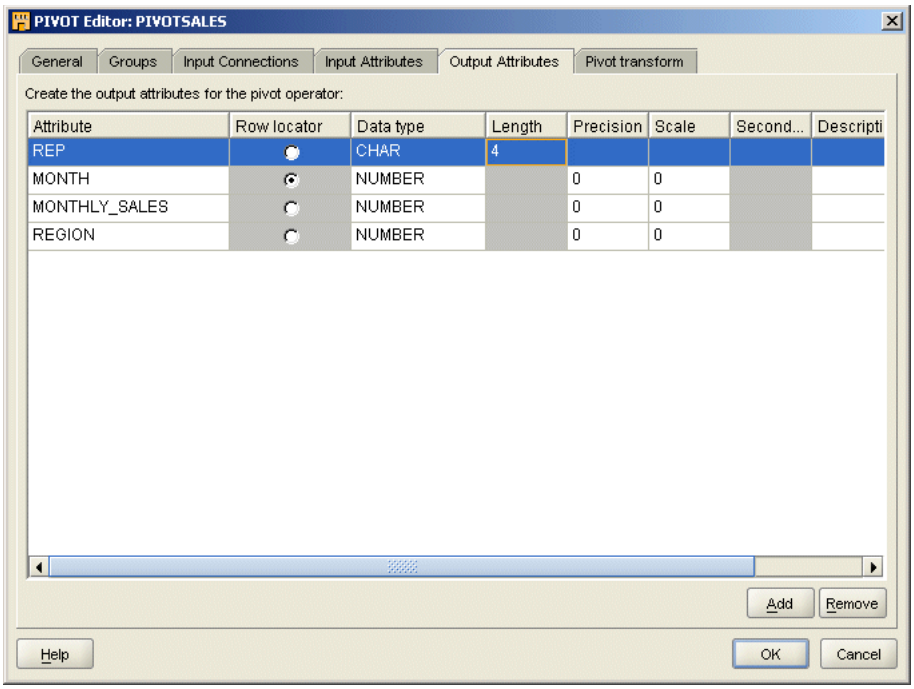
You can perform the following tasks from the Input Attributes page:

- **Add attributes:** Use the Add button to add input attributes.
- **Change attribute properties:** You can change the attribute name, data type, length, precision, and scale.
- **Add an optional description:** Type a description for the input attributes.
- **Designate attribute keys:** As an option, use the **Key** check box to indicate an attribute that uniquely identifies the input group.

Output Attributes

Use the Output Attributes page to create the output attributes for the pivot operator. If you designated any input attributes as keys on the Input Attributes tab or wizard page, Warehouse Builder displays those input attributes as output attributes that you cannot edit or delete. Figure 26–18 displays the output attributes with `MONTH` selected as the row locator.

Figure 26–18 Pivot Output Attributes Tab



You can perform the following tasks from the pivot Output Attributes Page:

- **Change attribute properties:** Except for attributes you designated as keys on the previous tab or wizard page, you can change the attribute name, data type, length, precision, and scale.
- **Add an optional description:** Type a description for the output attributes.
- **Designate a row locator:** Although you are not required to designate a row locator for the pivot operator, it is recommended. When you identify the row locator on the Output Attributes page or tab, it is easier for you to match your output data to the input data.

  In the pivot operator, the row locator is an output attribute that corresponds to the repeated set of data from the source. For example, if the source data contains separate attributes for each month, create an output attribute 'MONTH' and designate it as the row locator.

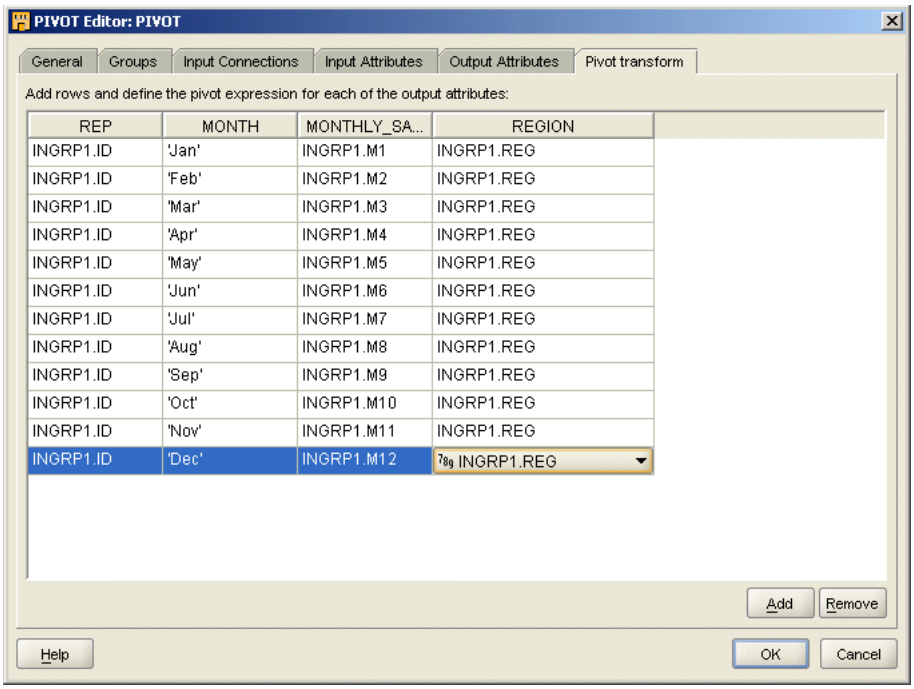
Pivot Transform

Use the Pivot Transform page to write expressions for each output attribute.

By default, Warehouse Builder displays two rows. Use **Add** to specify how many rows of output you want from a single row in the source. For example, if your source contains an attribute for each quarter in a year, you can specify 4 rows of output for each row in the source. If the source data contains an attribute for each month in the year, you can specify 12 rows of output for each row in the source.

Figure 26–19 shows the Pivot Transform tab with the pivot expressions defined for a source with an attribute for each month.

Figure 26–19 Pivot Transform Tab



Write pivot expressions based on the following types of output:

- **Row locator:** Specify a name for each row where the name is a value you want to load into the table. For example, if the row locator is 'MONTH', type 'Jan' for the first row.
- **Pivoted output data:** Select the appropriate expression from the list box. For example, for the row you define as 'Jan', select the expression that returns the set of values for January.
- **Attributes previously specified as keys:** Warehouse Builder defines the expression for you.
- **Unnecessary data:** If the Pivot Transform page contains data that you do not want as output, use the expression 'NULL'. Warehouse Builder outputs a repeated set of rows with no data for attributes you define as 'NULL'.

When using the wizard to create a new pivot operator, click **Finish** when you want to close the wizard. The Mapping Editor displays the operator you defined.

When using the Pivot Editor to edit an existing pivot operator, click **OK** when you have finished editing the operator. The Mapping Editor updates the operator with the changes you made.

Post-Mapping Process Operator

Use a Post-Mapping Process operator to define a procedure to be executed after running a PL/SQL mapping. For example, you can use a Post-Mapping Process operator to reenable and build indexes after a mapping completes successfully and loads data into the target.

The Post-Mapping Process operator calls a function or procedure that is defined in Warehouse Builder after the mapping is executed. The output parameter group provides the connection point for the returned value (if implemented through a

function) and the output parameters of the function or procedure. There are no restrictions on the connections of these output attributes

The Post-Mapping Process operator contains groups corresponding to the number and direction of the parameters associated with the selected PL/SQL procedure or function. This list of groups and attributes can only be modified through reconciliation.

You can only define one Post-Mapping Process operator for a mapping. If you want to run more than one procedure after a mapping, you must wrap the procedures into one procedure.

You can map constants, data generators, mapping input parameters, and output from a Pre-Mapping Process into a Post-Mapping Process operator. The Post-Mapping Process operator is not valid for an SQL*Loader mapping.

After you add a Post-Mapping Process operator to the Mapping Editor, use the operator properties dialog to specify run conditions in which to execute the process.

To use a Post-Mapping Process operator in a mapping:

1. Drag and drop a Post-Mapping Process operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Post-Mapping Process dialog.

2. Use the Add Post-Mapping Process dialog to select or create a transformation. For more information on how to use the Add Post-Mapping Process dialog, see "Adding Operators that Bind to Repository Objects" on page 6-12.

3. Connect the output attribute of a source operator to the input/output group of the Post-Mapping Process operator.

4. Set the run conditions for the operator.

To set run conditions for a Post-Mapping Process operator:

1. From the mapping canvas, select a Post-Mapping Process operator.

   The Post-Mapping Process Properties panel displays the properties of the Post-Mapping Process operator.

2. Click **Post-Mapping Process Run Condition** and select one of the following run conditions:

   Always: The process runs regardless of errors from the mapping.

   On Success: The process runs only if the mapping completes without errors.

   On Error: The process runs only if the mapping completes with errors exceeding the number of allowed errors set for the mapping.

   On Warning: The process runs only if the mapping completes with errors that are less than the number of allowed errors set for the mapping.

   If you select **On Error** or **On Warning** and the mapping runs in row based mode, you must verify the **Maximum Number of Errors** set for the mapping. To view the number of allowed errors, right-click the mapping in the Project Explorer, select **Configure,** and expand **Runtime Parameters.**

Pre-Mapping Process Operator

Use a Pre-Mapping Process operator to define a procedure to be executed before running a mapping.

For example, you can use a Pre-Mapping Process operator to truncate tables in a staging area before running a mapping that loads tables to that staging area. You can also use a Pre-Mapping Process operator to disable indexes before running a mapping that loads data to a target. You can then use a Post-Mapping Process operator to reenable and build the indexes after running the mapping that loads data to the target.

The Pre-Mapping Process operator calls a function or procedure whose metadata is defined in Warehouse Builder prior to executing a mapping. The output parameter group provides the connection point for the returned value (if implemented with a function) and the output parameters of the function or procedure. There are no restrictions on the connections of these output attributes.

When you drop a Pre-Mapping Process operator onto the Mapping Editor canvas, a dialog opens displaying the available libraries, categories, functions, and procedures. After you select a function or procedure from the tree, the operator displays with predefined input and output parameters.

The Pre-Mapping Process operator contains groups corresponding to the number and direction of the parameters associated with the selected PL/SQL procedure or function.

A mapping can only contain one Pre-Mapping Process operator. Only constants, mapping input parameters, and output from a Pre-Mapping Process can be mapped into a Post-Mapping Process operator.

After you add a Pre-Mapping Process operator to the Mapping Editor, use the operator property dialog to specify conditions in which to execute the mapping.

To use a Pre-Mapping Process operator in a mapping:

1. Drag and drop a **Pre-Mapping Process** operator onto the Mapping Editor canvas.

   The Add Pre-Mapping Process dialog is displayed.

2. Use the Add Pre-Mapping Process dialog to select or create a transformation. For more information on how to use this dialog, see "Adding Operators that Bind to Repository Objects" on page 6-12.

3. Connect the output attribute of the Pre-Mapping Process operator to the input group of a target operator.

4. Set the run conditions for the operator.

To set run conditions for a mapping with a Pre-Mapping Process operator:

1. In the mapping canvas, select the Pre-Mapping Process operator.

   The Pre-Mapping Process Properties panel displays the properties of the Pre-Mapping Process operator.

2. Click **Mapping Run Condition** and select one of the following run conditions:

   Always: Warehouse Builder runs the mapping after the process completes, regardless of the errors.

   On Success: Warehouse Builder runs the mapping only if the process completes without errors.

   On Error: Warehouse Builder runs the mapping only if the process completes with errors.

Set Operation Operator

Set operations combine the results of two component queries into a single result.

While a joiner combines separate rows into one row, set operators combine all data rows in their universal row. In set operators, although the data is added to one output, the column lists are not mixed together to form one combined column list.

The Set Operation operator enables you to use following set operations in a mapping:

- Union (default)
- Union All
- Intersect
- Minus

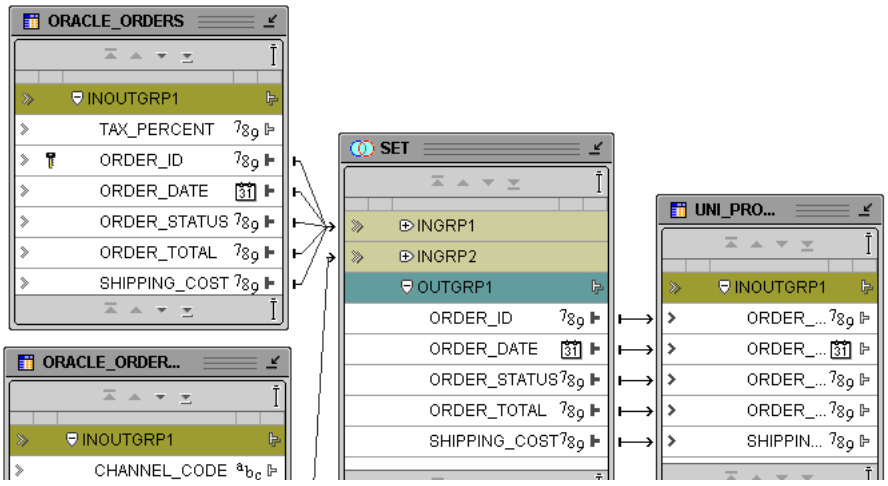
By default, the Set Operation operator contains two input groups and one output group. You can add input groups by using the operator editor. The number of attributes in the output group matches the number of attributes in the input group containing the most number of attributes.

To use the Set Operation operator, all sets must have the same number of attributes and the data types of corresponding attributes must match. Corresponding attributes are determined by the order of the attributes within an input group. For example, attribute 1 in input group 1 corresponds to attribute 1 in input group 2.

You must apply the set operation in top-down order. The order of the input groups determines the execution order of the set operation. This order only affects the minus operation. For example, A minus B is not the same as B minus A. The order of the attributes within the first input group determines the structure of a set. For example, {empno, ename} is not the same as {ename, empno}.

Figure 26–20 shows a mapping that uses the Set Operation operator. The data from the two source tables is combined using the Intersect set operation. The output of this operation is mapped to the target. The target table only contains the rows that are common to both the input tables.

Figure 26–20 Set Operation Operator in a Mapping



To use the Set Operation operator in a mapping:

1. Drag and drop a **Set Operation** operator onto the Mapping Editor canvas.
2. Connect source attributes to the Set Operation operator groups.
3. Select the Set operator header.

   The Set Operation Properties panel displays the properties of the Set operator.

4. Click the drop-down list on the **Set Operation** property and select an operation from the drop-down list.
5. Connect the Set Operation output group to a target input group.

Sorter Operator

You can produce a sorted row set using the Sorter operator.

The Sorter operator enables you to specify which input attributes are sorted and whether the sorting is performed in ascending or descending order. Warehouse Builder sorts data by placing an ORDER BY clause in the code generated by the mapping.

The Sorter operator has one input/output group. You can use the Sorter operator to sort data from any relational database source. You can place any operator after the Sorter operator.

Order By Clause

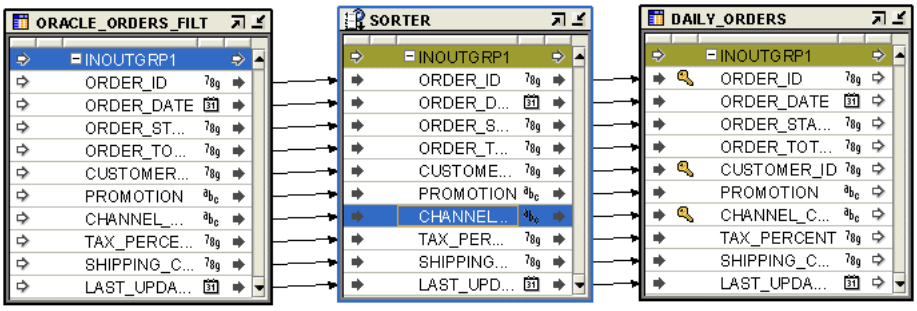
The Sorter operator contains the Order By clause. This clause is an ordered list of attributes in the input/output group to specify that sorting is performed in the same order as the ordered attribute list. You can set ascending or descending sorting for each attribute.

Most data in warehouses is loaded in batches. There can be some problems with the loading routines. For example, a batch of orders might contain a single order number multiple times with each order line representing a different state of the order. The order might have gone from status 'CREATED' to 'UPDATED' to 'BOOKED' during the day.

Because a SQL statement does not guarantee any ordering by default, the inserts and updates on the target table can take place in the wrong order. If the 'UPDATED' row is processed last, it becomes the final value for the day although the result should be status 'BOOKED'. Warehouse Builder enables you to solve this problem by creating an ordered extraction query using the Sorter operator. The ORDER BY clause can use the last updated attribute. This will ensure that the records appear in the order in which they were created.

Figure 26–21 shows a mapping that uses the Sorter operator to sort the records from the `ORACLE_ORDERS` table. Use the Order By Clause property of the Sorter operator to sort the input records on the `ORDER_ID` and the `LAST_UPDATED` attributes.

Figure 26–21 Sorter Operator in a Mapping



To use the Sorter operator in a mapping:

1. Drag and drop the **Sorter** operator onto the Mapping Editor canvas.
2. Connect a source operator group to the Sorter input/output group as shown in Figure 26–21.
3. Select the Sorter operator header.

   The Sorter Properties panel displays the properties of the operator.
4. Click the Ellipsis button in the Order By Clause field.

   The Order By Clause dialog is displayed.
5. Select the attributes you want to sort.

   Select an attribute from the Available Attributes list and click the right arrow button. Or, click the double right arrow button to select all of the Available Attributes.
6. Apply an ORDER BY clause to the attribute.

   Select the attribute in the ORDER BY Attributes list and select **ASC** (ascending) or **DESC** (descending) from the **ASC/DESC** drop-down list.
7. Click **OK.**
8. Connect the output of the Sorter operator to the target.

Splitter Operator

You can use the Splitter operator to split data from one source to several targets.

The Splitter operator splits a single input row set into several output row sets using a boolean split condition. Each output row set has a cardinality less than or equal to the input cardinality. This is useful when you want to move data to different targets based on a data driven condition. Instead of moving the data through multiple filters, you can use a splitter.

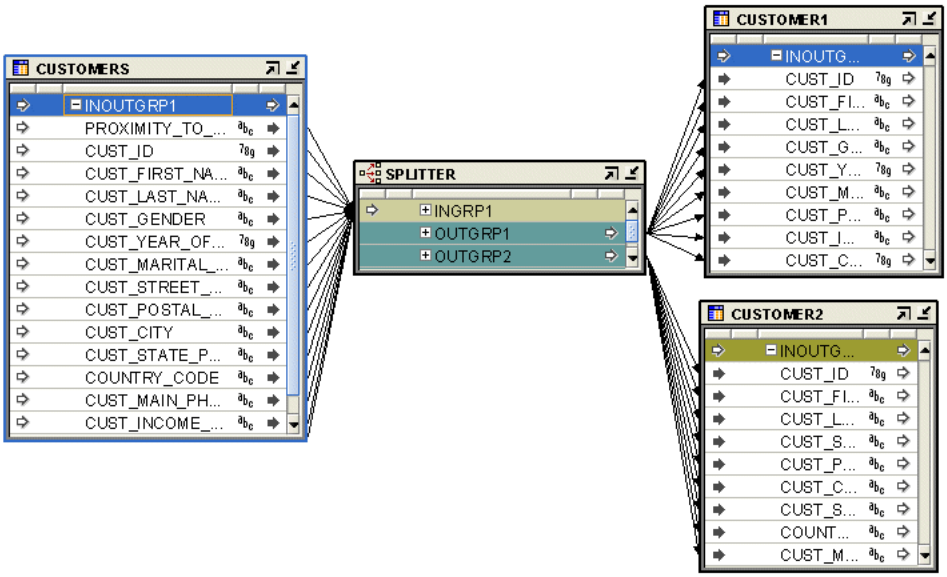
As an option, you can optimize mappings that split data from one source to multiple targets for improved performance. For more information, see "Example: Creating Mappings with Multiple Targets" on page 26-33.

The Splitter operator contains one input group and three output groups. The output groups are OUTGRP1, OUTGRP2, and REMAINING_ROWS. The output group REMAINING_ROWS contains all input rows that are not included in any output group. You can delete this output group, but you cannot edit it. You can create additional output groups, if required.

The Splitter Operator contains the split condition. For code generation, the source columns are substituted by the input attribute names in the expression template. The expression is a valid SQL expression that can be used in a WHERE clause.

Figure 26–22 shows the mapping that uses the Splitter operator to split customer data from the source table `CUSTOMERS` into two separate tables. One table contains only the customer addresses and the other table contains the remaining customer details. Use the Split Condition property of each output group in the Splitter operator to specify which data should be moved to a particular target table.

Figure 26–22 Splitter Operator in a Mapping



To use the Splitter operator in a mapping:

1. Drag and drop the **Splitter** operator onto the Mapping Editor canvas.
2. Connect a group from a source operator to the input group of the Splitter operator.

   The output attributes are created with data types matching the corresponding input data types.
3. Select the output group of the Splitter operator.

   The Group Properties panel displays the properties of the output group.
4. Click the Ellipsis button to the right of the Split Condition field.

   The Expression Builder dialog is displayed.
5. Define the split condition.

   For example, the split condition can be UPPER(INGRP1.OR_CHANNEL) = 'DIRECT'.
6. Define expressions for the split condition of each output group except the REMAINING ROWS group.
7. Connect the output groups to the targets.

Example: Creating Mappings with Multiple Targets

When you design a mapping with multiple targets, you have the option to optimize for improved performance. You may decide to not optimize if you require accurate auditing details for the mapping. If you decide to not optimize, Warehouse Builder generates separate insert statements for each target.

To optimize a multiple target mapping, you must take additional steps for Warehouse Builder to generate a single insert statement for all targets combined. In this case, Warehouse Builder generates a `multi_table_insert` SQL statement that takes advantage of parallel query and parallel DML services available in versions 9*i* and higher of the Oracle Database server.

To optimize a mapping with multiple targets:

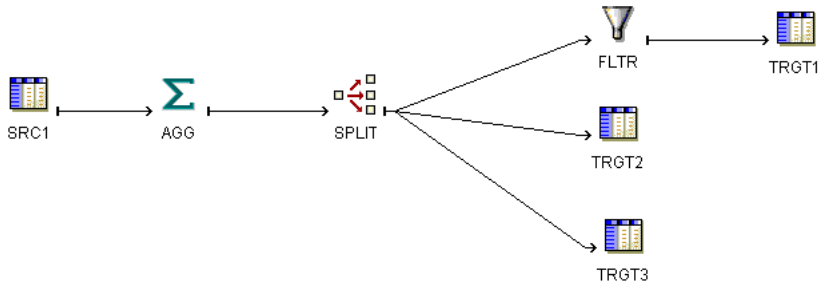
1. Define a mapping in an Oracle target module configured to generate Oracle9*i* or higher SQL.

   From Warehouse Builder, right-click the target module on the Project Explorer and select **Configure.** Under **Deployment System Type** and **PL/SQL Generation Mode,** select Oracle9*i* or higher.

2. In the Mapping Editor, design a mapping with a single source, a Splitter operator, and multiple targets.

   For Warehouse Builder to optimize the mapping, the targets must be tables, not views or materialized views. Each target table must have less than 999 columns. Between the Splitter operator and the targets, do not include any operators that change the cardinality. For example, you can place a Filter between the Splitter and the targets as shown in Figure 26–23, but not a Joiner or Aggregator operator. These restrictions only apply if you choose to optimize the mapping.

Figure 26–23 Example Mapping with Multiple Targets



3. From the Project Explorer, select the mapping and select **Design** from the menu bar, and select **Configure.** You can also right-click the mapping you want to configure and select **Configure.**

   Warehouse Builder displays the configuration properties dialog for a mapping.

4. Expand **Runtime Parameters** and set **Default Operating Mode** to set based.

5. Expand **Code Generation Options** and select **Optimized Code**.

When you run this mapping and view the generation results, Warehouse Builder returns one total SELECT and INSERT count for all targets.

Table Function Operator

Table function operators enable you to manipulate a set of input rows and return another set of rows of the same or different cardinality.

While a regular function only works on one row at a time, a table function enables you to apply the same complex PL/SQL logic on a set of rows and increase your performance. Unlike conventional functions, table functions can return a set of output rows that can be queried like a physical table.

The execution of the table function can also be parallelized where the returned rows are streamed directly to the next process without intermediate staging. Rows from a collection returned by a table function can also be pipelined or output one by one, as they are produced, instead of being output in a batch after processing of the entire table function input is completed.

Using table functions can greatly improve performance when loading your data warehouse.

To define a Table Function operator in a mapping:

Before you deploy the mapping containing the Table Function operator, you must manually create the table function in the target warehouse. The Table Function operator is bound to the actual table function object through the code generated by the mapping.

1. Drag and drop a **Table Function** operator onto the Mapping Editor canvas.

   A table function operator called TABLEFUNCTION is added to the Mapping Editor canvas.

2. Connect the appropriate source attributes to the input group of the table function operator.

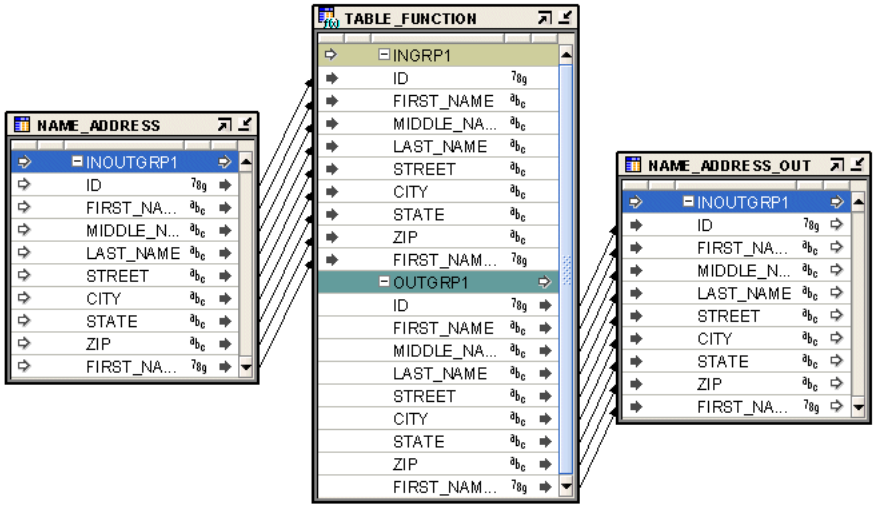
3. Right-click the Table Function operator and select **Open Details**.

   The Table Function Editor is displayed.

4. From the Groups tab, select **Add** to add an output group.

   Figure 26–24 shows a mapping that uses a Table Function operator to load data into a table.

Figure 26–24 Table Function Operator in a Mapping



Characteristics of Table Functions

- They do not support the passing of parameters by name.
- If the return type is TABLE of PLS Record, the name you select must match the name of PLS Record field. It is possible to select only one subset of the fields of the PLS Record in the select list.
- If the return type is TABLE of T1%ROWTYPE, the name you select must match the name of the columns of the table T1.
- If the return type is TABLE of Object Type, the name you select list must match the name of Object Type attribute.

- If the return type is TABLE of Scalar (like TABLE of NUMBER), only Select COLUMN_VALUE can be used to retrieve the scalar values returned by the table function.

Prerequisites for Using the Table Function Operator

Before you can use the Table Function operator in a mapping, create the table function in your target schema, external to Warehouse Builder. The table functions in the database that are supported by the unbound table function operator must meet the following requirements:

Input

- Ref Cursor returning PLS Record (the fields of the PLS Record) must be scalar data types supported by Warehouse Builder (0..n).
- There must be at least one input parameter.

Output

- PLS Record (the fields of the PLS Record should be scalar data types supported by Warehouse Builder).
- Object Type (the attributes of the Object Type should be scalar data types supported by Warehouse Builder).
- Scalar data types supported by Warehouse Builder.
- ROWTYPE

For a Table Function operator in a mapping:

- You must add one parameter group for each ref cursor type parameter.
- Multiple scalar parameters can be part of a single scalar type parameter group.
- The parameter groups and the parameters in a group can be entered in any order.
- The positioning of the parameters in the table function operator must be the same as the positioning of the parameters in the table function created in your target warehouse.

Table Function Operator Properties

You access the Table Function operator properties using the Properties panel of the Mapping Editor. The Properties panel displays the properties of the object selected on the canvas. For example, when you select the input group of the table function operator, the Properties panel displays the properties of the input parameter group.

Table Function Operator Properties

The Table Function operator has the following properties.

Table Function Name: Represents the name of the table function. The name specified here must match the actual name of the table function.

Table Function is Target: Select this option to indicate that the table function is a target. By default, this property is selected.

Input Parameter Group Properties

The table function operator accepts the following types of input parameters:

- **Input Parameter Type:** Valid input parameter types are REF_CURSOR_TYPE or SCALAR_TYPE.
- **REF_CURSOR_TYPE:** Returns a PLS Record { 0...N }. The fields of the PLS Record must be a scalar data type supported by Warehouse Builder.
- **SCALAR_TYPE:** Scalar data types supported by Warehouse Builder.
- **Parameter Position:** Indicates the position of the parameter in the table function signature corresponding to this parameter group. This property is applicable only to REF_CURSOR attribute groups and is used in conjunction with the scalar parameter positions.

Input Parameter Properties

- **Parameter Position:** The position of the parameter in the table function signature. This property is only applicable to scalar parameters.

Output Parameter Group Properties

- **Return Table of Scalar:** This property specifies whether the return of the table function is a TABLE of SCALAR or not. This information is required because the select list item for TABLE of SCALAR must be Select COLUMN_VALUE while in the other cases it should be an appropriate name.

Output Parameter

- **Type Attribute Name:** The name of the field of the PLS Record, attribute of the Object Type, or column of the ROWTYPE. This property is not applicable if the return type is TABLE of SCALAR. This name is used to invoke the table function.

Transformation Operator

You use the Transformation operator to transform the column value data of rows within a row set using a PL/SQL function, while preserving the cardinality of the input row set.

The Transformation operator must be bound to a function or procedure contained by one of the modules in the repository. The inputs and outputs of the Transformation operator correspond to the input and output parameters of the bound repository function or procedure. If the Transformation operator is bound to a function, a result output is added to the operator that corresponds to the result of the function. The bound function or procedure must be generated and deployed before the mapping can be deployed, unless the function or procedure already exists in the target system.

Warehouse Builder provides pre-defined PL/SQL library functions in the runtime schema that can be selected as a bound function when adding a Transformation operator onto a mapping. In addition, you can choose a function or procedure from the Global Shared Library.

The Transformation operator contains the following properties:

- **Function Call:** The text template for the function call that is generated by the code generator with the attribute names listed as the calling parameters. For the actual call, the attribute names are replaced with the actual source or target columns that are connected to the attributes.
- **Function Name:** The name of the function or procedure, to which this operator is bound.

- **Procedure:** A boolean value indicating, if true, that the bound transformation is a procedure rather than a function with no returned value.
- **Data Type:** Indicates the data type of the input, output, or result parameter of the bound function that corresponds to the given attribute. If the output of a mapping transformation is of CHAR data type, then Warehouse Builder applies an RTRIM on the result before moving the data to a target. This ensures that no extra spaces are contained in the output result.
- **Default Value:** The default value (blank if none) for the given attribute.
- **Optional Input:** A boolean value indicating, if true, that the given attribute is optional. If the attribute is optional, it need not be connected in the mapping.
- **Function Return:** A boolean value indicating, if true, that the given output attribute is the result attribute for the function. The result attribute is a named result. Use this property if another output is a named result, or if you change the name of the result output.

To use a Mapping Transformation operator in a mapping:

1. Drag and drop a **Mapping Transformation** operator onto the Mapping Editor canvas.

   The Add Mapping Transformation dialog is displayed.
2. Use the Add Mapping Transformation dialog to create a new transformation or select one or more transformations. For more information on these options, see "Adding Operators that Bind to Repository Objects" beginning on page 6-12.
3. Connect the source attributes to the inputs of the Mapping Transformation operator.
4. (Optional step) Right-click one of the inputs and select **Open Details.**

   The Attribute Properties panel displays the properties of the attribute.
5. Select an input attribute. If the Procedure property is set to True**,** then do not connect the input parameter.
6. Connect the Transformation operator output attributes to the target attributes.

Unpivot Operator

The unpivot operator converts multiple input rows into one output row.

The unpivot operator enables you to extract from a source once and produce one row from a set of source rows that are grouped by attributes in the source data. Like the pivot operator, the unpivot operator can be placed anywhere in a mapping.

Example: Unpivoting Sales Data

Table 26–3 shows a sample of data from the SALES relational table. In the crosstab format, the 'MONTH' column has 12 possible character values, one for each month of the year. All sales figures are contained in one column, 'MONTHLY_SALES'.

Table 26–3 Data in a Crosstab Format

REP	MONTH	MONTHLY_SALES	REGION
0675	Jan	10.5	4
0676	Jan	9.5	3

Table 26–3 (Cont.) Data in a Crosstab Format

REP	MONTH	MONTHLY_SALES	REGION
0679	Jan	8.7	3
0675	Feb	11.4	4
0676	Feb	10.5	3
0679	Feb	7.4	3
0675	Mar	9.5	4
0676	Mar	10.3	3
0679	Mar	7.5	3
0675	Apr	8.7	4
0676	Apr	7.6	3
0679	Apr	7.8	3

Figure 26–25 depicts data from the relational table 'SALES' after Warehouse Builder unpivoted the table. The data formerly contained in the 'MONTH' column (Jan, Feb, Mar...) corresponds to12 separate attributes (M1, M2, M3...). The sales figures formerly contained in the 'MONTHLY_SALES' are now distributed across the 12 attributes for each month.

Figure 26–25 Data Unpivoted from Crosstab Format

ID	Reg	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
0675	4	10.5	11.4	9.5	8.7	7.4	7.5	7.8	9.7				
0676	3	9.5	10.5	10.3	7.6	8.0	7.8	8.7	8.9				
0679	3	8.7	7.4	7.5	7.8	9.7	10.3	7.6	8.0				
0683	2	9.5	10.5	10.3	9.5	8.7	7.4	7.8	8.7				
0684	1	11.4	9.5	8.7	7.4	7.5	10.3	9.5	8.7				
0687	1	9.5	8.7	7.4	7.8	8.7	7.4	7.5	7.8				
0690	1	8.7	7.4	7.8	8.7	11.4	9.5	8.7	7.4				

The Row Locator

When you use the Unpivot operator, Warehouse Builder transforms multiple input rows into a single row based on the row locator. In the Unpivot operator, the row locator is an attribute that you must select from the source to correspond with a set of output attributes that you define. A row locator is required in an unpivot operator. In this example, the row locator is 'MONTH' from the 'SALES' table and it corresponds to attributes M1, M2, M3... M12 in the unpivoted output.

Using the Unpivot Operator

You have the following options for using an unpivot operator:

- **Define a new Unpivot operator:** Drag an Unpivot operator from the Palette onto the mapping. The Mapping Editor launches a wizard.
- **Edit an existing Unpivot operator:** Right-click the Unpivot operator and select **Open Details.** The Mapping Editor opens the Unpivot Editor.

Whether you are using the Unpivot Wizard or the Unpivot Editor, complete the following pages:

- General
- Groups

- Input Connections
- Input Attributes
- Row Locator
- Output Attributes
- Unpivot Transform

General

Use the General page to specify a name and optional description for the Unpivot operator. By default, the wizard names the operator "Unpivot".

Groups

Use the Groups page to specify one input and one output group.

In an Unpivot operator, the input group represents the source data in crosstab format. The output group represents the target data distributed across multiple attributes.

You can rename and add descriptions to the input and output groups. Since each Unpivot operator must have exactly one input and one output group, the wizard prevents you from adding or removing groups or changing group direction.

Input Connections

Use the Input Connections page to select attributes to copy and map into the unpivot operator.

To complete the Input connections page for an Unpivot operator:

1. Select complete groups or individual attributes from the left panel.

   To search for a specific attribute or group by name, type the text in **Search for** and click **Go.** To find the next match, click **Go** again.

   Hold the Shift key down to select multiple groups or attributes. If you want to select attributes from different groups, you must first combine the groups with a Joiner or Set operator.

2. Use the left to right arrow button in the middle of the page to move your selections to the right side of the wizard page.

   You can use the right to left arrow to move groups or attributes from the input connections list. Warehouse Builder removes the selection from the input group and removes the mapping line between the source operator and the unpivot operator.

Input Attributes

Use the Input Attributes page to modify the attributes you selected in the Input Connections tab or wizard page.

You can perform the following tasks from the Unpivot Input Attributes page:

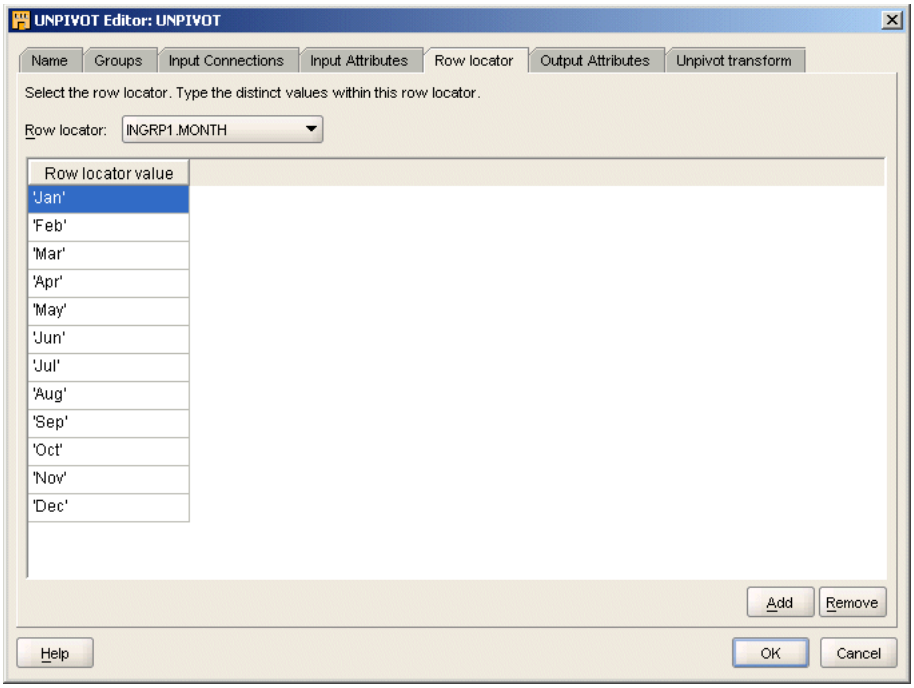
- **Add attributes:** Use the Add button to add input attributes.
- **Change attribute properties:** You can change the attribute name, data type, length, precision and scale.
- **Add an optional description:** Type a description for the input attributes.
- **Designate key attribute(s):** You must designate one or more key attributes for unpivot operators. Use the **Key** check box to indicate the attribute(s) that uniquely

identifies the input group. Input rows with the same value in their key attribute(s) produce one unpivoted output row.

Row Locator

Use the Row locator page to select a row locator and assign values to the distinct values contained in the row locator. Figure 26–26 shows the attribute MONTH selected as the row locator with values such as 'Jan', 'Feb', or 'Mar'.

Figure 26–26 Unpivot Row Locator Page



To complete the Unpivot Row Locator page:

1. Select an attribute from the Row locator list box.

   In the Unpivot operator, the row locator is the attribute from the source data that corresponds to a set of output attributes.

2. Use **Add** to specify the number of distinct values that exist in the row locator.

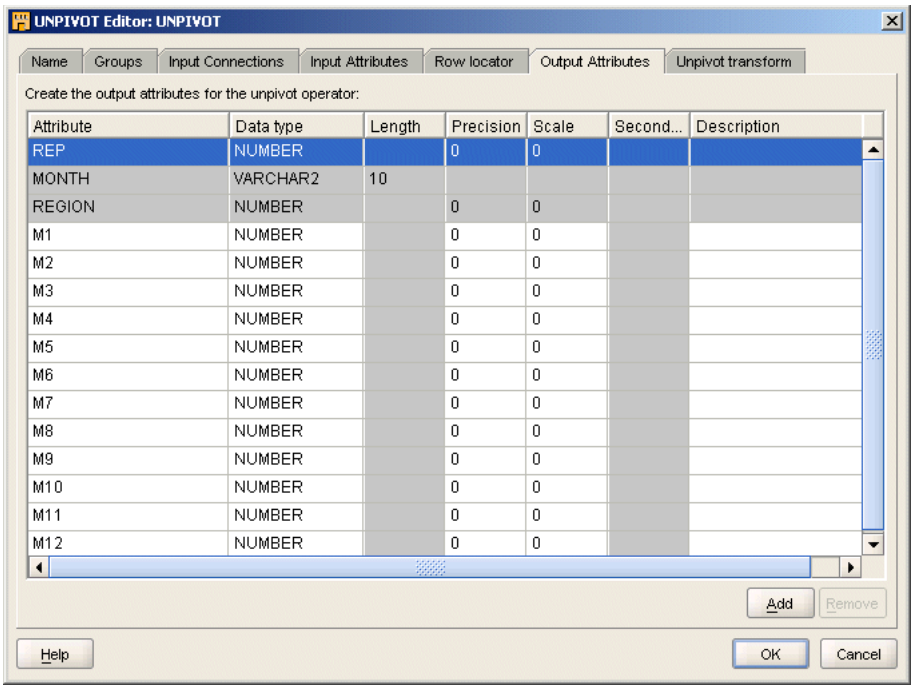
3. For each row locator value, type in the value as it appears in your source dataset.

   For string values, enclose the text in single quotes. For example, if the row locator is 'MONTH', there would be a total of 12 distinct values for that attribute. Click **Add** to add a row for each distinct value. For row locator values, type values exactly as they appear in the source dataset. For instance, the row locator values as shown in Table 26–3 are 'Jan', 'Feb', and 'Mar.'

Output Attributes

Use the Output Attributes page shown in Figure 26–27 to create the output attributes for the Unpivot operator.

Figure 26–27 Unpivot Output Attributes Page



If you designated any input attributes as keys on the Input Attributes tab or wizard page, Warehouse Builder displays those input attributes as output attributes that you cannot edit or remove.

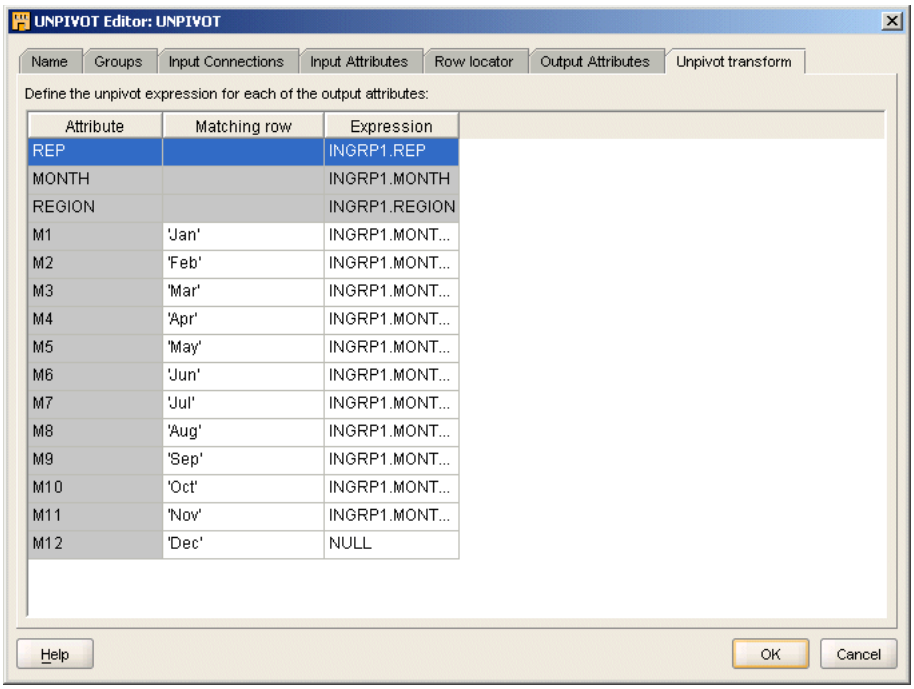
You can perform the following tasks from the Unpivot Output Attributes page:

- **Add attributes:** Use **Add** to increase the number of output attributes to accommodate the rows you specified on the Row locator tab or wizard page. If you specified 12 rows, specify 12 output attributes plus attributes for any other input attributes that you did not designate as a key.
- **Change attribute properties:** Except for attributes you designated as keys on the Input Attributes tab or wizard page, you can change the attribute name, data type, length, precision, and scale.
- **Add an optional description:** Type a description for the output attributes.

Unpivot Transform

Use the Unpivot Transform page shown in Figure 26–28 to write expressions for each output attribute.

Figure 26–28 Unpivot Transform Page



For attributes you designated as keys, Warehouse Builder defines the matching row and expression for you. Warehouse Builder displays the first row as the match for a key attribute. For all other output attributes, specify the matching row and the expression.

- **Matching row:** Select the appropriate option from the list box. For example, for the attribute you define as the first month of the year, 'M1', select 'Jan' from the list box.
- **Expression:** Select the appropriate expression from the list box. For all the new attributes you created to unpivot the data, select the same input attribute that contains the corresponding data. For example, the unpivot attributes M1, M2, M3... M12 would all share the same expression, INGRP1.MONTHLY_SALES. For all other output attributes, select the corresponding attribute from the list of input attributes.

27

Transformations

As you design mappings and process flows, you may want to use specialized transformations to transform data. This chapter describes all the predefined transformations provided by Warehouse Builder.

This chapter contains the following topics, each of which details all the predefined transformations in that category.

Administrative Transformations

Administrative transformations provide pre-built functionality to perform actions that are regularly performed in ETL processes. The main focus of these transformations is in the DBA related areas or to improve performance. For example, it is common to disable constraints when loading tables and then to re-enable them after loading has completed.

The administrative transformations in Warehouse Builder are custom functions. The Administrative transformation that Warehouse Builder provides are:

WB_ABORT

Syntax

```
WB_ABORT(p_code, p_message)
```

where *p_code* is the abort code, and must be between -20000 and -29999; and *p_message* is an abort message you specify.

Purpose

WB_ABORT enables you to abort the application from a Warehouse Builder component. You can run it from a post mapping process or as a transformation within a mapping.

Example

Use this administration function to abort an application. You can use this function in a post mapping process to abort deployment if there is an error in the mapping.

WB_COMPILE_PLSQL

Syntax

```
WB_COMPILE_PLSQL(p_name, p_type)
```

where *p_name* is the name of the object that is to be compiled; *p_type* is the type of object to be compiled. The legal types are:

```
'PACKAGE'
'PACKAGE BODY'
'PROCEDURE'
'FUNCTION'
'TRIGGER'
```

Purpose

This program unit compiles a stored object in the database.

Example

The following hypothetical example compiles the procedure called add_employee_proc:

```
EXECUTE WB_COMPILE_PLSQL('ADD_EMPLOYEE_PROC', 'PROCEDURE');
```

WB_DISABLE_ALL_CONSTRAINTS

Syntax

```
WB_DISABLE_ALL_CONSTRAINTS(p_name)
```

where *p_name* is the name of the table on which constraints are disabled.

Purpose

This program unit disables all constraints that are owned by the table as stated in the call to the program.

For faster loading of data sets, you can disable constraints on a table. The data is now loaded without validation. This is mainly done on relatively clean data sets.

Example

The following example shows the disabling of the constraints on the table OE.CUSTOMERS:

```
SELECT constraint_name
,      DECODE(constraint_type,'C','Check','P','Primary') Type
,      status
FROM user_constraints
WHERE table_name = 'CUSTOMERS';

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   ENABLED
CUST_LNAME_NN                  Check   ENABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   ENABLED
CUSTOMER_ID_MIN                Check   ENABLED
CUSTOMERS_PK                   Primary ENABLED
```

Perform the following in SQL*Plus or Warehouse Builder to disable all constraints:

```
EXECUTE WB_DISABLE_ALL_CONSTRAINTS('CUSTOMERS');

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   DISABLED
CUST_LNAME_NN                  Check   DISABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   DISABLED
CUSTOMER_ID_MIN                Check   DISABLED
CUSTOMERS_PK                   Primary DISABLED
```

Note: This statement uses a cascade option to allow dependencies to be broken by disabling the keys.

WB_DISABLE_ALL_TRIGGERS

Syntax

```
WB_DISABLE_ALL_TRIGGERS(p_name)
```

where *p_name* is the table name on which the triggers are disabled.

Purpose

This program unit disables all triggers owned by the table as stated in the call to the program. The owner of the table must be the current user (in variable USER). This action stops triggers and improves performance.

Example

The following example shows the disabling of all triggers on the table OE.OC_ORDERS:

```
SELECT trigger_name
,      status
FROM user_triggers
WHERE table_name = 'OC_ORDERS';

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     ENABLED
ORDERS_ITEMS_TRG               ENABLED
```

Perform the following in SQL*Plus or Warehouse Builder to disable all triggers on the table OC_ORDERS.

```
EXECUTE WB_DISABLE_ALL_TRIGGERS ('OC_ORDERS');

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     DISABLED
ORDERS_ITEMS_TRG               DISABLED
```

WB_DISABLE_CONSTRAINT

Syntax

```
WB_DISABLE_CONSTRAINT(p_constraintname, p_tablename)
```

where *p_constraintname* is the constraint name to be disabled; *p_tablename* is the table name on which the specified constraint is defined.

Purpose

This program unit disables the specified constraint that is owned by the table as stated in the call to the program. The user is the current user (in variable USER).

For faster loading of data sets, you can disable constraints on a table. The data is then loaded without validation. This reduces overhead and is mainly done on relatively clean data sets.

Example

The following example shows the disabling of the specified constraint on the table OE.CUSTOMERS:

```
SELECT constraint_name
, DECODE(constraint_type
, 'C', 'Check'
, 'P', 'Primary'
) Type
, status
FROM user_constraints
WHERE table_name = 'CUSTOMERS';

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   ENABLED
CUST_LNAME_NN                  Check   ENABLED
```

```
CUSTOMER_CREDIT_LIMIT_MAX      Check   ENABLED
CUSTOMER_ID_MIN                Check   ENABLED
CUSTOMERS_PK                   Primary ENABLED
```

Perform the following in SQL*Plus or Warehouse Builder to disable the specified constraint.

```
EXECUTE WB_DISABLE_CONSTRAINT('CUSTOMERS_PK','CUSTOMERS');

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   ENABLED
CUST_LNAME_NN                  Check   ENABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   ENABLED
CUSTOMER_ID_MIN                Check   ENABLED
CUSTOMERS_PK                   Primary DISABLED
```

Note: This statement uses a cascade option to allow dependencies to be broken by disabling the keys.

WB_DISABLE_TRIGGER

Syntax

```
WB_DISABLE_TRIGGER(p_name)
```

where *p_name* is the trigger name to be disabled.

Purpose

This program unit disables the specified trigger. The owner of the trigger must be the current user (in variable USER).

Example

The following example shows the disabling of a trigger on the table OE.OC_ORDERS:

```
SELECT trigger_name, status
FROM user_triggers
WHERE table_name = 'OC_ORDERS';

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     ENABLED
ORDERS_ITEMS_TRG               ENABLED
```

Perform the following in SQL*Plus or Warehouse Builder to disable the specified constraint.

```
ECECUTE WB_DISABLE_TRIGGER ('ORDERS_TRG');

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     DISABLED
ORDERS_ITEMS_TRG               ENABLED
```

WB_ENABLE_ALL_CONSTRAINTS

Syntax

```
WB_ENABLE_ALL_CONSTRAINTS(p_name)
```

where *p_name* is the name of the table for which all constraints should be enabled.

Purpose

This program unit enables all constraints that are owned by the table as stated in the call to the program.

For faster loading of data sets, you can disable constraints on a table. After the data is loaded, you must enable these constraints again using this program unit.

Example

The following example shows the enabling of the constraints on the table OE.CUSTOMERS:

```
SELECT constraint_name
, DECODE(constraint_type
, 'C', 'Check'
, 'P', 'Primary)
Type
, status
FROM user_constraints
WHERE table_name = 'CUSTOMERS';

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   DISABLED
CUST_LNAME_NN                  Check   DISABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   DISABLED
CUSTOMER_ID_MIN                Check   DISABLED
CUSTOMERS_PK                   Primary DISABLED
```

Perform the following in SQL*Plus or Warehouse Builder to enable all constraints.

```
EXECUTE WB_ENABLE_ALL_CONSTRAINTS('CUSTOMERS');

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   ENABLED
CUST_LNAME_NN                  Check   ENABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   ENABLED
CUSTOMER_ID_MIN                Check   ENABLED
CUSTOMERS_PK                   Primary ENABLED
```

WB_ENABLE_ALL_TRIGGERS

Syntax

```
WB_ENABLE_ALL_TRIGGERS(p_name)
```

where p_name is the table name on which the triggers are enabled.

Purpose

This program unit enables all triggers owned by the table as stated in the call to the program. The owner of the table must be the current user (in variable USER).

Example

The following example shows the enabling of all triggers on the table OE.OC_ORDERS:

```
SELECT trigger_name
,      status
FROM user_triggers
WHERE table_name = 'OC_ORDERS';

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     DISABLED
ORDERS_ITEMS_TRG               DISABLED
```

Perform the following in SQL*Plus or Warehouse Builder to enable all triggers defined on the table OE.OC_ORDERS.

```
EXECUTE WB_ENABLE_ALL_TRIGGERS ('OC_ORDERS');

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     ENABLED
ORDERS_ITEMS_TRG               ENABLED
```

WB_ENABLE_CONSTRAINT

Syntax

```
WB_ENABLE_CONSTRAINT(p_constraintname, p_tablename)
```

where *p_constraintname* is the constraint name to be disabled and *p_tablename* is the table name on which the specified constraint is defined.

Purpose

This program unit enables the specified constraint that is owned by the table as stated in the call to the program. The user is the current user (in variable USER). For faster loading of data sets, you can disable constraints on a table. After the loading is complete, you must re-enable these constraints. This program unit shows you how to enable the constraints one at a time.

Example

The following example shows the enabling of the specified constraint on the table OE.CUSTOMERS:

```
SELECT constraint_name
,      DECODE(constraint_type
       , 'C', 'Check'
       , 'P', 'Primary'
       ) Type
,      status
FROM user_constraints
WHERE table_name = 'CUSTOMERS';
```

```
CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   DISABLED
CUST_LNAME_NN                  Check   DISABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   DISABLED
CUSTOMER_ID_MIN                Check   DISABLED
CUSTOMERS_PK                   Primary DISABLED
```

Perform the following in SQL*Plus or Warehouse Builder to enable the specified constraint.

```
EXECUTE WB_ENABLE_CONSTRAINT('CUSTOMERS_PK', 'CUSTOMERS');

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   DISABLED
CUST_LNAME_NN                  Check   DISABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   DISABLED
CUSTOMER_ID_MIN                Check   DISABLED
CUSTOMERS_PK                   Primary ENABLED
```

WB_ENABLE_TRIGGER

Syntax

```
WB_ENABLE_TRIGGER(p_name)
```

where `p_name` is the trigger name to be enabled.

Purpose

This program unit enables the specified trigger. The owner of the trigger must be the current user (in variable `USER`).

Example

The following example shows the enabling of a trigger on the table `OE.OC_ORDERS`:

```
SELECT trigger_name
,      status
FROM user_triggers
WHERE table_name = 'OC_ORDERS';

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     DISABLED
ORDERS_ITEMS_TRG               ENABLED
```

Perform the following in SQL*Plus or Warehouse Builder to enable the specified constraint.

```
EXECUTE WB_ENABLE_TRIGGER ('ORDERS_TRG');

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     ENABLED
ORDERS_ITEMS_TRG               ENABLED
```

WB_TRUNCATE_TABLE

Syntax

```
WB_TRUNCATE_TABLE(p_name)
```

where *p_name* is the table name to be truncated.

Purpose

This program unit truncates the table specified in the command call. The owner of the trigger must be the current user (in variable USER). The command disables and re-enables all referencing constraints to enable the truncate table command. Use this command in a pre-mapping process to explicitly truncate a staging table and ensure that all data in this staging table is newly loaded data.

Example

The following example shows the truncation of the table OE.OC_ORDERS:

```
SELECT COUNT(*) FROM oc_orders;

  COUNT(*)
----------
       105
```

Perform the following in SQL*Plus or Warehouse Builder to enable the specified constraint.

```
EXECUTE WB_TRUNCATE_TABLE ('OC_ORDERS');

  COUNT(*)
----------
         0
```

Character Transformations

Character transformations enable Warehouse Builder users to perform transformations on Character objects. The custom functions provided with Warehouse Builder are prefixed with WB_.

The character transformations available in Warehouse Builder are:

ASCII

Syntax

```
ascii::=ASCII(attribute)
```

Purpose

ASCII returns the decimal representation in the database character set of the first character of `attribute`. An `attribute` can be of data type CHAR, VARCHAR2, NCHAR, or NVARCHAR2. The value returned is of data type NUMBER. If your database character set is 7-bit ASCII, this function returns an ASCII value. If your database character set is EBCDIC Code, this function returns an EBCDIC value. There is no corresponding EBCDIC character function.

Example

The following example returns the ASCII decimal equivalent of the letter Q:

```
SELECT ASCII('Q') FROM DUAL;
ASCII('Q')
----------
81
```

CHR

Syntax

```
chr::=CHR(attribute)
```

Purpose

CHR returns the character with the binary equivalent to the number specified in the attribute in either the database character set or the national character set.

If USING NCHAR_CS is not specified, this function returns the character with the binary equivalent to attribute as a VARCHAR2 value in the database character set. If USING NCHAR_CS is specified in the expression builder, this function returns the character with the binary equivalent to attribute as a NVARCHAR2 value in the national character set.

Examples

The following example is run on an ASCII-based machine with the database character set defined as WE8ISO8859P1:

```
SELECT CHR(67)||CHR(65)||CHR(84) "Dog"
   FROM DUAL;

Dog
---
CAT
```

To produce the same results on an EBCDIC-based machine with the WE8EBCDIC1047 character set, modify the preceding example as follows:

```
SELECT CHR(195)||CHR(193)||CHR(227) "Dog"
   FROM DUAL;

Dog
---
CAT
```

The following example uses the UTF8 character set:

```
SELECT CHR (50052 USING NCHAR_CS)
   FROM DUAL;
CH
--
Ä
```

CONCAT

Syntax

```
concat::=CONCAT(attribute1, attribute2)
```

Purpose

CONCAT returns attribute1 concatenated with attribute2. Both attribute1 and attribute2 can be CHAR or VARCHAR2 data types. The returned string is of VARCHAR2 data type contained in the same character set as attribute1. This function is equivalent to the concatenation operator (||).

Example

This example uses nesting to concatenate three character strings:

```
SELECT CONCAT(CONCAT(last_name, '''s job category is '), job_id) "Job"
   FROM employees
   WHERE employee_id = 152;
```

```
Job
-----------------------------------------------------
Hall's job category is SA_REP
```

INITCAP

Syntax

```
initcap::=INITCAP(attribute)
```

Purpose

INITCAP returns the content of the attribute with the first letter of each word in uppercase and all other letters in lowercase. Words are delimited by white space or by characters that are not alphanumeric. Attribute can be of the data types CHAR or VARCHAR2. The return value is the same data type as attribute.

Example

The following example capitalizes each word in the string:

```
SELECT INITCAP('the soap') "Capitals" FROM DUAL;

Capitals
---------
The Soap
```

INSTR, INSTR2, INSTR4, INSTRB, INSTRC

Syntax

```
instr::=INSTR(attribute1, attribute2, n, m)
instr2::=INSTR2(attribute1, attribute2, n, m)
instr4::=INSTR4(attribute1, attribute2, n, m)
instrb::=INSTRB(attribute1, attribute2, n, m)
instrc::=INSTRC(attribute1, attribute2, n, m)
```

Purpose

INSTR searches attribute1 beginning with its nth character for the mth occurrence of attribute2. It returns the position of the character in attribute1 that is the first character of this occurrence. INSTRB uses bytes instead of characters. INSTRC uses Unicode complete characters. INSTR2 UCS2 code points. INSTR4 uses UCS4 code points.

If n is negative, Oracle counts and searches backward from the end of attribute1. The value of m must be positive. The default values of both n and m are 1, which means that Oracle begins searching the first character of attribute1 for the first occurrence of attribute2. The return value is relative to the beginning of attribute1, regardless of the value of n, and is expressed in characters. If the search is unsuccessful (if attribute2 does not appear m times after the nth character of attribute1), then the return value is 0.

Examples

The following example searches the string "CORPORATE FLOOR", beginning with the third character, for the string "OR". It returns the position in CORPORATE FLOOR at which the second occurrence of "OR" begins:

```
SELECT INSTR('CORPORATE FLOOR','OR', 3, 2) "Instring"
   FROM DUAL;

Instring
----------
14
```

The next example begins searching at the third character from the end:

```
SELECT INSTR('CORPORATE FLOOR','OR', -3, 2) "Reversed Instring"
   FROM DUAL;

Reversed Instring
-----------------
2
```

This example assumes a double-byte database character set.

```
SELECT INSTRB('CORPORATE FLOOR','OR',5,2) "Instring in bytes"
   FROM DUAL;

Instring in bytes
-----------------
27
```

LENGTH, LENGTH2, LENGTH4, LENGTHB, LENGTHC

Syntax

```
length::=LENGTH(attribute)
length2::=LENGTH2(attribute)
length4::=LENGTH4(attribute)
lengthb::=LENGTHB(attribute)
lengthC::=LENGTHC(attribute)
```

Purpose

The length functions return the length of `attribute`, which can be of the data types `CHAR` or `VARCHAR2`. `LENGTH` calculates the length using characters as defined by the input character set. `LENGTHB` uses bytes instead of characters. `LENGTHC` uses Unicode complete characters. `LENGTH2` uses UCS2 code points. `LENGTH4` uses US4 code points. The return value is of data type NUMBER. If `attribute` has data type CHAR, the length includes all trailing blanks. If `attribute` contains a null value, this function returns null.

Example

The following examples use the `LENGTH` function using single- and multibyte database character set.

```
SELECT LENGTH('CANDIDE') "Length in characters"
   FROM DUAL;

Length in characters
--------------------
7
```

This example assumes a double-byte database character set.

```
SELECT LENGTHB ('CANDIDE') "Length in bytes"
   FROM DUAL;

Length in bytes
---------------
14
```

LOWER

Syntax

```
lower::=LOWER(attribute)
```

Purpose

LOWER returns attribute, with all letters in lowercase. The attribute can be of the data types CHAR and VARCHAR2. The return value is the same data type as that of attribute.

Example

The following example returns a string in lowercase:

```
SELECT LOWER('MR. SCOTT MCMILLAN') "Lowercase"
   FROM DUAL;

Lowercase
--------------------
mr. scott mcmillan
```

LPAD

Syntax

```
lpad::=LPAD(attribute1, n, attribute2)
```

Purpose

LPAD returns attribute1, left-padded to length n with the sequence of characters in attribute2. Attribute2 defaults to a single blank. If attribute1 is longer than n, this function returns the portion of attribute1 that fits in n.

Both attribute1 and attribute2 can be of the data types CHAR and VARCHAR2. The string returned is of VARCHAR2 data type and is in the same character set as attribute1. The argument n is the total length of the return value as it is displayed on your screen. In most character sets, this is also the number of characters in the return value. However, in some multibyte character sets, the display length of a character string can differ from the number of characters in the string.

Example

The following example left-pads a string with the characters "*.":

```
SELECT LPAD('Page 1',15,'*.') "LPAD example"
   FROM DUAL;

LPAD example
---------------
*.*.*.*.*Page 1
```

LTRIM

Syntax

```
ltrim::=LTRIM(attribute, set)
```

Purpose

LTRIM removes characters from the left of attribute, with all the left most characters that appear in set removed. Set defaults to a single blank. If attribute is a character literal, you must enclose it in single quotes. Warehouse Builder begins scanning attribute from its first character and removes all characters that appear in set until it reaches a character absent in set. Then it returns the result.

Both attribute and set can be any of the data types CHAR and VARCHAR2. The string returned is of VARCHAR2 data type and is in the same character set as attribute.

Example

The following example trims all of the left-most x's and y's from a string:

```
SELECT LTRIM('xyxXxyLAST WORD','xy') "LTRIM example"
   FROM DUAL;

LTRIM example
------------
XxyLAST WORD
```

NLSSORT

Syntax

```
nlssort::=NLSSORT(attribute, nlsparam)
```

Purpose

NLSSORT returns the string of bytes used to sort attribute. The parameter attribute is of type VARCHAR2. Use this function to compare based on a linguistic sort of sequence rather than on the binary value of a string.

The value of nlsparam can have the form 'NLS_SORT = sort' where sort is a linguistic sort sequence or BINARY. If you omit nlsparam, this function uses the default sort sequence for your session.

Example

The following example creates a table containing two values and shows how the values returned can be ordered by the NLSSORT function:

```
CREATE TABLE test (name VARCHAR2(15));
INSERT INTO TEST VALUES ('Gaardiner');
INSERT INTO TEST VALUES ('Gaberd');

SELECT * FROM test ORDER BY name;

NAME
------
Gaardiner
Gaberd
```

```
SELECT *
   FROM test
   ORDER BY NLSSORT(name, 'NLSSORT = XDanish');

Name
------
Gaberd
Gaardiner
```

NLS_INITCAP

Syntax

```
nls_initcap::=NLS_INITCAP(attribute, nlsparam)
```

Purpose

NLS_INITCAP returns attribute, with the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

The value of nlsparam can have the form 'NLS_SORT = sort', where sort is either a linguistic sort sequence or BINARY. The linguistic sort sequence handles special linguistic requirements for case conversions. These requirements can result in a return value of a different length than the char. If you omit 'nlsparam', this function uses the default sort sequence for your session.

Example

The following examples show how the linguistic sort sequence results in a different return value from the function:

```
SELECT NLS_INITCAP('ijsland') "InitCap"
   FROM dual;

InitCap
---------
Ijsland

SELECT NLS_INITCAP('ijsland','NLS_SORT=XDutch) "InitCap"
   FROM dual;

InitCap
---------
IJsland
```

NLS_LOWER

Syntax

```
nls_lower::=NLS_LOWER(attribute, nlsparam)
```

Purpose

NLS_LOWER returns attribute, with all letters lowercase. Both attribute and nlsparam can be any of the data types CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. The string returned is of data type VARCHAR2 and is in the same character set as attribute. The value of nlsparam can have the form 'NLS_SORT = sort', where sort is either a linguistic sort sequence or BINARY.

Example

The following example returns the character string `'citta''` using the XGerman linguistic sort sequence:

```
SELECT NLS_LOWER('CITTA''','NLS_SORT=XGerman) "Lowercase"
   FROM DUAL;

Lowercase
------------
citta'
```

NLS_UPPER

Syntax

```
nls_upper::=NLS_UPPER(attribute, nlsparam)
```

Purpose

`NLS_UPPER` returns `attribute`, with all letters uppercase. Both `attribute` and `nlsparam` can be any of the data types `CHAR`, `VARCHAR2`, `NCHAR`, `NVARCHAR2`, `CLOB`, or `NCLOB`. The string returned is of VARCHAR2 data type and is in the same character set as `attribute`. The value of `nlsparam` can have the form `'NLS_SORT = sort'`, where sort is either a linguistic sort sequence or BINARY.

Example

The following example returns a string with all letters converted to uppercase:

```
SELECT NLS_UPPER('große') "Uppercase"
   FROM DUAL;

Uppercase
------------
GROßE

SELECT NLS_UPPER('große', 'NLS_SORT=XGerman) "Uppercase"
   FROM DUAL;

Uppercase
------------
GROSSE
```

REPLACE

Syntax

```
replace::=REPLACE(attribute, 'search_string', 'replacement_string')
```

Purpose

`REPLACE` returns an `attribute` with every occurrence of `search_string` replaced with `replacement_string`. If `replacement_string` is omitted or null, all occurrences of `search_string` are removed. If `search_string` is null, `attribute` is returned.

Both search_string and replacement_string, as well as attribute, can be of the data types CHAR or VARCHAR2. The string returned is of VARCHAR2 data type and is in the same character set as attribute.

This function provides a superset of the functionality provided by the TRANSLATE function. TRANSLATE provides single-character, one-to-one substitution. REPLACE enables you to substitute one string for another, as well as to remove character strings.

Example

The following example replaces occurrences of "J" with "BL":

```
SELECT REPLACE('JACK and JUE','J','BL') "Changes"
   FROM DUAL;

Changes
--------------
BLACK and BLUE
```

REGEXP_INSTR

Syntax

```
regexp_instr:=REGEXP_INSTR(source_string, pattern, position, occurance,
                           return_option, match_parameter)
```

Purpose

REGEXP_INSTR extends the functionality of the INSTR function by letting you search a string for a regular expression pattern. The function evaluates strings using characters as defined by the input character set. It returns an integer indicating the beginning or ending position of the matched substring, depending on the value of the return_option argument. If no match is found, the function returns 0.

This function complies with the POSIX regular expression standard and the Unicode Regular Expression Guidelines.

- source_string is a character expression that serves as the search value. It is commonly a character column and can be of any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB.
- pattern is the regular expression. It is usually a text literal and can be of any of the datatypes CHAR, VARCHAR2, NCHAR, or NVARCHAR2. It can contain up to 512 bytes. If the datatype of pattern is different from the datatype of source_ string, Oracle Database converts pattern to the datatype of source_ string. For a listing of the operators you can specify in pattern, please refer to Appendix C, "Oracle Regular Expression Support".
- position is a positive integer indicating the character of source_string where Oracle should begin the search. The default is 1, meaning that Oracle begins the search at the first character of source_string.
- occurrence is a positive integer indicating which occurrence of pattern in source_string Oracle should search for. The default is 1, meaning that Oracle searches for the first occurrence of pattern.
- return_option lets you specify what Oracle should return in relation to the occurrence:

  - If you specify 0, then Oracle returns the position of the first character of the occurrence. This is the default.
  - If you specify 1, then Oracle returns the position of the character following the occurrence.

- `match_parameter` is a text literal that lets you change the default matching behavior of the function. You can specify one or more of the following values for match_parameter:
  - 'i' specifies case-insensitive matching.
  - 'c' specifies case-sensitive matching.
  - 'n' allows the period (.), which is the match-any-character character, to match the newline character. If you omit this parameter, the period does not match the newline character.
  - 'm' treats the source string as multiple lines. Oracle interprets ^ and $ as the start and end, respectively, of any line anywhere in the source string, rather than only at the start or end of the entire source string. If you omit this parameter, Oracle treats the source string as a single line.

  If you specify multiple contradictory values, Oracle uses the last value. For example, if you specify 'ic', then Oracle uses case-sensitive matching. If you specify a character other than those shown above, then Oracle returns an error.

  If you omit `match_parameter`, then:
  - The default case sensitivity is determined by the value of the NLS_SORT parameter.
  - A period (.) does not match the newline character.
  - The source string is treated as a single line.

Example

The following example examines the string, looking for occurrences of one or more non-blank characters. Oracle begins searching at the first character in the string and returns the starting position (default) of the sixth occurrence of one or more non-blank characters.

```
SELECT
     REGEXP_INSTR('500 Oracle Parkway, Redwood Shores, CA', '[^ ]+', 1, 6) FROM
DUAL;

REGEXP_INSTR
------------
37
```

The following example examines the string, looking for occurrences of words beginning with s, r, or p, regardless of case, followed by any six alphabetic characters. Oracle begins searching at the third character in the string and returns the position in the string of the character following the second occurrence of a seven letter word beginning with s, r, or p, regardless of case.

```
SELECT
     REGEXP_INSTR('500 Oracle Parkway, Redwood Shores, CA',
     '[s|r|p][[:alpha:]]{ 6 }', 3, 2, 1, 'i')
   FROM DUAL;

REGEXP_INSTR
------------
```

```
28
```

REGEXP_REPLACE

Syntax

```
regexp_replace:=REGEXP_REPLACE(source_string, pattern, replace_string,
                               position, occurance, match_parameter)
```

Purpose

REGEXP_REPLACE extends the functionality of the REPLACE function by letting you search a string for a regular expression pattern. By default, the function returns source_string with every occurrence of the regular expression pattern replaced with `replace_string`. The string returned is n the same character set as source_ string. The function returns VARCHAR2 if the first argument is not a LOB and returns CLOB if the first argument is a LOB.

This function complies with the POSIX regular expression standard and the Unicode Regular Expression Guidelines.

- `source_string` is a character expression that serves as the search value. It is commonly a character column and can be of any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB or NCLOB.
- `pattern` is the regular expression. It is usually a text literal and can be of any of the datatypes CHAR, VARCHAR2, NCHAR, or NVARCHAR2. It can contain up to 512 bytes. If the datatype of pattern is different from the datatype of source_ string, Oracle Database converts pattern to the datatype of source_ string.
- `replace_string` can be of any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. If replace_string is a CLOB or NCLOB, then Oracle truncates replace_string to 32K. The replace_string can contain up to 500 backreferences to subexpressions in the form \n, where n is a number from 1 to 9. If n is the blackslash character in replace_string, then you must precede it with the escape character (\\).
- `position` is a positive integer indicating the character of source_string where Oracle should begin the search. The default is 1, meaning that Oracle begins the search at the first character of source_string.
- `occurrence` is a nonnegative integer indicating the occurrence of the replace operation:
  - If you specify 0, then Oracle replaces all occurrences of the match.
  - If you specify a positive integer n, then Oracle replaces the nth occurrence.
- `match_parameter` is a text literal that lets you change the default matching behavior of the function. This argument affects only the matching process and has no effect on `replace_string`. You can specify one or more of the following values for `match_parameter`:
  - 'i' specifies case-insensitive matching.
  - 'c' specifies case-sensitive matching.
  - 'n' allows the period (.), which is the match-any-character character, to match the newline character. If you omit this parameter, the period does not match the newline character.

– 'm' treats the source string as multiple lines. Oracle interprets ^ and $ as the start and end, respectively, of any line anywhere in the source string, rather than only at the start or end of the entire source string. If you omit this parameter, Oracle treats the source string as a single line.

If you specify multiple contradictory values, Oracle uses the last value. For example, if you specify 'ic', then Oracle uses case-sensitive matching. If you specify a character other than those shown above, then Oracle returns an error. If you omit `match_parameter`, then:

– The default case sensitivity is determined by the value of the NLS_SORT parameter.

– A period (.) does not match the newline character.

– The source string is treated as a single line.

Example

The following example examines `phone_number`, looking for the pattern `xxx.xxx.xxxx`. Oracle reformats this pattern with `(xxx) xxx-xxxx`.

```
SELECT
     REGEXP_REPLACE(phone_number,
       '([[:digit:]]{ 3 })\.([[:digit:]]{ 3 })\.([[:digit:]]{ 4 })',
       '(\1) \2-\3')
FROM employees;

REGEXP_REPLACE
--------------------------------------------------------------------------------
(515) 123-4567
(515) 123-4568
(515) 123-4569
(590) 423-4567
. . .
```

The following example examines `country_name`. Oracle puts a space after each non-null character in the string.

```
SELECT
     REGEXP_REPLACE(country_name, '(.)', '\1 ') "REGEXP_REPLACE"
FROM countries;

REGEXP_REPLACE
--------------------------------------------------------------------------------
A r g e n t i n a
A u s t r a l i a
B e l g i u m
B r a z i l
C a n a d a
. . .
```

The following example examines the string, looking for two or more spaces. Oracle replaces each occurrence of two or more spaces with a single space.

```
SELECT
     REGEXP_REPLACE('500 Oracle Parkway, Redwood Shores, CA','( ){ 2, }', ' ')
FROM DUAL;

REGEXP_REPLACE
-----------------------------------
500 Oracle Parkway, Redwood Shores, CA
```

REGEXP_SUBSTR

Syntax

```
regexp_substr:=REGEXP_SUBSTR(source_string, pattern, position,
                             occurance, match_parameter)
```

Purpose

REGEXP_SUBSTR extends the functionality of the SUBSTR function by letting you search a string for a regular expression pattern. It is also similar to REGEXP_INSTR, but instead of returning the position of the substring, it returns the substring itself. This function is useful if you need the contents of a match string but not its position in the source string. The function returns the string as VARCHAR2 or CLOB data in the same character set as source_string.

This function complies with the POSIX regular expression standard and the Unicode Regular Expression Guidelines.

- `source_string` is a character expression that serves as the search value. It is commonly a character column and can be of any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB or NCLOB.
- `pattern` is the regular expression. It is usually a text literal and can be of any of the datatypes CHAR, VARCHAR2, NCHAR, or NVARCHAR2. It can contain up to 512 bytes. If the datatype of pattern is different from the datatype of source_ string, Oracle Database converts pattern to the datatype of source_ string.
- `replace_string` can be of any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. If replace_string is a CLOB or NCLOB, then Oracle truncates replace_string to 32K. The replace_string can contain up to 500 backreferences to subexpressions in the form \n, where n is a number from 1 to 9. If n is the blackslash character in replace_string, then you must precede it with the escape character (\\).
- `position` is a positive integer indicating the character of source_string where Oracle should begin the search. The default is 1, meaning that Oracle begins the search at the first character of source_string.
- `occurrence` is a nonnegative integer indicating the occurrence of the replace operation:
  - If you specify 0, then Oracle replaces all occurrences of the match.
  - If you specify a positive integer n, then Oracle replaces the nth occurrence.
- `match_parameter` is a text literal that lets you change the default matching behavior of the function. This argument affects only the matching process and has no effect on replace_string. You can specify one or more of the following values for match_parameter:
  - 'i' specifies case-insensitive matching.
  - 'c' specifies case-sensitive matching.
  - 'n' allows the period (.), which is the match-any-character character, to match the newline character. If you omit this parameter, the period does not match the newline character.

– 'm' treats the source string as multiple lines. Oracle interprets ^ and $ as the start and end, respectively, of any line anywhere in the source string, rather than only at the start or end of the entire source string. If you omit this parameter, Oracle treats the source string as a single line.

If you specify multiple contradictory values, Oracle uses the last value. For example, if you specify 'ic', then Oracle uses case-sensitive matching. If you specify a character other than those shown above, then Oracle returns an error. If you omit match_parameter, then:

– The default case sensitivity is determined by the value of the NLS_SORT parameter.

– A period (.) does not match the newline character.

– The source string is treated as a single line.

Example

The following example examines the string, looking for the first substring bounded by commas. Oracle Database searches for a comma followed by one or more occurrences of non-comma characters followed by a comma. Oracle returns the substring, including the leading and trailing commas.

```
SELECT
     REGEXP_SUBSTR('500 Oracle Parkway, Redwood Shores, CA',',[^,]+,')
FROM DUAL;

REGEXPR_SUBSTR
-----------------
, Redwood Shores,
```

The following example examines the string, looking for http:// followed by a substring of one or more alphanumeric characters and optionally, a period (.). Oracle searches for a minimum of three and a maximum of four occurrences of this substring between http:// and either a slash (/) or the end of the string.

```
SELECT
     REGEXP_SUBSTR('http://www.oracle.com/products',
                   'http://([[:alnum:]]+\.?){ 3,4 } /?')
FROM DUAL;

REGEXP_SUBSTR
----------------------
http://www.oracle.com/
```

RPAD

Syntax

```
rpad::=RPAD(attribute1, n, attribute2)
```

Purpose

`RPAD` returns `attribute1`, right-padded to length `n` with `attribute2`, replicated as many times as necessary. `Attribute2` defaults to a single blank. If `attribute1` is longer than `n`, this function returns the portion of `attribute1` that fits in `n`.

Both attribute1 and attribute2 can be of the data types CHAR or VARCHAR2. The string returned is of VARCHAR2 data type and is in the same character set as attribute1.

The argument n is the total length of the return value as it is displayed on your screen. In most character sets, this is also the number of characters in the return value. However, in some multibyte character sets, the display length of a character string can differ from the number of characters in the string.

Example

The following example rights-pads a name with the letters "ab" until it is 12 characters long:

```
SELECT RPAD('MORRISON',12,'ab') "RPAD example"
   FROM DUAL;

RPAD example
-----------------
MORRISONabab
```

RTRIM

Syntax

```
rtrim::=RTRIM(attribute, set)
```

Purpose

RTRIM returns attribute, with all the right most characters that appear in set removed; set defaults to a single blank. If attribute is a character literal, you must enclose it in single quotes. RTRIM works similarly to LTRIM. Both attribute and set can be any of the data types CHAR or VARCHAR2. The string returned is of VARCHAR2 data type and is in the same character set as attribute.

Example

The following example trims the letters "xy" from the right side of a string:

```
SELECT RTRIM('BROWNINGyxXxy','xy') "RTRIM e.g."
   FROM DUAL;

RTRIM e.g
-------------
BROWNINGyxX
```

SOUNDEX

Syntax

```
soundex::=SOUNDEX(attribute)
```

Purpose

SOUNDEX returns a character string containing the phonetic representation of attribute. This function enables you to compare words that are spelled differently, but sound similar in English.

The phonetic representation is defined in *The Art of Computer Programming, Volume 3: Sorting and Searching*, by Donald E. Knuth, as follows:

- Retain the first letter of the string and remove all other occurrences of the following letters: a, e, h, i, o, u, w, y.
- Assign numbers to the remaining letters (after the first) as follows:
  - b, f, p, v = 1
  - c, g, j, k, q, s, x, z = 2
  - d, t = 3
  - l = 4
  - m, n = 5
  - r = 6
- If two or more letters with the same number were adjacent in the original name (before step 1), or adjacent except for any intervening h and w, omit all but the first.
- Return the first four bytes padded with 0.

Data types for `attribute` can be CHAR and VARCHAR2. The return value is the same data type as `attribute`.

Example

The following example returns the employees whose last names are a phonetic representation of "Smyth":

```
SELECT last_name, first_name
   FROM hr.employees
   WHERE SOUNDEX(last_name) = SOUNDEX('SMYTHE');


LAST_NAME  FIRST_NAME
---------- ----------
Smith      Lindsey
```

SUBSTR, SUBSTR2, SUBSTR4, SUBSTRB, SUBSTRC

Syntax

```
substr::=SUBSTR(attribute, position, substring_length)
substr2::=SUBSTR2(attribute, position, substring_length)
substr4::=SUBSTR4(attribute, position, substring_length)
substrb::=SUBSTRB(attribute, position, substring_length)
substrc::=SUBSTRC(attribute, position, substring_length)
```

Purpose

The substring functions return a portion of `attribute`, beginning at character `position`, `substring_length` characters long. `SUBSTR` calculates lengths using characters as defined by the input character set. `SUBSTRB` uses bytes instead of characters. `SUBSTRC` uses Unicode complete characters. `SUBSTR2` uses UCS2 code points. `SUBSTR4` uses UCS4 code points.

- If `position` is 0, it is treated as 1.

- If position is positive, Warehouse Builder counts from the beginning of attribute to find the first character.
- If position is negative, Warehouse Builder counts backward from the end of attribute.
- If substring_length is omitted, Warehouse Builder returns all characters to the end of attribute. If substring_length is less than 1, a null is returned.

Data types for attribute can be CHAR and VARCHAR2. The return value is the same data type as attribute. Floating-point numbers passed as arguments to SUBSTR are automatically converted to integers.

Examples

The following example returns several specified substrings of "ABCDEFG":

```
SELECT SUBSTR('ABCDEFG',3,4) "Substring"
   FROM DUAL;

Substring
---------
CDEF

SELECT SUBSTR('ABCDEFG',-5,4) "Substring"
   FROM DUAL;

Substring
---------
CDEF
```

Assume a double-byte database character set:

```
SELECT SUBSTRB('ABCDEFG',5,4.2) "Substring with bytes"
   FROM DUAL;

Substring with bytes
--------------------
CD
```

TRANSLATE

Syntax

```
translate::=TRANSLATE(attribute, from_string, to_string)
```

Purpose

TRANSLATE returns attribute with all occurrences of each character in from_string replaced by its corresponding character in to_string. Characters in attribute that are not in from_string are not replaced. The argument from_string can contain more characters than to_string. In this case, the extra characters at the end of from_string have no corresponding characters in to_string. If these extra characters appear in attribute, they are removed from the return value.

You cannot use an empty string for to_string to remove all characters in from_string from the return value. Warehouse Builder interprets the empty string as null, and if this function has a null argument, it returns null.

Examples

The following statement translates a license number. All letters 'ABC...Z' are translated to 'X' and all digits '012 . . . 9' are translated to '9':

```
SELECT TRANSLATE('2KRW229','0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'9999999999XXXXXXXXXXXXXXXXXXXXXXXXXX') "License"
FROM DUAL;

License
--------
9XXX999
```

The following statement returns a license number with the characters removed and the digits remaining:

```
SELECT TRANSLATE('2KRW229','0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ', '0123456789')
"Translate example"
FROM DUAL;

Translate example
-----------------
2229
```

TRIM

Syntax

```
trim::=TRIM(attribute)
```

Purpose

`TRIM` enables you to trim leading or trailing spaces (or both) from a character string. The function returns a value with data type VARCHAR2. The maximum length of the value is the length of `attribute`.

Example

This example trims leading and trailing spaces from a string:

```
SELECT TRIM ('   Warehouse    ') "TRIM Example"
   FROM DUAL;

TRIM example
------------
Warehouse
```

UPPER

Syntax

```
upper::=UPPER(attribute)
```

Purpose

`UPPER` returns `attribute`, with all letters in uppercase; `attribute` can be of the data types CHAR and VARCHAR2. The return value is the same data type as `attribute`.

Example

The following example returns a string in uppercase:

```
SELECT UPPER('Large') "Uppercase"
   FROM DUAL;

Upper
-----
LARGE
```

WB_LOOKUP_CHAR (number)

Syntax

```
WB.LOOKUP_CHAR (table_name
, column_name
, key_column_name
, key_value
)
```

where `table_name` is the name of the table to perform the lookup on and `column_name` is the name of the VARCHAR2 column that will be returned. For example, the result of the lookup `key_column_name` is the name of the NUMBER column used as the key to match on in the lookup table, `key_value` is the value of the key column mapped into the `key_column_name` with which the match will be done.

Purpose

To perform a key lookup on a number that returns a VARCHAR2 value from a database table using a NUMBER column as the matching key.

Example

Consider the following table as a lookup table LKP1:

```
KEY_COLUMN    TYPE    COLOR
10            Car     Red
20            Bike    Green
```

Using this package with the following call:

```
WB.LOOKUP_CHAR ('LKP1'
, 'TYPE'
, 'KEYCOLUMN'
, 20
)
```

returns the value of 'Bike' as output of this transform. This output would then be processed in the mapping as the result of an inline function call.

> **Note:** This function is a row-based key lookup. Set-based lookups are supported when you use the lookup operator.

WB_LOOKUP_CHAR (varchar2)

Syntax

```
WB.LOOKUP_CHAR (table_name
```

```
, column_name
, key_column_name
, key_value
)
```

where `table_name` is the name of the table to perform the lookup on; `column_name` is the name of the VARCHAR2 column that will be returned, for instance, the result of the lookup; `key_column_name` is the name of the VARCHAR2 column used as the key to match on in the lookup table; `key_value` is the value of the key column, for instance, the value mapped into the `key_column_name` with which the match will be done.

Purpose

To perform a key lookup on a VARCHAR2 character that returns a VARCHAR2 value from a database table using a VARCHAR2 column as the matching key.

Example

Consider the following table as a lookup table LKP1:

```
KEYCOLUMN  TYPE  COLOR
ACV        Car   Red
ACP        Bike  Green
```

Using this package with the following call:

```
WB.LOOKUP_CHAR ('LKP1'
, 'TYPE'
, 'KEYCOLUMN'
, 'ACP'
)
```

returns the value of 'Bike' as output of this transformation. This output is then processed in the mapping as the result of an inline function call.

> **Note:** This function is a row-based key lookup. Set-based lookups are supported when you use the lookup operator.

WB_IS_SPACE

Syntax

```
WB_IS_SPACE(attibute)
```

Purpose

Checks whether a string value only contains spaces. This function returns a Boolean value. In mainframe sources, some fields contain many spaces to make a file adhere to the fixed length format. This function provides a way to check for these spaces.

Example

`WB_IS_SPACE` returns TRUE if attribute contains only spaces.

Control Center Transformations

Control Center transformations are used in a process flow or in custom transformations to enable you to access information about the Control Center at

execution time. For example, you can use a Control Center transformation in the expression on a transition to help control the flow through a process flow at execution time. You can also use Control Center transformations within custom functions. These custom functions can in turn be used in the design of your process flow.

All Control Center transformations require an audit ID that provides a handle to the audit data stored in the Control Center repository. The audit ID is a key into the public view `ALL_RT_AUDIT_EXECUTIONS`. The transformations can be used to obtain data specific to that audit ID at execution time. When run in the context of a process flow, you can obtain the audit ID at execution time using the pseudo variable `audit_id` in a process flow expression. This variable is evaluated as the audit ID of the currently executing job. For example, for a map input parameter, this represents the map execution and for a transition this represents the job at the source of the transition.

The Control Center transformations are:

WB_RT_GET_ELAPSED_TIME

Syntax

```
WB_RT_GET_ELAPSED_TIME(audit_id)
```

Purpose

This function returns the elapsed time, in seconds, for the job execution given by the specified `audit_id`. It returns null if the specified audit ID does not exist. For example, you can use this function on a transition if you want to make a choice dependent on the time taken by the previous activity.

Example

The following example returns the time elapsed since the activity represented by `audit_id` was started:

```
declare
   audit_id NUMBER := 1812;
   l_time NUMBER;
begin
   l_time:= WB_RT_GET_ELAPSED_TIME(audit_id);
end;
```

WB_RT_GET_JOB_METRICS

Syntax

```
WB_RT_GET_JOB_METRICS(audit_id, no_selected, no_deleted, no_updated, no_inserted,
no_discarded, no_merged, no_corrected)
```

where `no_selected` represents the number of rows selected, `no_deleted` represents the number of rows deleted, `no_updated` represents the number of rows updated, `no_inserted` represents the number of rows inserted, `no_discarded` represents the number of rows discarded, `no_merged` represents the number of rows merged, and `no_corrected` represents the number of rows corrected during the job execution.

Purpose

This procedure returns the metrics of the job execution represented by the specified `audit_id`. The metrics include the number of rows selected, deleted, updated, inserted, discarded, merged, and corrected.

Example

The following example retrieves the job metrics for the audit ID represented by `audit_id`.

```
declare
   audit_id NUMBER := 16547;
   l_nselected NUMBER;
   l_ndeleted NUMBER;
   l_nupdated NUMBER;
   l_ninserted NUMBER;
   l_ndiscarded NUMBER;
   l_nmerged NUMBER;
   l_ncorrected NUMBER;
begin
   WB_RT_GET_JOB_METRICS(audit_id, l_nselected, l_ndeleted, l_nupdated,
                         l_ninserted, l_ndiscarded, l_nmerged, l_ncorrected);
   dbms_output.put_line('sel=' || l_nselected || ', del=' l_ndeleted ||
                        ', upd=' || l_nupdated);
   dbms_output.put_line('ins='|| l_ninserted || ' , dis=' || l_ndiscarded );
   dbms_output.put_line('mer=' || l_nmerged || ', cor=' ||l_ncorrected);
 end;
```

WB_RT_GET_LAST_EXECUTION_TIME

Syntax

```
WB_RT_GET_LAST_EXECUTION_TIME(objectName, objectType, objectLocationName)
```

where `objectName` represents the name of the object, `objectType` represents the type of the object (for example MAPPING, DATA_AUDITOR, PROCESS_FLOW, SCHEDULABLE), and `objectLocationName` represents the location to which the object is deployed.

Purpose

This transformation gives you access to time-based data. Typically, you can use this in a Process Flow to model some design aspect that is relevant to "time". For example you

can design a path that may execute different maps if the time since the last execution is more than 1 day.

You can also use this transformation to determine time-synchronization across process flows that are running concurrently. For example, you can choose a path in a process flow according to whether another Process Flow has completed.

Example

The following example retrieves the time when the mapping TIMES_MAP was last executed and the if condition determines whether this time was within 1 day of the current time. Based on this time, it can perform different actions.

```
declare
    last_exec_time DATE;
begin
    last_exec_time:=WB_RT_GET_LAST_EXECUTION_TIME('TIMES_MAP','MAPPING','WH_
LOCATION');
    if last_exec_time < sysdate - 1 then
--       last-execution was more than one day ago
--       provide details of action here
          NULL;
    Else
--       provide details of action here
          NULL;
    end if;
end;
```

WB_RT_GET_MAP_RUN_AUDIT

Syntax

```
WB_RT_GET_MAP_RUN_AUDIT(audit_id)
```

Purpose

This function returns the map run ID for a job execution that represents a map activity. It returns null if `audit_id` does not represent the job execution for a map. For example, you can use the returned ID as a key to access the ALL_RT_MAP_RUN_<name> views for more information.

Example

The following example retrieves the map run ID for a job execution whose audit ID is 67265. It then uses this map run ID to obtain the name of the source from the ALL_RT_MAP_RUN_EXECUTIONS public view.

```
declare
  audit_id NUMBER := 67265;
  l_sources VARCHAR2(256);
  l_run_id NUMBER;
begin
  l_run_id := WB_RT_GET_MAP_RUN_AUDIT_ID(audit_id);
  SELECT source_name INTO l_sources FROM all_rt_map_run_sources
         WHERE map_run_id = l_run_id;
end;
```

WB_RT_GET_NUMBER_OF_ERRORS

Syntax

```
WB_RT_GET_NUMBER_OF_ERRORS(audit_id)
```

Purpose

This function returns the number of errors recorded for the job execution given by the specified `audit_id`. It returns null if the specific `audit_id` is not found.

Example

The following example retrieves the number of errors generated by the job execution whose audit ID is 8769. You can then perform different actions based on the number of errors.

```
declare
   audit_id NUMBER := 8769;
   l_errors NUMBER;
begin
   l_errors := WB_RT_GET_NUMBER_OF_ERRORS(audit_id);
   if l_errors < 5 then
      .....
   else
      .....
   end if;
end;
```

WB_RT_GET_NUMBER_OF_WARNINGS

Syntax

```
WB_RT_GET_NUMBER_OF_WARNINGS(audit_id)
```

Purpose

This function returns the number of warnings recorded for the job executions represented by `audit_id`. It returns null if `audit_id` does not exist.

Example

The following example returns the number of warnings generated by the job execution whose audit ID is 54632. You can then perform different actions based on the number of warnings.

```
declare
   audit_is NUMBER := 54632;
   l_warnings NUMBER;
begin
   l_ warnings:= WB_RT_GET_NUMBER_OF_WARNINGS (audit_id);
   if l_warnings < 5 then
      .....
   else
      .....
   end if;
end;
```

WB_RT_GET_PARENT_AUDIT_ID

Syntax

```
WB_RT_GET_PARENT_AUDIT_ID(audit_id)
```

Purpose

This function returns the audit id for the process that owns the job execution represented by audit_id. It returns null if audit_id does not exist. You can then use the returned audit id as a key into other public views such as ALL_RT_AUDIT_ EXECUTIONS, or other Control Center transformations if further information is required.

Example

The following example retrieves the parent audit ID for a job execution whose audit ID is 76859. It then uses this audit ID to determine the elapsed time for the parent activity. You can perform different actions based on the elapsed time of the parent activity.

```
declare
   audit_id NUMBER := 76859;
   l_elapsed_time NUMBER;
   l_parent_id NUMBER;
begin
   l_parent_id := WB_RT_GET_PARENT_AUDIT_ID(audit_id);
   l_elapsed_time := WB_RT_GET_ELAPSED_TIME(l_parent_id);
   if l_elpased_time < 100 then
      .....
   else
      .....
   end if;
end;
```

WB_RT_GET_RETURN_CODE

Syntax

```
WB_RT_GET_RETURN_CODE(audit_id)
```

Purpose

This function returns the return code recorded for the job execution represented by `audit_id`. It returns null if `audit_id` does not exist. For a successful job execution, the return code is greater than or equal to 0. A return code of less than 0 signifies that the job execution has failed.

Example

The following example retrieves the return code for the job execution whose audit ID is represented by `audit_id`.

```
declare
   audit_id NUMBER:=69;
   l_code NUMBER;
begin
   l_code:= WB_RT_GET_RETURN_CODE(audit_id);
end;
```

WB_RT_GET_START_TIME

Syntax

```
WB_RT_GET_START_TIME(audit_id)
```

Purpose

This function returns the start time for the job execution represented by `audit_id`. It returns null if `audit_id` does not exist. For example, you can use this in a transition if you wanted to make a choice dependent on when the previous activity started.

Example

The following example determines the start time of the job execution whose audit ID is 354.

```
declare
   audit_id NUMBER:=354;
   l_date TIMESTAMP WITH TIMEZONE;
begin
   l_date := WB_RT_GET_START_TIME(audit_id);
end;
```

Conversion Transformations

The conversion transformations enable Warehouse Builder users to perform functions that allow conditional conversion of values. These functions achieve "if -then" constructions within SQL.

The conversion transformations available in Warehouse Builder are:

- ASCIISTR on page 27-36
- COMPOSE on page 27-36
- CONVERT on page 27-37
- HEXTORAW on page 27-37
- NUMTODSINTERVAL on page 27-38
- NUMTOYMINTERVAL on page 27-39
- RAWTOHEX on page 27-39
- RAWTONHEX on page 27-40
- SCN_TO_TIMESTAMP on page 27-40
- TIMESTAMP_TO_SCN on page 27-41
- TO_BINARY_DOUBLE on page 27-42
- TO_BINARY_FLOAT on page 27-43
- TO_CHAR on page 27-43
- TO_CLOB on page 27-45
- TO_DATE on page 27-45
- TO_DSINTERVAL on page 27-45
- TO_MULTI_BYTE on page 27-46

ASCIISTR

Syntax

```
asciistr::=ASCII(attribute)
```

Purpose

`ASCIISTR` takes as its argument a string of data type VARCHAR2 and returns an ASCII version of the string. Non-ASCII characters are converted to the form \xxxx, where xxxx represents a UTF-16 code unit.

Example

The following example returns the ASCII string equivalent of the text string 'ABÄDE':

```
SELECT ASCIISTR('ABÄDE') FROM DUAL;

ASCIISTR('
----------
AB\00C4CDE
```

COMPOSE

Syntax

```
compose::=COMPOSE(attribute)
```

Purpose

`COMPOSE` returns a Unicode string in its fully normalized form in the same character set as the input. The parameter `attribute` can be any of the data types CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. For example, an `o` code point qualified by an umlaut code point will be returned as the o-umlaut code point.

Example

The following example returns the o-umlaut code point:

```
SELECT COMPOSE('o' || UNISTR('\038')) FROM DUAL;

CO
---
ö
```

CONVERT

Syntax

```
convert::=CONVERT(attribute, dest_char_set, source_char_set)
```

Purpose

CONVERT converts a character string specified in an operator attribute from one character set to another. The data type of the returned value is VARCHAR2.

- The attribute argument is the value to be converted. It can of the data types CHAR and VARCHAR2.
- The *dest_char_set* argument is the name of the character set to which *attribute* is converted.
- The source_char_set argument is the name of the character set in which attribute is stored in the database. The default value is the database character set.

Both the destination and source character set arguments can be either literals or columns containing the name of the character set. For complete correspondence in character conversion, the destination character set must contain a representation of all the characters defined in the source character set. When a character does not exist in the destination character set, it is substituted with a replacement character. Replacement characters can be defined as part of a character set definition.

Example

The following example illustrates character set conversion by converting a Latin-1 string to ASCII. The result is the same as importing the same string from a WE8ISO8859P1 database to a US7ASCII database.

```
SELECT CONVERT('Ä Ê Í Õ Ø A B C D E ', 'US7ASCII', 'WE8ISO8859P1')
   FROM DUAL;

CONVERT('ÄÊÍÕØABCDE'
--------------------
A E I ? ? A B C D E ?
```

Common character sets include:

- US7ASCII: US 7-bit ASCII character set
- WE8DEC: West European 8-bit character set
- WE8HP: HP West European Laserjet 8-bit character set
- F7DEC: DEC French 7-bit character set
- WE8EBCDIC500: IBM West European EBCDIC Code Page 500
- WE8PC850: IBM PC Code Page 850
- WE8ISO8859P1: ISO 8859-1 West European 8-bit character set

HEXTORAW

Syntax

```
hextoraw::=HEXTORAW(attribute)
```

Purpose

HEXTORAW converts attribute containing hexadecimal digits in the CHAR, VARCHAR2, NCHAR, or NVARCHAR2 character set to a raw value. This function does not support CLOB data directly. However, CLOBs can be passed in as arguments through implicit data conversion.

Example

The following example creates a simple table with a raw column, and inserts a hexadecimal value that has been converted to RAW:

```
CREATE TABLE test (raw_col RAW(10));

INSERT INTO test VALUES (HEXTORAW('7D'));
```

NUMTODSINTERVAL

Syntax

```
numtodsinterval::=NUMTODSINTERVAL(n,interval_unit)
```

Purpose

NUMTODSINTERVAL converts n to an INTERVAL DAY TO SECOND literal. The argument n can be any NUMBER value or an expression that can be implicitly converted to a NUMBER value. The argument interval_unit can be of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type. The value for interval_unit specifies the unit of n and must resolve to one of the following string values:

- 'DAY'
- 'HOUR'
- 'MINUTE'
- 'SECOND'

The parameter interval_unit is case insensitive. Leading and trailing values within the parentheses are ignored. By default, the precision of the return is 9.

Example

The following example calculates, for each employee, the number of employees hired by the same manager within the past 100 days from his or her hire date:

```
SELECT manager_id, last_name, hire_date,
       COUNT(*) OVER (PARTITION BY manager_id ORDER BY hire_date
       RANGE NUMTODSINTERVAL(100, 'day') PRECEDING) AS t_count
   FROM employees;

MANAGER_ID LAST_NAME                 HIRE_DATE    T_COUNT
---------- ---------                 ---------    -------
      100  Kochhar                   21-SEP-89          1
      100  De Haan                   13-JAN-93          1
      100  Raphaely                  07-DEC-94          1
      100  Kaufling                  01-MAY-95          1
      100  Hartstein                 17-FEB-96          1
. . .
      149  Grant                     24-MAY-99          1
      149  Johnson                   04-JUN-00          1
      210  Goyal                     17-AUG-97          1
```

```
      205  Gietz                     07-JUN-94          1
           King                      17-JUN-87          1
```

NUMTOYMINTERVAL

Syntax

```
numtoyminterval::=NUMTOYMINTERVAL(n,interval_unit)
```

Purpose

NUMTOYMINTERVAL converts n to an INTERVAL YEAR TO MONTH literal. The argument n can be any NUMBER value or an expression that can be implicitly converted to a NUMBER value. The argument interval_unit can be of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type. The value for interval_unit specifies the unit of n and must resolve to one of the following string values:

- 'DAY'
- 'HOUR'
- 'MINUTE'
- 'SECOND'

The parameter interval_unit is case insensitive. Leading and trailing values within the parentheses are ignored. By default, the precision of the return is 9.

Example

The following example calculates, for each employee, the total salary of employees hired in the past one year from his or her hire date.

```
SELECT last_name, hire_date, salary, SUM(salary)
       OVER (ORDER BY hire_date
       RANGE NUMTOYMINTERVAL(1,'year') PRECEDING) AS t_sal
   FROM employees;

LAST_NAME                 HIRE_DATE     SALARY      T_SAL
---------                 ---------     ------      -----
King                      17-JUN-87      24000      24000
Whalen                    17-SEP-87       4400      28400
Kochhar                   21-SEP-89      17000      17000
. . .
Markle                    08-MAR-00       2200     112400
Ande                      24-MAR-00       6400     106500
Banda                     21-APR-00       6200     109400
Kumar                     21-APR-00       6100     109400
```

RAWTOHEX

Syntax

```
rawtohex::=RAWTOHEX(raw)
```

Purpose

RAWTOHEX converts raw to a character value containing its hexadecimal equivalent. The argument must be RAW data type. You can specify a BLOB argument for this function if it is called from within a PL/SQL block.

Example

The following hypothetical example returns the hexadecimal equivalent of a RAW column value:

```
SELECT RAWTOHEX(raw_column) "Graphics"
   FROM grpahics;

Graphics
--------
7D
```

RAWTONHEX

Syntax

```
rawtonhex::=RAWTONHEX(raw)
```

Purpose

RAWTONHEX converts raw to an NVARCHAR2 character value containing its hexadecimal equivalent.

Example

The following hypothetical example returns the hexadecimal equivalent of a RAW column value:

```
SELECT RAWTONHEX(raw_column),
     DUMP ( RAWTONHEX (raw_column) ) "DUMP"
   FROM graphics;

RAWTONHEX(RA)           DUMP
----------------------- ------------------------------
7D                      Typ=1 Len=4: 0,55,0,68
```

SCN_TO_TIMESTAMP

Syntax

```
scn_to_timestamp::=SCN_TO_TIMESTAMP(number)
```

Purpose

SCN_TO_TIMESTAMP takes as an argument a number that evaluates to a system change number (SCN), and returns the approximate timestamp associated with that SCN. The returned value is of TIMESTAMP data type. This function is useful when you want to know the timestamp associated with an SCN. For example, it can be used in conjunction with the ORA_ROWSCN pseudocolumn to associate a timestamp with the most recent change to a row.

Example

The following example uses the ORA_ROWSCN pseudocolumn to determine the system change number of the last update to a row and uses SCN_TO_TIMESTAMP to convert that SCN to a timestamp:

```
SELECT SCN_TO_TIMESTAMP(ORA_ROWSCN) FROM employees
       WHERE employee_id=188;
```

You could use such a query to convert a system change number to a timestamp for use in an Oracle Flashback Query:

```
SELECT salary FROM employees WHERE employee_id = 188;

    SALARY
----------
      3800

UPDATE employees SET salary = salary*10 WHERE employee_id = 188;
COMMIT;

SELECT salary FROM employees WHERE employee_id = 188;

    SALARY
----------
      3800

SELECT SCN_TO_TIMESTAMP(ORA_ROWSCN) FROM employees
       WHERE employee_id=188;

SCN_TO_TIMESTAMP(ORA_ROWSCN)
---------------------------------------------------------
28-AUG-03 01.58.01.000000000 PM

FLASHBACK TABLE employees TO TIMESTAMP
   TO_TIMESTAMP('28-AUG-03 01.00.00.000000000 PM');

SELECT salary FROM employees WHERE employee_id = 188;
    SALARY
----------
      3800
```

TIMESTAMP_TO_SCN

Syntax

```
timestamp_to_scn::=TIMESTAMP_TO_SCN(timestamp)
```

Purpose

TIMESTAMP_TO_SCN takes as an argument a timestamp value and returns the approximate system change number (SCN) associated with that timestamp. The returned value is of data type NUMBER. This function is useful any time you want to know the SCN associated with a particular timestamp.

Example

The following example inserts a row into the oe.orders table and then uses TIMESTAMP_TO_SCN to determine the system change number of the insert operation. (The actual SCN returned will differ on each system.)

```
INSERT INTO orders (order_id, order_date, customer_id, order_total)
   VALUES (5000, SYSTIMESTAMP, 188, 2345);
COMMIT;

SELECT TIMESTAMP_TO_SCN(order_date) FROM orders
   WHERE order_id = 5000;

TIMESTAMP_TO_SCN(ORDER_DATE)
```

```
---------------------------
                     574100
```

TO_BINARY_DOUBLE

Syntax

```
to_binary_double::=TO_BINARY_DOUBLE(expr, fmt, nlsparam)
```

Purpose

TO_BINARY_DOUBLE returns a double-precision floating-point number. The parameter expr can be a character string or a numeric value of type NUMBER, BINARY_FLOAT, or BINARY_DOUBLE. If expr is BINARY_DOUBLE, then the function returns expr.

The arguments fmt and nlsparam are optional and are valid only if expr is a character string. They serve the same purpose as for the TO_CHAR (number) function. The case-insensitive string 'INF' is converted to positive infinity. The case-insensitive string '-INF' is converted to negative identity. The case-insensitive string 'NaN' is converted to NaN (not a number).

You cannot use a floating-point number format element (F, f, D, or d) in a character string expr. Also, conversions from character strings or NUMBER to BINARY_DOUBLE can be inexact, because the NUMBER and character types use decimal precision to represent the numeric value, and BINARY_DOUBLE uses binary precision. Conversions from BINARY_FLOAT to BINARY_DOUBLE are exact.

Example

The examples that follow are based on a table with three columns, each with a different numeric data type:

```
CREATE TABLE float_point_demo
  (dec_num NUMBER(10,2), bin_double BINARY_DOUBLE, bin_float BINARY_FLOAT);

INSERT INTO float_point_demo VALUES (1234.56,1234.56,1234.56);

SELECT * FROM float_point_demo;

   DEC_NUM BIN_DOUBLE  BIN_FLOAT
---------- ---------- ----------
   1234.56 1.235E+003 1.235E+003
```

The following example converts a value of data type NUMBER to a value of data type BINARY_DOUBLE:

```
SELECT dec_num, TO_BINARY_DOUBLE(dec_num)
  FROM float_point_demo;

   DEC_NUM TO_BINARY_DOUBLE(DEC_NUM)
---------- -------------------------
   1234.56                1.235E+003
```

The following example compares extracted dump information from the dec_num and bin_double columns:

```
SELECT DUMP(dec_num) "Decimal",
     DUMP(bin_double) "Double"
   FROM float_point_demo;
```

```
Decimal                     Double
--------------------------- ---------------------------------------------
Typ=2 Len=4: 194,13,35,57   Typ=101 Len=8: 192,147,74,61,112,163,215,10
```

TO_BINARY_FLOAT

Syntax

```
to_binary_float::=TO_BINARY_FLOAT(expr, fmt, nlsparam)
```

Purpose

TO_BINARY_FLOAT returns a single-precision floating-point number. The parameter expr can be a character string or a numeric value of type NUMBER, BINARY_FLOAT, or BINARY_DOUBLE. If expr is BINARY_FLOAT, then the function returns expr.

The arguments fmt and nlsparam are optional and are valid only if expr is a character string. They serve the same purpose as for the TO_CHAR (number) function. The case-insensitive string 'INF' is converted to positive infinity. The case-insensitive string '-INF' is converted to negative identity. The case-insensitive string 'NaN' is converted to NaN (not a number).

You cannot use a floating-point number format element (F, f, D, or d) in a character string expr. Also, conversions from character strings or NUMBER to BINARY_FLOAT can be inexact, because the NUMBER and character types use decimal precision to represent the numeric value, and BINARY_FLOAT uses binary precision. Conversions from BINARY_DOUBLE to BINARY_FLOAT are inexact if the BINARY_DOUBLE value uses more bits of precision than supported by the BINARY_FLOAT.

Example

Using table float_point_demo created for TO_BINARY_DOUBLE, the following example converts a value of data type NUMBER to a value of data type BINARY_FLOAT:

```
SELECT dec_num, TO_BINARY_FLOAT(dec_num)
  FROM float_point_demo;

   DEC_NUM TO_BINARY_FLOAT(DEC_NUM)
---------- ------------------------
   1234.56               1.235E+003
```

TO_CHAR

Syntax

```
to_char_date::=TO_CHAR(attribute, fmt, nlsparam)
```

Purpose

TO_CHAR converts attribute of DATE or NUMBER data type to a value of VARCHAR2 data type in the format specified by the format fmt. If you omit fmt, a date is converted to a VARCHAR2 value in the default date format and a number is converted to a VARCHAR2 value exactly long enough to hold its significant digits.

If attribute is a date, the nlsparam specifies the language in which month and day names and abbreviations are returned. This argument can have this form: 'NLS_

DATE_LANGUAGE = language' If you omit nlsparam, this function uses the default date language for your session.

If attribute is a number, the nlsparam specifies these characters that are returned by number format elements:

- Decimal character
- Group separator
- Local currency symbol
- International currency symbol

This argument can have the following form:

```
'NLS_NUMERIC_CHARACTERS = ''dg''
 NLS_CURRENCY = ''text''
 NLS_ISO_CURRENCY = territory '
```

The characters d and g represent the decimal character and group separator, respectively. They must be different single-byte characters. Within the quoted string, you must use two single quotation marks around the parameter values. Ten characters are available for the currency symbol.

If you omit nlsparam or any one of the parameters, this function uses the default parameter values for your session.

Example

The following example applies various conversions on the system date in the database:

```
SELECT TO_CHAR(sysdate) no_fmt FROM DUAL;

NO_FMT
---------
26-MAR-02

SELECT TO_CHAR(sysdate, 'dd-mm-yyyy') fmted FROM DUAL;

FMTED
----------
26-03-2002
```

In this example, the output is blank padded to the left of the currency symbol.

```
SELECT TO_CHAR(-10000,'L99G999D99MI') "Amount" FROM DUAL;

Amount
--------------
$10,000.00-
SELECT TO_CHAR(-10000,'L99G999D99MI'
     'NLS_NUMERIC_CHARACTERS = '',.''
     NLS_CURRENCY = ''AusDollars'' ') "Amount"
   FROM DUAL;

Amount
-------------------
AusDollars10.000,00-
```

TO_CLOB

Syntax

```
to_clob::=TO_CLOB(attribute)
```

Purpose

TO_CLOB converts NCLOB values in a LOB column or other character strings to CLOB values. char can be any of the data types CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. Oracle Database executes this function by converting the underlying LOB data from the national character set to the database character set.

Example

The following statement converts NCLOB data from the sample pm.print_media table to CLOB and inserts it into a CLOB column, replacing existing data in that column.

```
UPDATE PRINT_MEDIA SET AD_FINALTEXT = TO_CLOB (AD_FLTEXTN);
```

TO_DATE

Syntax

```
to_date::=TO_DATE(attribute, fmt, nlsparam)
```

Purpose

TO_DATE converts attribute of CHAR or VARCHAR2 data type to a value of data type DATE. The fmt is a date format specifying the format of attribute. If you omit fmt, attribute must be in the default date format. If fmt is 'J', for Julian, then attribute must be an integer. The nlsparam has the same purpose in this function as in the TO_CHAR function for date conversion.

Do not use the TO_DATE function with a DATE value for the attribute argument. The first two digits of the returned DATE value can differ from the original attribute, depending on fmt or the default date format.

Example

The following example converts character strings into dates:

```
SELECT TO_DATE('January 15, 1989, 11:00 A.M.','Month dd, YYYY, HH:MI A.M.',
     'NLS_DATE_LANGUAGE = American')
   FROM DUAL;

TO_DATE
---------
15-JAN-89
```

TO_DSINTERVAL

Syntax

```
to_dsinterval::=TO_DSINTERVAL(char, nlsparam)
```

Purpose

TO_DSINTERVAL converts a character string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to an INTERVAL DAY TO SECOND value. The argument char represents the character string to be converted. The only valid nlsparam you can specify in this function is NLS_NUMERIC_CHARACTERS. nlsparam can have the form: NLS_NUMERIC_CHARACTERS = "dg", where d represents the decimal character and g represents the group separator.

Example

The following example selects from the employees table the employees who had worked for the company for at least 100 days on January 1, 1990:

```
SELECT employee_id, last_name
   FROM employees
   WHERE hire_date + TO_DSINTERVAL('100 10:00:00') <= DATE '1990-01-01';

EMPLOYEE_ID LAST_NAME
----------- ---------------
        100 King
        101 Kochhar
        200 Whalen
```

TO_MULTI_BYTE

Syntax

```
to_multi_byte::=TO_MULTI_BYTE(attribute)
```

Purpose

TO_MULTI_BYTE returns attribute with all of its single-byte characters converted to their corresponding multibyte characters; attribute can be of data type CHAR or VARCHAR2. The value returned is in the same data type as attribute. Any single-byte characters in attribute that have no multibyte equivalents appear in the output string as single-byte characters.

This function is useful only if your database character set contains both single-byte and multibyte characters.

Example

The following example illustrates converting from a single byte 'A' to a multi byte:

```
'A' in UTF8:
SELECT dump(TO_MULTI_BYTE('A')) FROM DUAL;

DUMP(TO_MULTI_BYTE('A'))
------------------------
Typ=1 Len=3: 239,188,161
```

TO_NCHAR

Syntax

```
to_nchar::=TO_NCHAR(c, fmt, nlsparam)
```

Purpose

TO_NCHAR converts a character string, CLOB, or NCLOB value from the database character set to the national character set. This function is equivalent to the TRANSLATE ... USING function with a USING clause in the national character set.

Example

The following example converts NCLOB data from the pm.print_media table to the national character set:

```
SELECT TO_NCHAR(ad_fltextn) FROM print_media
   WHERE product_id = 3106;

TO_NCHAR(AD_FLTEXTN)
--------------------------------------------------------------------------------
TIGER2  Tastaturen...weltweit fuehrend in Computer-Ergonomie.
TIGER2 3106 Tastatur
Product Nummer: 3106
Nur 39 EURO!
Die Tastatur KB 101/CH-DE ist eine Standard PC/AT Tastatur mit 102 Tasten. Tasta
turbelegung: Schweizerdeutsch.
. NEU: Kommt mit ergonomischer Schaumstoffunterlage.
. Extraflache und ergonimisch-geknickte Versionen verfugbar auf Anfrage.
. Lieferbar in Elfenbein, Rot oder Schwarz.
```

TO_NCLOB

Syntax

```
to_nclob::=TO_NCLOB(char)
```

Purpose

TO_NCLOB converts CLOB values in a LOB column or other character strings to NCLOB values. char can be any of the data types CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. Oracle Database implements this function by converting the character set of char from the database character set to the national character set.

Example

The following example inserts some character data into an NCLOB column of the pm.print_media table by first converting the data with the TO_NCLOB function:

```
INSERT INTO print_media (product_id, ad_id, ad_fltextn)
   VALUES (3502, 31001, TO_NCLOB('Placeholder for new product description'));
```

TO_NUMBER

Syntax

```
to_number::=TO_NUMBER(attribute, fmt, nlsparam)
```

Purpose

TO_NUMBER converts attribute to a value of CHAR or VARCHAR2 data type containing a number in the format specified by the optional format fmt, to a value of NUMBER data type.

Examples

The following example converts character string data into a number:

```
UPDATE employees
   SET salary = salary + TO_NUMBER('100.00', '9G999D99')
   WHERE last_name = 'Perkins';
```

The `nlsparam` string in this function has the same purpose as it does in the `TO_CHAR` function for number conversions.

```
SELECT TO_NUMBER('-AusDollars100','L9G999D99',
' NLS_NUMERIC_CHARACTERS = '',.''
NLS_CURRENCY = ''AusDollars''
') "Amount"
FROM DUAL;

Amount
----------
-100
```

TO_SINGLE_BYTE

Syntax

```
to_single_byte::=TO_SINGLE_BYTE(attribute)
```

Purpose

`TO_SINGLE_BYTE` returns `attribute` with all of its multibyte characters converted to their corresponding single-byte characters; `attribute` can be of data type CHAR or VARCHAR2. The value returned is in the same data type as `attribute`. Any multibyte characters in `attribute` that have no single-byte equivalents appear in the output as multibyte characters.

This function is useful only if your database character set contains both single-byte and multibyte characters.

Example

The following example illustrates going from a multibyte 'A' in UTF8 to a single byte ASCII 'A':

```
SELECT TO_SINGLE_BYTE( CHR(15711393)) FROM DUAL;

T
-
A
```

TO_TIMESTAMP

Syntax

```
to_timestamp::=TO_TIMESTAMP(char, fnt, nlsparam)
```

Purpose

`TO_TIMESTAMP` converts `char` of data type CHAR, VARCHAR2, NCHAR, or NVARCHAR2 to a value of TIMESTAMP data type. The optional `fmt` specifies the format of `char`. If you omit `fmt`, then `char` must be in the default format of the TIMESTAMP data type, which is determined by the NLS_TIMESTAMP_FORMAT

initialization parameter. The optional nlsparam argument has the same purpose in this function as in the TO_CHAR function for date conversion.

This function does not support CLOB data directly. However, CLOBs can be passed in as arguments through implicit data conversion.

Example

The following example converts a character string to a timestamp. The character string is not in the default TIMESTAMP format, so the format mask must be specified:

```
SELECT TO_TIMESTAMP ('10-Sep-02 14:10:10.123000', 'DD-Mon-RR HH24:MI:SS.FF')
      FROM DUAL;

TO_TIMESTAMP('10-SEP-0214:10:10.123000','DD-MON-RRHH24:MI:SS.FF')
---------------------------------------------------------------------------
10-SEP-02 02.10.10.123000000 PM
```

TO_TIMESTAMP_TZ

Syntax

```
to_timestamp_tz::=TO_TIMESTAMP_TZ(char, fmt, nlsparam)
```

Purpose

TO_TIMESTAMP_TZ converts char of data type CHAR, VARCHAR2, NCHAR, or NVARCHAR2 to a value of TIMESTAMP WITH TIME ZONE data type. The optional fmt specifies the format of char. If you omit fmt, then char must be in the default format of the TIMESTAMP WITH TIME ZONE data type. The optional nlsparam has the same purpose in this function as in the TO_CHAR function for date conversion.

> **Note:** This function does not convert character strings to TIMESTAMP WITH LOCAL TIME ZONE.

Example

The following example converts a character string to a value of TIMESTAMP WITH TIME ZONE:

```
SELECT TO_TIMESTAMP_TZ('1999-12-01 11:00:00 -8:00','YYYY-MM-DD HH:MI:SS TZH:TZM')
     FROM DUAL;

TO_TIMESTAMP_TZ('1999-12-0111:00:00-08:00','YYYY-MM-DDHH:MI:SSTZH:TZM')
---------------------------------------------------------------------
01-DEC-99 11.00.00.000000000 AM -08:00
```

The following example casts a null column in a UNION operation as TIMESTAMP WITH LOCAL TIME ZONE using the sample tables oe.order_items and oe.orders:

```
SELECT order_id, line_item_id, CAST(NULL AS TIMESTAMP WITH LOCAL TIME ZONE)
       order_date
    FROM order_items
    UNION
SELECT order_id, to_number(null), order_date
   FROM orders;

  ORDER_ID LINE_ITEM_ID ORDER_DATE
---------- ------------ ---------------------------------
```

```
      2354           1
      2354           2
      2354           3
      2354           4
      2354           5
      2354           6
      2354           7
      2354           8
      2354           9
      2354          10
      2354          11
      2354          12
      2354          13
      2354          14-JUL-00 05.18.23.234567 PM
      2355           1
      2355           2
...
```

TO_YMINTERVAL

Syntax

```
to_yminterval::=TO_YMINTERVAL(char)
```

Purpose

`TO_YMINTERVAL` converts a character string, represented by `char`, of data type CHAR, VARCHAR2, NCHAR, or NVARCHAR2 to an INTERVAL YEAR TO MONTH type.

Example

The following example calculates for each employee in the sample hr.employees table a date one year two months after the hire date:

```
SELECT hire_date, hire_date + TO_YMINTERVAL('01-02') "14 months"
   FROM employees;

HIRE_DATE 14 months
--------- ---------
17-JUN-87 17-AUG-88
21-SEP-89 21-NOV-90
13-JAN-93 13-MAR-94
03-JAN-90 03-MAR-91
21-MAY-91 21-JUL-92
. . .
```

UNISTR

Syntax

```
unistr::=UNISTR(string)
```

Purpose

`UNISTR` takes as its argument a text string, represented by `string`, and returns it in the national character set. The national character set of the database can be either AL16UTF16 or UTF8. UNISTR provides support for Unicode string literals by letting

you specify the Unicode encoding value of characters in the string. This is useful, for example, for inserting data into NCHAR columns.

The Unicode encoding value has the form '\xxxx' where 'xxxx' is the hexadecimal value of a character in UCS-2 encoding format. To include the backslash in the string itself, precede it with another backslash (\\). For portability and data preservation, Oracle recommends that in the UNISTR string argument you specify only ASCII characters and the Unicode encoding values.

Example

The following example passes both ASCII characters and Unicode encoding values to the UNISTR function, which returns the string in the national character set:

```
SELECT UNISTR('abc\00e5\00f1\00f6') FROM DUAL;

UNISTR
------
abcåñö
```

Date Transformations

Date transformations provide Warehouse Builder users with functionality to perform transformations on date attributes. These transformations are ordered and the custom functions provided with Warehouse Builder are all in the format `WB_<function name>`.

The date transformations provided with Warehouse Builder are:

- ADD_MONTHS on page 27-52
- CURRENT_DATE on page 27-53
- DBTIMEZONE on page 27-53
- FROM_TZ on page 27-53
- LAST_DAY on page 27-54
- MONTHS_BETWEEN on page 27-54
- NEW_TIME on page 27-55
- NEXT_DAY on page 27-56
- ROUND (date) on page 27-56
- SESSIONTIMEZONE on page 27-56
- SYSDATE on page 27-57
- SYSTIMESTAMP on page 27-57
- SYS_EXTRACT_UTC on page 27-58
- TRUNC (date) on page 27-58
- WB_CAL_MONTH_NAME on page 27-58
- WB_CAL_MONTH_OF_YEAR on page 27-59
- WB_CAL_MONTH_SHORT_NAME on page 27-59
- WB_CAL_QTR on page 27-60
- WB_CAL_WEEK_OF_YEAR on page 27-60

ADD_MONTHS

Syntax

```
add_months::=ADD_MONTHS(attribute, n)
```

Purpose

`ADD_MONTHS` returns the date in the `attribute` plus `n` months. The argument `n` can be any integer. This will typically be added from an `attribute` or from a constant.

If the date in `attribute` is the last day of the month or if the resulting month has fewer days than the day component of `attribute`, then the result is the last day of the resulting month. Otherwise, the result has the same day component as `attribute`.

Example

The following example returns the month after the `hire_date` in the sample table employees:

```
SELECT TO_CHAR(ADD_MONTHS(hire_date,1), 'DD-MON-YYYY') "Next month"
   FROM employees
   WHERE last_name = 'Baer';

Next Month
-----------
07-JUL-1994
```

CURRENT_DATE

Syntax

```
current_date::=CURRENT_DATE()
```

Purpose

CURRENT_DATE returns the current date in the session time zone, in a value in the Gregorian calendar of data type DATE.

Example

The following example illustrates that CURRENT_DATE is sensitive to the session time zone:

```
ALTER SESSION SET TIME_ZONE = '-5:0';
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
SELECT SESSIONTIMEZONE, CURRENT_DATE FROM DUAL;

SESSIONTIMEZONE CURRENT_DATE
--------------- --------------------
-05:00          29-MAY-2000 13:14:03

ALTER SESSION SET TIME_ZONE = '-8:0';
SELECT SESSIONTIMEZONE, CURRENT_DATE FROM DUAL;

SESSIONTIMEZONE CURRENT_DATE
--------------- --------------------
-08:00          29-MAY-2000 10:14:33
```

DBTIMEZONE

Syntax

```
dbtimezone::+DBTIMEZONE()
```

Purpose

DBTIMEZONE returns the value of the database time zone. The return type is a time zone offset (a character type in the format '[+|-]TZH:TZM') or a time zone region name, depending on how the user specified the database time zone value in the most recent CREATE DATABASE or ALTER DATABASE statement.

Example

The following example assumes that the database time zone is set to UTC time zone:

```
SELECT DBTIMEZONE FROM DUAL;

DBTIME
------
+00:00
```

FROM_TZ

Syntax

```
from_tz::=FROM_TZ(timestamp_value, time_zone_value)
```

Purpose

FROM_TZ converts a timestamp value, represented by timestamp_value, and a time zone, represented by time_zone_value, to a TIMESTAMP WITH TIME ZONE value. time_zone_value is a character string in the format 'TZH:TZM' or a character expression that returns a string in TZR with optional TZD format.

Example

The following example returns a timestamp value to TIMESTAMP WITH TIME ZONE:

```
SELECT FROM_TZ(TIMESTAMP '2000-03-28 08:00:00', '3:00')
   FROM DUAL;

FROM_TZ(TIMESTAMP'2000-03-2808:00:00','3:00')
---------------------------------------------------------------
28-MAR-00 08.00.00 AM +03:00
```

LAST_DAY

Syntax

```
last_day::=LAST_DAY(attribute)
```

Purpose

LAST_DAY returns the date of the last day of the month that contains the date in attribute.

Examples

The following statement determines how many days are left in the current month.

```
SELECT SYSDATE, LAST_DAY(SYSDATE) "Last", LAST_DAY(SYSDATE) - SYSDATE "Days Left"
   FROM DUAL;

SYSDATE   Last Days Left
--------- --------- ----------
23-OCT-97 31-OCT-97 8
```

MONTHS_BETWEEN

Syntax

```
months_between::=MONTHS_BETWEEN(attribute1, attribute2)
```

Purpose

MONTHS_BETWEEN returns the number of months between dates in attribute1 and attribute2. If attribute1 is later than attribute2, the result is positive; if earlier, then the result is negative.

If attribute1 and attribute2 are either the same day of the month or both last days of months, the result is always an integer. Otherwise, Oracle calculates the fractional portion of the result-based on a 31-day month and considers the difference in time components attribute1 and attribute2.

Example

The following example calculates the months between two dates:

```
SELECT MONTHS_BETWEEN(TO_DATE('02-02-1995','MM-DD-YYYY'),
     TO_DATE('01-01-1995','MM-DD-YYYY') ) "Months"
   FROM DUAL;

Months
----------
1.03225806
```

NEW_TIME

Syntax

```
new_time::=NEW_TIME(attribute, zone1, zone2)
```

Purpose

`NEW_TIME` returns the date and time in time zone `zone2` when date and time in time zone `zone1` are the value in `attribute`. Before using this function, you must set the `NLS_DATE_FORMAT` parameter to display 24-hour time.

The arguments `zone1` and `zone2` can be any of these text strings:

- AST, ADT: Atlantic Standard or Daylight Time
- BST, BDT: Bering Standard or Daylight Time
- CST, CDT: Central Standard or Daylight Time
- CST, EDT: Eastern Standard or Daylight Time
- GMT: Greenwich Mean Time
- HST, HDT: Alaska-Hawaii Standard Time or Daylight Time.
- MST, MDT: Mountain Standard or Daylight Time
- NST: Newfoundland Standard Time
- PST, PDT: Pacific Standard or Daylight Time
- YST, YDT: Yukon Standard or Daylight Time

Example

The following example returns an Atlantic Standard time, given the Pacific Standard time equivalent:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
SELECT NEW_TIME (TO_DATE ('11-10-99 01:23:45', 'MM-DD-YY HH24:MI:SS'),
      'AST', 'PST') "New Date and Time"
   FROM DUAL;

New Date and Time
--------------------
09-NOV-1999 21:23:45
```

NEXT_DAY

Syntax

```
next_day::=NEXT_DAY(attribute1, attribute2)
```

Purpose

NEXT_DAY returns the date of the first weekday named by the string in attribute2 that is later than the date in attribute1. The argument attribute2 must be a day of the week in the date language of your session, either the full name or the abbreviation. The minimum number of letters required is the number of letters in the abbreviated version. Any characters immediately following the valid abbreviation are ignored. The return value has the same hours, minutes, and seconds component as the argument attribute1.

Example

This example returns the date of the next Tuesday after February 2, 2001:

```
SELECT NEXT_DAY('02-FEB-2001','TUESDAY') "NEXT DAY"
   FROM DUAL;

NEXT DAY
-----------
06-FEB-2001
```

ROUND (date)

Syntax

```
round_date::=ROUND(attribute, fmt)
```

Purpose

ROUND returns the date in attribute rounded to the unit specified by the format model fmt. If you omit fmt, date is rounded to the nearest day.

Example

The following example rounds a date to the first day of the following year:

```
SELECT ROUND (TO_DATE ('27-OCT-00'),'YEAR') "New Year"
   FROM DUAL;

New Year
---------
01-JAN-01
```

SESSIONTIMEZONE

Syntax

```
sessiontimezone::=SESSIONTIMEZONE()
```

Purpose

SESSIONTIMEZONE returns the time zone of the current session. The return type is a time zone offset (a character type in the format '[+|]TZH:TZM') or a time zone region

name, depending on how the user specified the session time zone value in the most recent ALTER SESSION statement. You can set the default client session time zone using the ORA_SDTZ environment variable.

Example

The following example returns the time zone of the current session:

```
SELECT SESSIONTIMEZONE FROM DUAL;

SESSION
-------
-08:00
```

SYSDATE

Syntax

```
sysdate::=SYSDATE
```

Purpose

SYSDATE returns the current date and time. The data type of the returned value is DATE. The function requires no arguments. In distributed SQL statements, this function returns the date and time on your local database. You cannot use this function in the condition of a CHECK constraint.

Example

The following example returns the current date and time:

```
SELECT TO_CHAR(SYSDATE, 'MM-DD-YYYY HH24:MI:SS')"NOW" FROM DUAL;

NOW
-------------------
04-13-2001 09:45:51
```

SYSTIMESTAMP

Syntax

```
systimestamp::=SYSTIMESTAMP()
```

Purpose

SYSTIMESTAMP returns the system date, including fractional seconds and time zone, of the system on which the database resides. The return type is TIMESTAMP WITH TIME ZONE.

Example

The following example returns the system timestamp:

```
SELECT SYSTIMESTAMP FROM DUAL;

SYSTIMESTAMP
---------------------------------------------------------------------
28-MAR-00 12.38.55.538741 PM -08:00
```

The following example shows how to explicitly specify fractional seconds:

```
SELECT TO_CHAR(SYSTIMESTAMP, 'SSSSS.FF') FROM DUAL;

TO_CHAR(SYSTIME
---------------
55615.449255
```

SYS_EXTRACT_UTC

Syntax

```
sys_extract_utc::=SYS_EXTRACT_UTC(datetime_with_timezone)
```

Purpose

SYS_EXTRACT_UTC extracts the UTC (Coordinated Universal Time—formerly Greenwich Mean Time) from a datetime value with time zone offset or time zone region name.

Example

The following example extracts the UTC from a specified datetime:

```
SELECT SYS_EXTRACT_UTC(TIMESTAMP '2000-03-28 11:30:00.00 -08:00')
   FROM DUAL;

SYS_EXTRACT_UTC(TIMESTAMP'2000-03-2811:30:00.00-08:00')
------------------------------------------------------------------
28-MAR-00 07.30.00 PM
```

TRUNC (date)

Syntax

```
trunc_date::=TRUNC(attribute, fmt)
```

Purpose

`TRUNC` returns `attribute` with the time portion of the day truncated to the unit specified by the format model `fmt`. If you omit `fmt`, date is truncated to the nearest day.

Example

The following example truncates a date:

```
SELECT TRUNC(TO_DATE('27-OCT-92','DD-MON-YY'), 'YEAR') "New Year"
   FROM DUAL;

New Year
---------
01-JAN-92
```

WB_CAL_MONTH_NAME

Syntax

```
WB_CAL_MONTH_NAME(attribute)
```

Purpose

The function call returns the full-length name of the month for the date specified in `attribute`.

Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
SELECT WB_CAL_MONTH_NAME(sysdate)
   FROM DUAL;

WB_CAL_MONTH_NAME(SYSDATE)
---------------------------
March

SELECT WB_CAL_MONTH_NAME('26-MAR-2002')
   FROM DUAL;

WB_CAL_MONTH_NAME('26-MAR-2002')
---------------------------------
March
```

WB_CAL_MONTH_OF_YEAR

Syntax

```
WB_CAL_MONTH_OF_YEAR(attribute)
```

Purpose

`WB_CAL_MONTH_OF_YEAR` returns the month (1-12) of the year for date in `attribute`.

Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
SELECT WB_CAL_MONTH_OF_YEAR(sysdate) month
   FROM DUAL;

     MONTH
----------
         3

SELECT WB_CAL_MONTH_OF_YEAR('26-MAR-2002') month
FROM DUAL;

     MONTH
----------
         3
```

WB_CAL_MONTH_SHORT_NAME

Syntax

```
WB_CAL_MONTH_SHORT_NAME(attribute)
```

Purpose

WB_CAL_MONTH_SHORT_NAME returns the short name of the month (for example 'Jan') for date in attribute.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
SELECT WB_CAL_MONTH_SHORT_NAME (sysdate) month
FROM DUAL;

MONTH
---------
Mar

SELECT WB_CAL_MONTH_SHORT_NAME ('26-MAR-2002') month
FROM DUAL;

MONTH
---------
Mar
```

WB_CAL_QTR

Syntax

```
WB_CAL_QTR(attribute)
```

Purpose

WB_CAL_QTR returns the quarter of the Gregorian calendar year (for example Jan - March = 1) for the date in attribute.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
SELECT WB_CAL_QTR (sysdate) quarter
FROM DUAL;

   QUARTER
----------
         1

SELECT WB_CAL_QTR ('26-MAR-2002') quarter
FROM DUAL;

   QUARTER
----------
         1
```

WB_CAL_WEEK_OF_YEAR

Syntax

```
WB_CAL_WEEK_OF_YEAR(attribute)
```

Purpose

WB_CAL_WEEK_OF_YEAR returns the week of the year (1-53) for the date in attribute.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
SELECT WB_CAL_WEEK_OF_YEAR (sysdate) w_of_y
FROM DUAL;

    W_OF_Y
----------
        13

SELECT WB_CAL_WEEK_OF_YEAR ('26-MAR-2002') w_of_y
FROM DUAL;

    W_OF_Y
----------
        13
```

WB_CAL_YEAR

Syntax

```
WB_CAL_YEAR(attribute)
```

Purpose

WB_CAL_YEAR returns the numerical year component for the date in attribute.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
SELECT WB_CAL_YEAR (sysdate) year
FROM DUAL;

      YEAR
----------
      2002

SELECT WB_CAL_YEAR ('26-MAR-2002') w_of_y
FROM DUAL;

      YEAR
----------
      2002
```

WB_CAL_YEAR_NAME

Syntax

```
WH_CAL_YEAR_NAME(attribute)
```

Purpose

WB_CAL_YEAR_NAME returns the spelled out name of the year for the date in attribute.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
select WB_CAL_YEAR_NAME (sysdate) name
from dual;

NAME
---------------------------------------------
Two Thousand Two

select WB_CAL_YEAR_NAME ('26-MAR-2001') name
from dual;

NAME
---------------------------------------------
Two Thousand One
```

WB_DATE_FROM_JULIAN

Syntax

```
WB_DATE_FROM_JULIAN(attribute)
```

Purpose

WB_DATE_FROM_JULIAN converts Julian date attribute to a regular date.

Example

The following example shows the return value on a specified Julian date:

```
select to_char(WB_DATE_FROM_JULIAN(3217345),'dd-mon-yyyy') JDate
from dual;

JDATE
-----------
08-sep-4096
```

WB_DAY_NAME

Syntax

```
WB_DAY_NAME(attribute)
```

Purpose

WB_DAY_NAME returns the full name of the day for the date in attribute.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
select WB_DAY_NAME (sysdate) name
from dual;
```

```
NAME
-----------------------------------------
Thursday

select WB_DAY_NAME ('26-MAR-2002') name
from dual;

NAME
-----------------------------------------
Tuesday
```

WB_DAY_OF_MONTH

Syntax

```
WB_DAY_OF_MONTH(attribute)
```

Purpose

WB_DAY_OF_MONTH returns the day number within the month for the date in attribute.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
select WB_DAY_OF_MONTH (sysdate) num
from dual;

       NUM
----------
        28

select WB_DAY_OF_MONTH ('26-MAR-2002') num
from dual

       NUM
----------
        26
```

WB_DAY_OF_WEEK

Syntax

```
WB_DAY_OF_WEEK(attribute)
```

Purpose

WB_DAY_OF_WEEK returns the day number within the week for date attribute based on the database calendar.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
select WB_DAY_OF_WEEK (sysdate) num
```

```
from dual;

       NUM
----------
         5

select WB_DAY_OF_WEEK ('26-MAR-2002') num
from dual;


       NUM
----------
         3
```

WB_DAY_OF_YEAR

Syntax

```
WB_DAY_OF_YEAR(attribute)
```

Purpose

`WB_DAY_OF_YEAR` returns the day number within the year for the date `attribute`.

Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_DAY_OF_YEAR (sysdate) num
from dual;

       NUM
----------
        87

select WB_DAY_OF_YEAR ('26-MAR-2002') num
from dual;

       NUM
----------
        85
```

WB_DAY_SHORT_NAME

Syntax

```
WB_DAY_SHORT_NAME(attribute)
```

Purpose

`WB_DAY_SHORT_NAME` returns the three letter abbreviation or name for the date `attribute`.

Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_DAY_SHORT_NAME  (sysdate) abbr
from dual;

ABBR
-----------------------------------
Thu

select WB_DAY_SHORT_NAME  ('26-MAR-2002') abbr
from dual;

NUM
-----------------------------------
Tue
```

WB_DECADE

Syntax

```
WB_DECADE(attribute)
```

Purpose

WB_DECADE returns the decade number within the century for the date attribute.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
select WB_DECADE  (sysdate) dcd
from dual;

       DCD
----------
         2

select WB_DECADE  ('26-MAR-2002') DCD
from dual;

       DCD
----------
         2
```

WB_HOUR12

Syntax

```
WB_HOUR12(attribute)
```

Purpose

WB_HOUR12 returns the hour (in a 12-hour setting) component of the date corresponding to attribute.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
select WB_HOUR12 (sysdate) h12
from dual;

       H12
----------
         9

select WB_HOUR12 ('26-MAR-2002') h12
from dual;

       H12
----------
        12
```

Note: For a date not including the timestamp (in the second example), Oracle uses the 12:00 (midnight) timestamp and therefore returns 12 in this case.

WB_HOUR12MI_SS

Syntax

```
WB_HOUR12MI_SS(attribute)
```

Purpose

`WB_HOUR12MI_SS` returns the timestamp in `attribute` formatted to HH12:MI:SS.

Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_HOUR12MI_SS (sysdate) h12miss
from dual;

H12MISS
-------------------------------------
09:08:52

select WB_HOUR12MI_SS ('26-MAR-2002') h12miss
from dual;

H12MISS
-------------------------------------
12:00:00
```

Note: For a date not including the timestamp (in the second example), Oracle uses the 12:00 (midnight) timestamp and therefore returns 12 in this case.

WB_HOUR24

Syntax

```
WB_HOUR24(attribute)
```

Purpose

WB_HOUR24 returns the hour (in a 24-hour setting) component of date corresponding to attribute.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
select WB_HOUR24 (sysdate) h24
from dual;

       H24
----------
         9

select WB_HOUR24 ('26-MAR-2002') h24
from dual;

       H24
----------
         0
```

Note: For a date not including the timestamp (in the second example), Oracle uses the 00:00:00 timestamp and therefore returns the timestamp in this case.

WB_HOUR24MI_SS

Syntax

```
WB_HOUR24MI_SS(attribute)
```

Purpose

WB_HOUR24MI_SS returns the timestamp in attribute formatted to HH24:MI:SS.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
select WB_HOUR24MI_SS (sysdate) h24miss
from dual;

H24MISS
-----------------------------------
09:11:42

select WB_HOUR24MI_SS ('26-MAR-2002') h24miss
from dual;
```

```
H24MISS
-----------------------------------
00:00:00
```

> **Note:** For a date not including the timestamp (in the second example), Oracle uses the 00:00:00 timestamp and therefore returns the timestamp in this case.

WB_IS_DATE

Syntax

```
WB_IS_DATE(attribute, fmt)
```

Purpose

To check whether `attribute` contains a valid date. The function returns a Boolean value which is set to true if `attribute` contains a valid date. `Fmt` is an optional date format. If `fmt` is omitted, the date format of your database session is used.

You can use this function when you validate your data before loading it into a table. This way the value can be transformed before it reaches the table and causes an error.

Example

`WB_IS_DATE` returns true in PL/SQL if `attribute` contains a valid date.

WB_JULIAN_FROM_DATE

Syntax

```
WB_JULIAN_FROM_DATE(attribute)
```

Purpose

`WB_JULIAN_FROM_DATE` returns the Julian date of date corresponding to `attribute`.

Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_JULIAN_FROM_DATE (sysdate) jdate
from dual;

     JDATE
----------
   2452362

select WB_JULIAN_FROM_DATE ('26-MAR-2002') jdate
from dual;

     JDATE
----------
   2452360
```

WB_MI_SS

Syntax

```
WB_MI_SS(attribute)
```

Purpose

WB_MI_SS returns the minutes and seconds of the time component in the date corresponding to attribute.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
select WB_MI_SS (sysdate) mi_ss
from dual;

MI_SS
-----------------------------------------
33:23

select WB_MI_SS ('26-MAR-2002') mi_ss
from dual;

MI_SS
-----------------------------------------
00:00
```

Note: For a date not including the timestamp (in the second example), Oracle uses the 00:00:00 timestamp and therefore returns the timestamp in this case.

WB_WEEK_OF_MONTH

Syntax

```
WB_WEEK_OF_MONTH(attribute)
```

Purpose

WB_WEEK_OF_MONTH returns the week number within the calendar month for the date corresponding to attribute.

Example

The following example shows the return value on the sysdate and on a specified date string:

```
select WB_WEEK_OF_MONTH (sysdate) w_of_m
from dual;

    W_OF_M
----------
         4

select WB_WEEK_OF_MONTH ('26-MAR-2002') w_of_m
from dual;
```

```
    W_OF_M
----------
         4
```

Number Transformations

Number transformations provide Warehouse Builder users with functionality to perform transformations on numeric values. The custom functions provided with Warehouse Builder are prefixed with `WB_`.

All numerical transformations provided with Warehouse Builder are:

- ABS on page 27-71
- ACOS on page 27-71
- ASIN on page 27-71
- ATAN on page 27-72
- ATAN2 on page 27-72
- BITAND on page 27-72
- CEIL on page 27-73
- COS on page 27-74
- COSH on page 27-74
- EXP on page 27-74
- FLOOR on page 27-75
- LN on page 27-75
- LOG on page 27-75
- MOD on page 27-76
- NANVL on page 27-76
- POWER on page 27-77
- REMAINDER on page 27-77
- ROUND (number) on page 27-78
- SIGN on page 27-78
- SIN on page 27-79
- SINH on page 27-79
- SQRT on page 27-79
- TAN on page 27-80
- TANH on page 27-80
- TRUNC (number) on page 27-80
- WB_LOOKUP_NUM (on a number) on page 27-81
- WB_LOOKUP_NUM (on a varchar2) on page 27-82
- WB_IS_NUMBER on page 27-82

- WIDTH_BUCKET on page 27-83

ABS

Syntax

```
abs::=ABS(attribute)
```

Purpose

ABS returns the absolute value of `attribute`.

Example

The following example returns the absolute value of -15:

```
SELECT ABS(-15) "Absolute" FROM DUAL;
Absolute
----------
15
```

ACOS

Syntax

```
acos::= ACOS(attribute)
```

Purpose

`ACOS` returns the arc cosine of `attribute`. The argument `attribute` must be in the range of -1 to 1, and the function returns values in the range of 0 to pi, expressed in radians.

Example

The following example returns the arc cosine of .3:

```
SELECT ACOS(.3) "Arc_Cosine" FROM DUAL;

Arc_Cosine
----------
1.26610367
```

ASIN

Syntax

```
asin::=ASIN(attribute)
```

Purpose

`ASIN` returns the arc sine of `attribute`. The argument `attribute` must be in the range of -1 to 1, and the function returns values in the range of -pi/2 to pi/2, expressed in radians.

Example

The following example returns the arc cosine of .3:

```
SELECT ACOS(.3) "Arc_Sine" FROM DUAL;
```

```
Arc_Sine
----------
.304692654
```

ATAN

Syntax

```
atan::=ATAN(attribute)
```

Purpose

`ATAN` returns the arc tangent of `attribute`. The argument `attribute` can be in an unbounded range, and the function returns values in the range of -pi/2 to pi/2, expressed in radians.

Example

The following example returns the arc tangent of .3:

```
SELECT ATAN(.3) "Arc_Tangent" FROM DUAL;

Arc_Tangent
----------
.291456794
```

ATAN2

Syntax

```
atan2::=ATAN2(attribute1, attribute2)
```

Purpose

`ATAN2` returns the arc tangent of `attribute1` and `attribute2`. The argument attribute1 can be in an unbounded range, and the function returns values in the range of -pi to pi, depending on the signs of attribute1 and attribute2, and are expressed in radians. `ATAN2(attribute1,attribute2)` is the same as `ATAN2(attribute1/attribute2)`.

Example

The following example returns the arc tangent of .3 and .2:

```
SELECT ATAN2(.3,.2) "Arc_Tangent2" FROM DUAL;

Arc_Tangent2
------------
.982793723
```

BITAND

Syntax

```
bitand::=BITAND(expr1,expr2)
```

Purpose

BITAND computes an AND operation on the bits of expr1 and expr2, both of which must resolve to nonnegative integers, and returns an integer. This function is commonly used with the DECODE function, as illustrated in the example that follows.

Both arguments can be any numeric data type, or any nonnumeric data type that can be implicitly converted to NUMBER. The function returns NUMBER.

Example

The following represents each order_status in the sample table oe.orders by individual bits. (The example specifies options that can total only 7, so rows with order_status greater than 7 are eliminated.)

```
SELECT order_id, customer_id,
  DECODE(BITAND(order_status, 1), 1, 'Warehouse', 'PostOffice')
      Location,
  DECODE(BITAND(order_status, 2), 2, 'Ground', 'Air') Method,
  DECODE(BITAND(order_status, 4), 4, 'Insured', 'Certified') Receipt
  FROM orders
  WHERE order_status < 8;

  ORDER_ID CUSTOMER_ID LOCATION   MET RECEIPT
---------- ----------- ---------- --- ---------
      2458         101 Postoffice Air Certified
      2397         102 Warehouse  Air Certified
      2454         103 Warehouse  Air Certified
      2354         104 Postoffice Air Certified
      2358         105 Postoffice G   Certified
      2381         106 Warehouse  G   Certified
      2440         107 Warehouse  G   Certified
      2357         108 Warehouse  Air Insured
      2394         109 Warehouse  Air Insured
      2435         144 Postoffice G   Insured
      2455         145 Warehouse  G   Insured
. . .
```

CEIL

Syntax

```
ceil::=CEIL(attribute)
```

Purpose

CEIL returns smallest integer greater than or equal to attribute.

Example

The following example returns the smallest integer greater than or equal to 15.7:

```
 SELECT CEIL(15.7) "Ceiling" FROM DUAL;
 Ceiling
----------
16
```

COS

Syntax

```
cos::=COS(attribute)
```

Purpose

COS returns the cosine of attribute (an angle expressed in degrees).

Example

The following example returns the cosine of 180 degrees:

```
SELECT COS(180 * 3.14159265359/180) "Cosine" FROM DUAL;

Cosine
------
     -1
```

COSH

Syntax

```
cosh::=COSH(attribute)
```

Purpose

COSH returns the hyperbolic cosine of attribute.

Example

The following example returns the hyperbolic cosine of 0:

```
SELECT COSH(0) "Hyperbolic Cosine" FROM DUAL;

Hyperbolic Cosine
-----------------
                1
```

EXP

Syntax

```
exp::=EXP(attribute)
```

Purpose

EXP returns e raised to the nth power represented in attribute, where e = 2.71828183...

Example

The following example returns e to the 4th power:

```
SELECT EXP(4) "e to the 4th power" FROM DUAL;

e to the 4th power
------------------
54.59815
```

FLOOR

Syntax

```
floor::=FLOOR(attribute)
```

Purpose

FLOOR returns the largest integer equal to or less than the numerical value in attribute.

Example

The following example returns the largest integer equal to or less than 15.7:

```
SELECT FLOOR(15.7) "Floor" FROM DUAL;

Floor
----------
15
```

LN

Syntax

```
ln::=LN(attribute)
```

Purpose

LN returns the natural logarithm of attribute, where attribute is greater than 0.

Example

The following example returns the natural logarithm of 95:

```
SELECT LN(95) "Natural Logarithm" FROM DUAL;

Natural Logarithm
-----------------
4.55387689
```

LOG

Syntax

```
log::=LOG(attribute1, attribute2)
```

Purpose

LOG returns the logarithm, base attribute1 of attribute2. The base attribute1 can be any positive number other than 0 or 1 and attribute2 can be any positive number.

Example

The following example returns the logarithm of 100:

```
SELECT LOG(10,100) "Log base 10 of 100" FROM DUAL;
```

```
Log base 10 of 100
------------------
                2
```

MOD

Syntax

```
mod::=MOD(attribute1, attribute2)
```

Purpose

`MOD` returns the remainder of `attribute1` divided by `attribute2`. It returns `attribute1` if `attribute2` is 0.

Example

The following example returns the remainder of 11 divided by 4:

```
 SELECT MOD(11,4) "Modulus" FROM DUAL;
 Modulus
----------
3
```

NANVL

Syntax

```
nanvl::=NANVL(m,n)
```

Purpose

The NANVL function is useful only for floating-point numbers of type BINARY_FLOAT or BINARY_DOUBLE. It instructs Oracle Database to return an alternative value `n` if the input value `m` is NaN (not a number). If `m` is not NaN, then Oracle returns `m`. This function is useful for mapping NaN values to NULL.

This function takes as arguments any numeric data type or any nonnumeric data type that can be implicitly converted to a numeric data type. Oracle determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that data type, and returns that data type.

Example

Using table `float_point_demo` created for TO_BINARY_DOUBLE, insert a second entry into the table:

```
INSERT INTO float_point_demo
  VALUES (0,'NaN','NaN');

SELECT * FROM float_point_demo;

   DEC_NUM BIN_DOUBLE  BIN_FLOAT
---------- ---------- ----------
   1234.56 1.235E+003 1.235E+003
         0        Nan        Nan
```

The following example returns bin_float if it is not a number. Otherwise, 0 is returned.

```
SELECT bin_float, NANVL(bin_float,0)
  FROM float_point_demo;

 BIN_FLOAT NANVL(BIN_FLOAT,0)
---------- ------------------
1.235E+003         1.235E+003
       Nan                  0
```

POWER

Syntax

```
power::=POWER(attribute1, attribute2)
```

Purpose

POWER returns attribute1 raised to the nth power represented in attribute2. The base attribute1 and the exponent in attribute2 can be any numbers, but if attribute1 is negative, then attribute2 must be an integer.

Example

The following example returns three squared:

```
SELECT POWER(3,2) "Raised" FROM DUAL;
Raised
----------
9
```

REMAINDER

Syntax

```
remainder::=REMAINDER(m,n)
```

Purpose

REMAINDER returns the remainder of m divided by n. This function takes as arguments any numeric data type or any nonnumeric data type that can be implicitly converted to a numeric data type. Oracle determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that data type, and returns that data type.

If n = 0 or m = infinity, then Oracle returns an error if the arguments are of type NUMBER and NaN if the arguments are BINARY_FLOAT or BINARY_DOUBLE. If n != 0, then the remainder is m - (n*N) where N is the integer nearest m/n. If m is a floating-point number, and if the remainder is 0, then the sign of the remainder is the sign of m. Remainders of 0 are unsigned for NUMBER values.

The MOD function is similar to REMAINDER except that it uses FLOOR in its formula, whereas REMAINDER uses ROUND.

Example

Using table `float_point_demo` created for TO_BINARY_DOUBLE, the following example divides two floating-point numbers and returns the remainder of that operation:

```
SELECT bin_float, bin_double, REMAINDER(bin_float, bin_double)
  FROM float_point_demo;

 BIN_FLOAT BIN_DOUBLE REMAINDER(BIN_FLOAT,BIN_DOUBLE)
---------- ---------- --------------------------------
1.235E+003 1.235E+003                     5.859E-005
```

ROUND (number)

Syntax

```
round_number::=ROUND(attribute1, attribute2)
```

Purpose

`ROUND` returns `attribute1` rounded to `attribute2` places right of the decimal point. If `attribute2` is omitted, `attribute1` is rounded to 0 places. Additionally, `attribute2` can be negative to round off digits left of the decimal point and `attribute2` must be an integer.

Examples

The following example rounds a number to one decimal point:

```
SELECT ROUND(15.193,1) "Round" FROM DUAL;

Round
----------
15.2
```

The following example rounds a number one digit to the left of the decimal point:

```
SELECT ROUND(15.193,-1) "Round" FROM DUAL;
Round
----------
20
```

SIGN

Syntax

```
sign::=SIGN(attribute)
```

Purpose

If `attribute` < 0, `SIGN` returns -1. If `attribute` = 0, the function returns 0. If `attribute` > 0, `SIGN` returns 1. This can be used in validation of measures where only positive numbers are expected.

Example

The following example indicates that the function's argument (-15) is <0:

```
SELECT SIGN(-15) "Sign" FROM DUAL;
 Sign
----------
-1
```

SIN

Syntax

```
sin::=SIN(attribute)
```

Purpose

`SIN` returns the sine of `attribute` (expressed as an angle)

Example

The following example returns the sine of 30 degrees:

```
SELECT SIN(30 * 3.14159265359/180) "Sine of 30 degrees" FROM DUAL;

Sine of 30 degrees
------------------
                .5
```

SINH

Syntax

```
sinh::=SINH(attribute)
```

Purpose

`SINH` returns the hyperbolic sine of attribute.

Example

The following example returns the hyperbolic sine of 1:

```
SELECT SINH(1) "Hyperbolic Sine of 1" FROM DUAL;

Hyperbolic Sine of 1
--------------------
          1.17520119
```

SQRT

Syntax

```
sqrt::=SQRT(attribute)
```

Purpose

`SQRT` returns square root of `attribute`. The value in `attribute` cannot be negative. `SQRT` returns a "real" result.

Example

The following example returns the square root of 26:

```
SELECT SQRT(26) "Square root" FROM DUAL;

Square root
-----------
5.09901951
```

TAN

Syntax

```
tan::=TAN(attrin=bute)
```

Purpose

`TAN` returns the tangent of `attribute` (an angle expressed in radians).

Example

The following example returns the tangent of 135 degrees:

```
SELECT TAN(135 * 3.14159265359/180) "Tangent of 135 degrees" FROM DUAL;

Tangent of 135 degrees
----------------------
                    -1
```

TANH

Syntax

```
tanh::=TANH(attribute)
```

Purpose

`TANH` returns the hyperbolic tangent of `attribute`.

Example

The following example returns the hyperbolic tangent of 5:

```
SELECT TANH(5) "Hyperbolic tangent of 5" FROM DUAL;

Hyperbolic tangent of 5
-----------------------
             .462117157
```

TRUNC (number)

Syntax

```
trunc_number::=TRUNC(attribute, m)
```

Purpose

TRUNC returns attribute truncated to m decimal places. If m is omitted, attribute is truncated to 0 places. m can be negative to truncate (make zero) m digits left of the decimal point.

Example

The following example truncates numbers:

```
SELECT TRUNC(15.79,1) "Truncate"
FROM DUAL;
Truncate
----------
15.7
 SELECT TRUNC(15.79,-1) "Truncate"
FROM DUAL;
 Truncate
----------
10
```

WB_LOOKUP_NUM (on a number)

Syntax

```
 WB_LOOKUP_NUM (table_name
, column_name
, key_column_name
, key_value
)
```

where table_name is the name of the table to perform the lookup on; column_name is the name of the NUMBER column that will be returned, for instance, the result of the lookup; key_column_name is the name of the NUMBER column used as the key to match on in the lookup table; key_value is the value of the key column, for example, the value mapped into the key_column_name with which the match will be done.

Purpose

To perform a key look up that returns a NUMBER value from a database table using a NUMBER column as the matching key.

Example

Consider the following table as a lookup table LKP1:

```
KEYCOLUMN  TYPE_NO  TYPE
10         100123   Car
20         100124   Bike
```

Using this package with the following call:

```
WB_LOOKUP_CHAR('LKP1'
, 'TYPE_NO'
, 'KEYCOLUMN'
, 20
)
```

returns the value of 100124 as output of this transformation. This output is then processed in the mapping as the result of an inline function call.

> **Note:** This function is a row-based key lookup. Set-based lookups are supported when you use the lookup operator.

WB_LOOKUP_NUM (on a varchar2)

Syntax:

```
WB_LOOKUP_CHAR(table_name
, column_name
, key_column_name
, key_value
)
```

where `table_name` is the name of the table to perform the lookup on; `column_name` is the name of the NUMBER column that will be returned (such as the result of the lookup); `key_column_name` is the name of the NUMBER column used as the key to match on in the lookup table; `key_value` is the value of the key column, such as the value mapped into the `key_column_name` with which the match will be done.

Purpose:

To perform a key lookup which returns a NUMBER value from a database table using a VARCHAR2 column as the matching key.

Example

Consider the following table as a lookup table `LKP1`:

```
KEYCOLUMN  TYPE_NO  TYPE
ACV        100123   Car
ACP        100124   Bike
```

Using this package with the following call:

```
WB_LOOKUP_CHAR ('LKP1'
, 'TYPE'
, 'KEYCOLUMN'
, 'ACP'
)
```

returns the value of `100124` as output of this transformation. This output is then processed in the mapping as the result of an inline function call.

> **Note:** This function is a row-based key lookup. Set-based lookups are supported when you use the lookup operator described in Key Lookup Operator on page 26-17.

WB_IS_NUMBER

Syntax

```
WB_IS_NUMBER(attibute, fmt)
```

Purpose

To check whether attribute contains a valid number. The function returns a Boolean value, which is set to true if attribute contains a valid number. Fmt is an optional number format. If fmt is omitted, the number format of your session is used.

You can use this function when you validate the data before loading it into a table. This way the value can be transformed before it reaches the table and causes an error.

Example

WB_IS_NUMBER returns true in PL/SQL if attribute contains a valid number.

WIDTH_BUCKET

Syntax

```
width_bucket::=WIDTH_BUCKET(expr,min_value,max_value,num_buckets)
```

Purpose

For a given expression, WIDTH_BUCKET returns the bucket number into which the value of this expression would fall after being evaluated. WIDTH_BUCKET lets you construct equiwidth histograms, in which the histogram range is divided into intervals that have identical size. Ideally each bucket is a closed-open interval of the real number line. For example, a bucket can be assigned to scores between 10.00 and 19.999... to indicate that 10 is included in the interval and 20 is excluded. This is sometimes denoted as (10, 20).

The argument expr represents the expression for which the histogram is being created. This expression must evaluate to a numeric or datetime value or to a value that can be implicitly converted to a numeric or datetime value. If expr evaluates to null, then the expression returns null. min_value and max_value are expressions that resolve to the end points of the acceptable range for expr. Both of these expressions must also evaluate to numeric or datetime values, and neither can evaluate to null. num_buckets is an expression that resolves to a constant indicating the number of buckets. This expression must evaluate to a positive integer.

When needed, Oracle Database creates an underflow bucket numbered 0 and an overflow bucket numbered num_buckets+1. These buckets handle values less than min_value and more than max_value and are helpful in checking the reasonableness of endpoints.

Example

The following example creates a ten-bucket histogram on the credit_limit column for customers in Switzerland in the sample table oe.customers and returns the bucket number ("Credit Group") for each customer. Customers with credit limits greater than the maximum value are assigned to the overflow bucket, 11:

```
SELECT customer_id, cust_last_name, credit_limit,
   WIDTH_BUCKET(credit_limit, 100, 5000, 10) "Credit Group"
   FROM customers WHERE nls_territory = 'SWITZERLAND'
   ORDER BY "Credit Group";

CUSTOMER_ID CUST_LAST_NAME       CREDIT_LIMIT Credit Group
----------- -------------------- ------------ ------------
        825 Dreyfuss                      500            1
        826 Barkin                        500            1
        853 Palin                         400            1
        827 Siegel                        500            1
```

```
843 Oates                    700           2
844 Julius                   700           2
835 Eastwood                1200           3
840 Elliott                 1400           3
842 Stern                   1400           3
841 Boyer                   1400           3
837 Stanton                 1200           3
836 Berenger                1200           3
848 Olmos                   1800           4
849 Kaurusmdki              1800           4
828 Minnelli                2300           5
829 Hunter                  2300           5
852 Tanner                  2300           5
851 Brown                   2300           5
850 Finney                  2300           5
830 Dutt                    3500           7
831 Bel Geddes              3500           7
832 Spacek                  3500           7
838 Nicholson               3500           7
839 Johnson                 3500           7
833 Moranis                 3500           7
834 Idle                    3500           7
845 Fawcett                 5000          11
846 Brando                  5000          11
847 Streep                  5000          11
```

OLAP Transformations

OLAP transformations enable Warehouse Builder users to load data stored in relational dimensions and cubes into an analytic workspace.

The OLAP transformations provided by Warehouse Builder are:

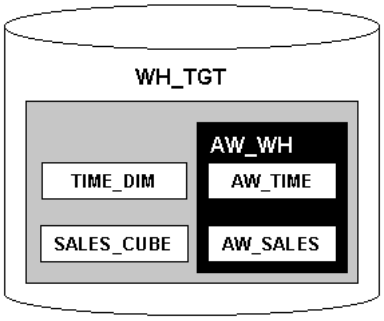
- WB_OLAP_AW_PRECOMPUTE on page 27-85
- WB_OLAP_LOAD_CUBE on page 27-86
- WB_OLAP_LOAD_DIMENSION on page 27-86
- WB_OLAP_LOAD_DIMENSION_GENUK on page 27-87

The `WB_OLAP_LOAD_CUBE`, `WB_OLAP_LOAD_DIMENSION`, and `WB_OLAP_LOAD_DIMENSION_GENUK` transformations are used for cube cloning in Warehouse Builder. Use these OLAP transformations only if your database version is Oracle Database 9*i* or Oracle Database 10*g* Release 1. Starting with Oracle 10*g* Release 2, you can directly deploy dimensions and cubes into an analytic workspace.

The `WB_OLAP_AW_PRECOMPUTE` only works with the Oracle Warehouse Builder 10*g* Release 2.

The examples used to explain these OLAP transformations are based on the scenario depicted in Figure 27–1.

Figure 27–1 Example of OLAP Transformations



The relational dimension TIME_DIM and the relational cube SALES_CUBE are stored in the schema WH_TGT. The analytic workspace AW_WH, into which the dimension and cube are loaded, is also created in the WH_TGT schema.

WB_OLAP_AW_PRECOMPUTE

Syntax

```
wb_olap_aw_precompute::=WB_OLAP_AW_PRECOMPUTE(p_aw_name, p_cube_name,
        p_measure_name, p_allow_parallel_ solve, p_max_job_queues_allocated)
```

where p_aw_name is the name of the AW where cube is deployed, p_cube_name is the name of the cube to solve, p_measure_name is the optional name of a specific measure to solve (if no measure is specified, then all measures will be solved), p_allow_parallel_solve is the boolean to indicate parallelization of solve based on partitioning (performance related parameter), p_max_job_queues_allocated is the number of DBMS jobs to execute in parallel (default value is 0). If 5 is defined and there are 20 partitions then a pool of 5 DBMS jobs will be used to perform the data load.

There is a subtle different between parallel and non-parallel solving. With non-parallel solve, the solve happens synchronously, so when the API call is completed the solve is complete. Parallel solve executes asynchronously, the API call will return with a job id of the job launched. The job will control parallel solving using the max job queues parameter to control its processing. The user may then use the job id to query the all_scheduler_* views to check on the status of the activity.

Purpose

WB_OLAP_AW_PRECOMPUTE is used for solving a non-compressed cube (compressed cubes are auto-solved). The load and solve steps can be done independently. By default, the cube map loads data, then solves (precomputes) the cube. You can load data using the map, then perform the solve at a different point of time (since the solve/build time is the costliest operation).

Example

The following example loads data from the relational cubes MART and SALES_CUBE into a cube called SALES and performs a simple solve execution working serially. This example has parameters for parallel solve and max number of job queues. If parallel solve is performed then an ASYNCHRONOUS solve job is launched and the master job ID is returned through the return function.

```
declare
  rslt varchar2(4000);
```

```
begin
…
  rslt :=wb_olap_aw_precompute('MART','SALES_CUBE','SALES');
…
end;
/
```

WB_OLAP_LOAD_CUBE

Syntax

```
wb_olap_load_cube::=WB_OLAP_LOAD_CUBE(olap_aw_owner, olap_aw_name,
        olap_cube_owner, olap_cube_name, olap_tgt_cube_name)
```

where `olap_aw_owner` is the name of the database schema that owns the analytic workspace; `olap_aw_name` is the name of the analytic workspace that stores the cube data; `olap_cube_owner` is the name of the database schema that owns the related relational cube; `olap_cube_name` is the name of the relational cube; `olap_tgt_cube_name` is the name of the cube in the analytic workspace.

Purpose

`WB_OLAP_LOAD_CUBE` loads data from the relational cube into the analytic workspace. This allows further analysis of the cube data. This is for loading data in an AW cube from a relational cube which it was cloned from. This is a wrapper around some of the procedures in the DBMS_AWM package for loading a cube.

Example

The following example loads data from the relational cube `SALES_CUBE` into a cube called `AW_SALES` in the `AW_WH` analytic workspace:

```
WB_OLAP_LOAD_CUBE('WH_TGT', 'AW_WH', 'WH_TGT', 'SALES_CUBE', 'AW_SALES')
```

WB_OLAP_LOAD_DIMENSION

Syntax

```
wb_olap_load_dimension::=WB_OLAP_LOAD_DIMENSION(olap_aw_owner, olap_aw_name,
        olap_dimension_owner, olap_dimension_name, olap_tgt_dimension_name)
```

where `olap_aw_owner` is the name of the database schema that owns the analytic workspace; `olap_aw_name` is the name of the analytic workspace that stores the dimension data; `olap_dimension_owner` is the name of the database schema in which the related relational dimension is stored; `olap_dimension_name` is the name of the relational dimension; `olap_tgt_dimension_name` is the name of the dimension in the analytic workspace.

Purpose

`WB_OLAP_LOAD_DIMENSION` loads data from the relational dimension into the analytic workspace. This allows further analysis of the dimension data. This is for loading data in an AW dimension from a relational dimension which it was cloned from. This is a wrapper around some of the procedures in the DBMS_AWM package for loading a dimension.

Example

The following example loads the data from the relational dimension TIME_DIM into a dimension called AW_TIME in the analytic workspace AW_WH:

```
WB_OLAP_LOAD_DIMENSION('WH_TGT', 'AW_WH', 'WH_TGT', 'TIME_DIM', 'AW_TIME')
```

WB_OLAP_LOAD_DIMENSION_GENUK

Syntax

```
wb_olap_load_dimension_genuk::=WB_OLAP_LOAD_DIMENSION_GENUK(olap_aw_owner,
        olap_aw_name, olap_dimension_owner, olap_dimension_name,
        olap_tgt_dimension_name)
```

where olap_aw_owner is the name of the database schema that owns the analytic workspace; olap_aw_name is the name of the analytic workspace that stores the dimension data; olap_dimension_owner is the name of the database schema in which the related relational dimension is stored; olap_dimension_name is the name of the relational dimension; olap_tgt_dimension_name is the name of the dimension in the analytic workspace.

Purpose

WB_OLAP_LOAD_DIMENSION_GENUK loads data from the relational dimension into the analytic workspace. Unique dimension identifiers will be generated across all levels. This is for loading data in an AW dimension from a relational dimension which it was cloned from. This is a wrapper around some of the procedures in the DBMS_AWM package for loading a dimension.

If a cube has been cloned and if you select YES for the Generate Surrogate Keys for Dimensions option, then when you want to reload the dimensions, you should use the WB_OLAP_LOAD_DIMENSION_GENUK procedure. This procedure generates surrogate identifiers for all levels in the AW, because the AW requires all level identifiers to be unique across all levels of a dimension.

Example

Consider an example in which the dimension TIME_DIM has been deployed to the OLAP server by cloning the cube. The parameter generate surrogate keys for Dimension was set to true. To now reload data from the relational dimension TIME_DIM into the dimension AW_TIME in the analytic workspace AW_WH, use the following syntax.

```
WB_OLAP_LOAD_CUBE('WH_TGT', 'AW_WH', 'WH_TGT', 'TIME_DIM', 'AW_TIME')
```

Other Transformations

Other transformations included with Warehouse Builder enable you to perform various functions which are not restricted to certain data types. This section describes those types. Other transformations provided by Warehouse Builder are:

- DEPTH on page 27-88
- DUMP on page 27-89
- EMPTY_BLOB, EMPTY_CLOB on page 27-90
- NLS_CHARSET_DECL_LEN on page 27-90

DEPTH

Syntax

```
depth::=DEPTH(correlation_integer)
```

Purpose

`DEPTH` is an ancillary function used only with the UNDER_PATH and EQUALS_PATH conditions. It returns the number of levels in the path specified by the UNDER_PATH condition with the same correlation variable. The `correlation_integer` can be any NUMBER integer. Use it to correlate this ancillary function with its primary condition if the statement contains multiple primary conditions. Values less than 1 are treated as 1.

Example

The EQUALS_PATH and UNDER_PATH conditions can take two ancillary functions, DEPTH and PATH. The following example shows the use of both ancillary functions. The example assumes the existence of the XMLSchema `warehouses.xsd`.

```
SELECT PATH(1), DEPTH(2)
   FROM RESOURCE_VIEW
   WHERE UNDER_PATH(res, '/sys/schemas/OE', 1)=1
     AND UNDER_PATH(res, '/sys/schemas/OE', 2)=1;

PATH(1)                          DEPTH(2)
-------------------------------- --------
/www.oracle.com                         1

/www.oracle.com/xwarehouses.xsd         2
```

DUMP

Syntax

```
dump::=DUMP(expr,return_fmt,start_position,length)
```

Purpose

DUMP returns a VARCHAR2 value containing the data type code, length in bytes, and internal representation of expr. The returned result is always in the database character set. The argument return_fmt specifies the format of the return value and can have any of the following values:

- 8 returns result in octal notation.
- 10 returns result in decimal notation.
- 16 returns result in a hexadecimal notation.
- 17 returns result as single characters.

By default, the return value contains no character set information. To retrieve the character set name of expr, add 1000 to any of the preceding format values. For example, a return_fmt of 1008 returns the result in octal and provides the character set name of expr.

The arguments start_position and length combine to determine which portion of the internal representation to return. The default is to return the entire internal representation in decimal notation. If expr is null, then this function returns null.

> **Note:** This function does not support CLOB data directly. However, CLOBs can be passed in as arguments through implicit data conversion.

Example

The following examples show how to extract dump information from a string expression and a column:

```
SELECT DUMP('abc', 1016)
   FROM DUAL;

DUMP('ABC',1016)
----------------------------------------
Typ=96 Len=3 CharacterSet=WE8DEC: 61,62,63

SELECT DUMP(last_name, 8, 3, 2) "OCTAL"
   FROM employees
   WHERE last_name = 'Hunold';

OCTAL
----------------------------------------------------------------------
Typ=1 Len=6: 156,157

SELECT DUMP(last_name, 10, 3, 2) "ASCII"
   FROM employees
   WHERE last_name = 'Hunold';

ASCII
----------------------------------------------------------------------
Typ=1 Len=6: 110,111
```

EMPTY_BLOB, EMPTY_CLOB

Syntax

```
empty_blob::=EMPTY_BLOB()
empty_clob::=EMPTY_CLOB()
```

Purpose

EMPTY_BLOB and EMPTY_CLOB return an empty LOB locator that can be used to initialize a LOB variable or, in an INSERT or UPDATE statement, to initialize a LOB column or attribute to EMPTY. EMPTY means that the LOB is initialized, but not populated with data. You must initialize a LOB attribute that is part of an object type before you can access and populate it.

Note: You cannot use the locator returned from this function as a parameter to the DBMS_LOB package or the OCI.

Example

The following example initializes the ad_photo column of the sample pm.print_media table to EMPTY:

```
UPDATE print_media SET ad_photo = EMPTY_BLOB();
```

NLS_CHARSET_DECL_LEN

Syntax

```
nls_charset_decl_len::=NLS_CHARSET_DECL_LEN(byte_count,charset_id)
```

Purpose

NLS_CHARSET_DECL_LEN returns the declaration width (in number of characters) of an NCHAR column. The byte_count argument is the width of the column. The charset_id argument is the character set ID of the column.

Example

The following example returns the number of characters that are in a 200-byte column when you are using a multibyte character set:

```
SELECT NLS_CHARSET_DECL_LEN(200, nls_charset_id('ja16eucfixed')) FROM DUAL;

NLS_CHARSET_DECL_LEN(200,NLS_CHARSET_ID('JA16EUCFIXED'))
-------------------------------------------------------
                                                    100
```

NLS_CHARSET_ID

Syntax

```
nls_charset_id::= NLS_CHARSET_ID(text)
```

Purpose

NLS_CHARSET_ID returns the character set ID number corresponding to character set name text. The text argument is a run-time VARCHAR2 value. The text value 'CHAR_CS' returns the database character set ID number of the server. The text value 'NCHAR_CS' returns the national character set ID number of the server.

Invalid character set names return null.

Example

The following example returns the character set ID number of a character set:

```
SELECT NLS_CHARSET_ID('ja16euc') FROM DUAL;

NLS_CHARSET_ID('JA16EUC')
--------------------------
                       830
```

NLS_CHARSET_NAME

Syntax

```
nls_charset_name::= NLS_CHARSET_NAME(number)
```

Purpose

NLS_CHARSET_NAME returns the name of the character set corresponding to ID number. The character set name is returned as a VARCHAR2 value in the database character set.

If number is not recognized as a valid character set ID, then this function returns null.

Example

The following example returns the character set corresponding to character set ID number 2:

```
SELECT NLS_CHARSET_NAME(2) FROM DUAL;

NLS_CH
--------
WE8DEC
```

NULLIF

Syntax

```
nullif::=NULLIF(expr1,expr2)
```

Purpose

NULLIF compares expr1 and expr2. If they are equal, then the function returns null. If they are not equal, then the function returns expr1. You cannot specify the literal NULL for expr1.

If both arguments are numeric data types, then Oracle Database determines the argument with the higher numeric precedence, implicitly converts the other argument to that data type, and returns that data type. If the arguments are not numeric, then they must be of the same data type, or Oracle returns an error.

The NULLIF function is logically equivalent to the following CASE expression:

```
CASE WHEN expr1 = expr 2 THEN NULL ELSE expr1 END
```

Example

The following example selects those employees from the sample schema hr who have changed jobs since they were hired, as indicated by a job_id in the job_history table different from the current job_id in the employees table:

```
SELECT e.last_name, NULLIF(e.job_id, j.job_id) "Old Job ID"
   FROM employees e, job_history j
   WHERE e.employee_id = j.employee_id
   ORDER BY last_name;

LAST_NAME                 Old Job ID
------------------------- ----------
De Haan                   AD_VP
Hartstein                 MK_MAN
Kaufling                  ST_MAN
Kochhar                   AD_VP
Kochhar                   AD_VP
Raphaely                  PU_MAN
Taylor                    SA_REP
Taylor
Whalen                    AD_ASST
Whalen
```

NVL

Syntax

```
nvl::=NVL(attribute1, attribute2)
```

Purpose

If attribute1 is null, NVL returns attribute2. If attribute1 is not null, then NVL returns attribute1. The arguments attribute1 and attribute2 can be any data type. If their data types are different, attribute2 is converted to the data type of attribute1 before they are compared. Warehouse Builder provides three variants of NVL to support all input values.

The data type of the return value is always the same as the data type of attribute1, unless attribute1 is character data, in which case the return value data type is VARCHAR2, in the character set of attribute1.

Example

The following example returns a list of employee names and commissions, substituting "Not Applicable" if the employee receives no commission:

```
SELECT last_name, NVL(TO_CHAR(commission_pct), 'Not Applicable') "COMMISSION"
FROM employees
WHERE last_name LIKE 'B%';

LAST_NAME                 COMMISSION
------------------------- ----------------------------------------
Baer                      Not Applicable
Baida                     Not Applicable
Banda                     .11
```

```
Bates                     .16
Bell                      Not Applicable
Bernstein                 .26
Bissot                    Not Applicable
Bloom                     .21
Bull                      Not Applicable
```

NVL2

Syntax

```
nvl2::=NVL2(expr1,expr2,expr3)
```

Purpose

NVL2 lets you determine the value returned by a query based on whether a specified expression is null or not null. If expr1 is not null, then NVL2 returns expr2. If expr1 is null, then NVL2 returns expr3. The argument expr1 can have any data type. The arguments expr2 and expr3 can have any data types except LONG.

If the data types of expr2 and expr3 are different:

- If expr2 is character data, then Oracle Database converts expr3 to the data type of expr2 before comparing them unless expr3 is a null constant. In that case, a data type conversion is not necessary. Oracle returns VARCHAR2 in the character set of expr2.
- If expr2 is numeric, then Oracle determines which argument has the highest numeric precedence, implicitly converts the other argument to that data type, and returns that data type.

Example

The following example shows whether the income of some employees is made up of salary plus commission, or just salary, depending on whether the commission_pct column of employees is null or not.

```
SELECT last_name, salary, NVL2(commission_pct,
   salary + (salary * commission_pct), salary) income
   FROM employees WHERE last_name like 'B%'
   ORDER BY last_name;

LAST_NAME                     SALARY     INCOME
------------------------- ---------- ----------
Baer                           10000      10000
Baida                           2900       2900
Banda                           6200       6882
Bates                           7300       8468
Bell                            4000       4000
Bernstein                       9500      11970
Bissot                          3300       3300
Bloom                          10000      12100
Bull                            4100       4100
```

ORA_HASH

Syntax

```
ora_hash::=ORA_HASH(expr,max_bucket,seed_value)
```

Purpose

ORA_HASH is a function that computes a hash value for a given expression. This function is useful for operations such as analyzing a subset of data and generating a random sample. The function returns a NUMBER value.

The `expr` argument determines the data for which you want Oracle Database to compute a hash value. There are no restrictions on the type or length of data represented by `expr`, which commonly resolves to a column name. The argument `max_bucket` is optional and it determines the maximum bucket value returned by the hash function. You can specify any value between 0 and 4294967295. The default is 4294967295. The optional `seed_value` argument enables Oracle to produce many different results for the same set of data. Oracle applies the hash function to the combination of `expr` and `seed_value`. You can specify any value between 0 and 4294967295. The default is 0.

Example

The following example creates a hash value for each combination of customer ID and product ID in the ***sh.sales*** table, divides the hash values into a maximum of 100 buckets, and returns the sum of the `amount_sold` values in the first bucket (bucket 0). The third argument (5) provides a seed value for the hash function. You can obtain different hash results for the same query by changing the seed value.

```
SELECT SUM(amount_sold) FROM sales
   WHERE ORA_HASH(CONCAT(cust_id, prod_id), 99, 5) = 0;

SUM(AMOUNT_SOLD)
----------------
            7315
```

The following example retrieves a subset of the data in the `sh.sales` table by specifying 10 buckets (0 to 9) and then returning the data from bucket 1. The expected subset is about 10% of the rows (the sales table has 960 rows):

```
SELECT * FROM sales
   WHERE ORA_HASH(cust_id, 9) = 1;

   PROD_ID    CUST_ID TIME_ID   C   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- - ---------- ------------- -----------
      2510       6950 01-FEB-98 S       9999             2          78
      9845       9700 04-FEB-98 C       9999            17         561
      3445      33530 07-FEB-98 T       9999             2         170
. . .
       740      22200 13-NOV-00 S       9999             4         156
      9425       4750 29-NOV-00 I       9999            11         979
      1675      46750 29-NOV-00 S       9999            19        1121
```

PATH

Syntax

```
path::=PATH(correlation_integer)
```

Purpose

`PATH` is an ancillary function used only with the UNDER_PATH and EQUALS_PATH conditions. It returns the relative path that leads to the resource specified in the parent condition. The `correlation_integer` can be any NUMBER integer and is used to

correlate this ancillary function with its primary condition. Values less than 1 are treated as 1.

Example

Please refer to the example for DEPTH. This example uses both the ancillary functions of the EQUALS_PATH and UNDER_PATH.

SYS_CONTEXT

Syntax

```
sys_context::=SYS_CONTEXT(namespace,parameter,length)
```

Purpose

SYS_CONTEXT returns the value of parameter associated with the context namespace. You can use this function in both SQL and PL/SQL statements. For namespace and parameter, you can specify either a string or an expression that resolves to a string designating a namespace or an attribute. The context namespace must already have been created, and the associated parameter and its value must also have been set using the DBMS_SESSION.set_context procedure. The namespace must be a valid SQL identifier. The parameter name can be any string. It is not case sensitive, but it cannot exceed 30 bytes in length.

The data type of the return value is VARCHAR2. The default maximum size of the return value is 256 bytes. You can override this default by specifying the optional length parameter, which must be a NUMBER or a value that can be implicitly converted to NUMBER. The valid range of values is 1 to 4000 bytes. If you specify an invalid value, then Oracle Database ignores it and uses the default.

Oracle provides a built-in namespace called USERENV, which describes the current session. For a description of the predefined parameters of the namespace USERENV, refer Table 7-11 of the *Oracle Database SQL Reference*.

Example

The following statement returns the name of the user who logged onto the database:

```
CONNECT OE/OE
SELECT SYS_CONTEXT ('USERENV', 'SESSION_USER')
   FROM DUAL;

SYS_CONTEXT ('USERENV', 'SESSION_USER')
----------------------------------------------------
OE
```

The following hypothetical example returns the group number that was set as the value for the attribute group_no in the PL/SQL package that was associated with the context hr_apps when hr_apps was created:

```
SELECT SYS_CONTEXT ('hr_apps', 'group_no') "User Group"
   FROM DUAL;
```

SYS_GUID

Syntax

```
sys_guid::=SYS_GUID()
```

Purpose

SYS_GUID generates and returns a globally unique identifier (RAW value) made up of 16 bytes. On most platforms, the generated identifier consists of a host identifier, a process or thread identifier of the process or thread invoking the function, and a nonrepeating value (sequence of bytes) for that process or thread.

Example

The following example adds a column to the sample table hr.locations, inserts unique identifiers into each row, and returns the 32-character hexadecimal representation of the 16-byte RAW value of the global unique identifier:

```
ALTER TABLE locations ADD (uid_col RAW(32));

UPDATE locations SET uid_col = SYS_GUID();

SELECT location_id, uid_col FROM locations;

LOCATION_ID UID_COL
----------- --------------------------------------
       1000 7CD5B7769DF75CEFE034080020825436
       1100 7CD5B7769DF85CEFE034080020825436
       1200 7CD5B7769DF95CEFE034080020825436
       1300 7CD5B7769DFA5CEFE034080020825436
. . .
```

SYS_TYPEID

Syntax

```
sys_typeid::=SYS_TYPEID(object_type_value)
```

Purpose

SYS_TYPEID returns the typeid of the most specific type of the operand. This value is used primarily to identify the type-discriminant column underlying a substitutable column. For example, you can use the value returned by SYS_TYPEID to build an index on the type-discriminant column. You can use the SYS_TYPEID function to create an index on the type-discriminant column of a table.

You can use this function only on object type operands. All final root object types—that is, final types not belonging to a type hierarchy—have a null typeid. Oracle Database assigns to all types belonging to a type hierarchy a unique non-null typeid.

UID

Syntax

```
uid::=UID()
```

Purpose

UID returns an integer that uniquely identifies the session user, such as the user who is logged on when running the session containing the transformation. In a distributed SQL statement, the UID function identifies the user on your local database.

Use this function when logging audit information into a target table to identify the user running the mappings.

Example

The following returns the local database user id logged into this session:

```
SELECT uid FROM dual;

       UID
----------
        55
```

USER

Syntax

```
user::=USER()
```

Purpose

USER returns the name of the session user (the user who logged on) with the data type VARCHAR2.

Oracle compares values of this function with blank-padded comparison semantics. In a distributed SQL statement, the UID and USER functions identify the user on your local database.

Use this function when logging audit information into a target table to identify the user running the mappings.

Example

The following example returns the local database user logged into this session:

```
SELECT user FROM dual;

USER
------------------------------
OWB9I_RUN
```

USERENV

Syntax

```
userenv::=USERENV(parameter)
```

Purpose

> **Note:** USERENV is a legacy function that is retained for backward compatibility. Oracle recommends that you use the SYS_CONTEXT function with the built-in USERENV namespace for current functionality.

USERENV returns information about the current session. This information can be useful for writing an application-specific audit trail table or for determining the language-specific characters currently used by your session. You cannot use USERENV in the condition of a CHECK constraint. Table 27–1 describes the values for the parameter argument. All calls to USERENV return VARCHAR2 data except for calls

with the SESSIONID, ENTRYID, and COMMITSCN parameters, which return NUMBER.

Table 27–1 Parameters of the USERENV function

Parameter	Return Value
CLIENT_INFO	CLIENT_INFO returns up to 64 bytes of user session information that can be stored by an application using the DBMS_APPLICATION_INFO package. **Caution:** Some commercial applications may be using this context value. Please refer to the applicable documentation for those applications to determine what restrictions they may impose on use of this context area.
ENTRYID	The current audit entry number. The audit entryid sequence is shared between fine-grained audit records and regular audit records. You cannot use this attribute in distributed SQL statements
ISDBA	ISDBA returns 'TRUE' if the user has been authenticated as having DBA privileges either through the operating system or through a password file.
LANG	LANG returns the ISO abbreviation for the language name, a shorter form than the existing 'LANGUAGE' parameter.
LANGUAGE	LANGUAGE returns the language and territory used by the current session along with the database character set in this form: `language_territory.characterset`
SESSIONID	SESSIONID returns the auditing session identifier. You cannot specify this parameter in distributed SQL statements.
TERMINAL	TERMINAL returns the operating system identifier for the terminal of the current session. In distributed SQL statements, this parameter returns the identifier for your local session. In a distributed environment, this parameter is supported only for remote SELECT statements, not for remote INSERT, UPDATE, or DELETE operations.

Example

The following example returns the `LANGUAGE` parameter of the current session:

```
SELECT USERENV('LANGUAGE') "Language" FROM DUAL;

Language
----------------------------------
AMERICAN_AMERICA.WE8ISO8859P1
```

VSIZE

Syntax

```
vsize::=VSIZE(expr)
```

Purpose

`VSIZE` returns the number of bytes in the internal representation of expr. If expr is null, then this function returns null. This function does not support CLOB data directly. However, CLOBs can be passed in as arguments through implicit data conversion.

Example

The following example returns the number of bytes in the last_name column of the employees in department 10:

```
SELECT last_name, VSIZE (last_name) "BYTES"
  FROM employees
  WHERE department_id = 10;

LAST_NAME            BYTES
---------------  ----------
Whalen                    6
```

Spatial Transformations

Spatial Transformation is an integrated set of functions and procedures that enables spatial data to be stored, accessed, and analyzed quickly and efficiently in an Oracle Database. Spatial transformations included with Warehouse Builder are:

- SDO_AGGR_CENTROID on page 27-99
- SDO_AGGR_CONVEXHULL on page 27-100
- SDO_AGGR_MBR on page 27-100
- SDO_AGGR_UNION on page 27-100

SDO_AGGR_CENTROID

Syntax

```
sdo_aggr_centroid::= SDO_AGGR_CENTROID(AggregateGeometry SDOAGGRTYPE)
```

Purpose

SDO_AGGR_CENTROID returns a geometry object that is the centroid (center of gravity) of the specified geometry objects. The behavior of the function depends on whether the geometry objects are all polygons, all points, or a mixture of polygons and points:

- If the geometry objects are all polygons, the centroid of all the objects is returned.
- If the geometry objects are all points, the centroid of all the objects is returned.
- If the geometry objects are a mixture of polygons and points (specifically, if they include at least one polygon and at least one point), any points are ignored, and the centroid of all the polygons is returned.

The result is weighted by the area of each polygon in the geometry objects. If the geometry objects are a mixture of polygons and points, the points are not used in the calculation of the centroid. If the geometry objects are all points, the points have equal weight.

Example

The following example returns the centroid of the geometry objects in the `COLA_MARKETS` table.

```
SELECT SDO_AGGR_CENTROID(SDOAGGRTYPE(shape, 0.005))
  FROM cola_markets;
```

```
SDO_AGGR_CENTROID(SDOAGGRTYPE(SHAPE,0.005))(SDO_GTYPE, SDO_SRID, SDO_POINT
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(5.21295938, 5.00744233, NULL), NULL, NULL)
```

SDO_AGGR_CONVEXHULL

Syntax

```
sod_aggr_convexhull::= SDO_AGGR_CONVEXHULL(AggregateGeometry SDOAGGRTYPE)
```

Purpose

SDO_AGGR_CONVEXHULL returns a geometry object that is the convex hull of the specified geometry objects.

Example

The following example returns the convex hull of the geometry objects in the COLA_MARKETS table.

```
SELECT SDO_AGGR_CONVEXHULL(SDOAGGRTYPE(shape, 0.005))
  FROM cola_markets;

SDO_AGGR_CONVEXHULL(SDOAGGRTYPE(SHAPE,0.005))(SDO_GTYPE, SDO_SRID, SDO_POI
--------------------------------------------------------------------------------
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_
ARRAY(8, 1, 10, 7, 10, 11, 8, 11, 6, 11, 1, 7, 1, 1, 8, 1))
```

SDO_AGGR_MBR

Syntax

```
sod_aggr_mbr::= SDO_AGGR_MBR(geom SDO_GEOMETRY)
```

Purpose

SDO_AGGR_MBR returns the minimum bounding rectangle (MBR) of the specified geometries, that is, a single rectangle that minimally encloses the geometries.

Example

The following example returns the minimum bounding rectangle of the geometry objects in the COLA_MARKETS table.

```
SELECT SDO_AGGR_MBR(shape) FROM cola_markets;

SDO_AGGR_MBR(C.SHAPE)(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SD
--------------------------------------------------------------------------------
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_
ARRAY(1, 1, 10, 11))
```

SDO_AGGR_UNION

Syntax

```
SDO_AGGR_UNION(
```

```
    AggregateGeometry SDOAGGRTYPE

    ) RETURN SDO_GEOMETRY;
```

Purpose

SDO_AGGR_UNION returns a geometry object that is the topological union (OR operation) of the specified geometry objects.

Example

The following example returns the union of the first three geometry objects in the COLA_MARKETS table (that is, all except cola_d).

```
SELECT SDO_AGGR_UNION(
  SDOAGGRTYPE(c.shape, 0.005))
  FROM cola_markets c
  WHERE c.name < 'cola_d';

SDO_AGGR_UNION(SDOAGGRTYPE(C.SHAPE,0.005))(SDO_GTYPE, SDO_SRID, SDO_POINT(
--------------------------------------------------------------------------------
SDO_GEOMETRY(2007, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 2, 11, 1003, 1), SDO
_ORDINATE_ARRAY(8, 11, 6, 9, 8, 7, 10, 9, 8, 11, 1, 7, 1, 1, 5, 1, 8, 1, 8, 6, 5,
7, 1, 7))
```

Streams Transformations

The Streams transformations category contains one transformation called REPLICATE. The following section describes this transformation.

REPLICATE

Syntax

```
REPLICATE(lcr, conflict_resolution)
```

where `lcr` stands for Logical Change Record and encapsulates the DML change. Its data type is SYS.LCR$_ROW_RECORD. `conflict_resolution` is a Boolean variable. If its value is TRUE, any conflict resolution defined for the table will be used to resolve conflicts resulting from the execution of the LCR. For more information about conflict resolution, refer to *Oracle Streams Replication Administrators' Guide*.

Purpose

REPLICATE is used to replicate a DML change (INSERT, UPDATE, or DELETE) that has occurred on a table in the source system on an identical table in the target system. The table in the target system should be identical to the table in the source system in the following respects:.

- The name of the schema that contains the target table should be the same as the name of the schema that contains the source table.
- The name of the target table should the same as the name of the source table.
- The structure of the target table should be the same as that of the source table. The structure includes the number, name, and data type of the columns in the table.

Example

Consider a table T1(c1 varchar2(10), c2 number primary key) in schema S on the source system and an identical table in the target system. Consider the following insert operation on the table T1 on the source system

```
insert into T1 values ('abcde', 10)
```

An LCR representing the change following the above insert of a row on the table T1 in the source system will have the following details

```
LCR.GET_OBJECT_OWNER will be 'S'
LCR.GET_OBJECT_NAME will be 'T1'
LCR.GET_COMMAND_TYPE will be 'INSERT'
LCR.GET_VALUE('c1', 'new') will have the value for the column 'c1', that is,
'abcde'
LCR.GET_VALUE('c2', 'new') will have the value for the column 'c2', that is, 10
```

Such an LCR will be created and enqueued by a Streams Capture Process on the source system that captures changes on table S.T1

REPLICATE(lcr, true) - will result in a row ('abcde', 10) being inserted into the table T1 on the target system.

Note: Using this approach will not provide lineage information. If lineage is important, then do not use this function. Use the more direct approach of using an LCRCast operator bound to the source table and a table operator bound to the target table and connecting the attributes of these two operators with the same name ('Match by name'). Further information on LCR (Logical Change Record) is available in Oracle Database 10*g* Documentation (Information Integration)

XML Transformations

XML transformations provide Warehouse Builder users with functionality to perform transformations on XML objects. These transformations enable Warehouse Builder users to load and transform XML documents and Oracle AQs.

To enable loading of XML sources, Warehouse Builder provides access to the database XML functionality through custom functions, as detailed in this chapter.

Following are the XML transformations:

- EXISTSNODE on page 27-103
- EXTRACT on page 27-103
- EXTRACTVALUE on page 27-104
- SYS_XMLAGG on page 27-104
- SYS_XMLGEN on page 27-105
- WB_XML_LOAD on page 27-106
- WB_XML_LOAD_F on page 27-106
- XMLCONCAT on page 27-107
- XMLSEQUENCE on page 27-108
- XMLTRANSFORM on page 27-109

EXISTSNODE

Syntax

```
existsnode::=EXISTSNODE(XMLType_instance,XPath_string,namespace_string)
```

Purpose

EXISTSNODE determines whether traversal of an XML document using a specified path results in any nodes. It takes as arguments the XMLType instance containing an XML document and a VARCHAR2 XPath string designating a path. The optional namespace_string must resolve to a VARCHAR2 value that specifies a default mapping or namespace mapping for prefixes, which Oracle Database uses when evaluating the XPath expression(s).

This function returns a NUMBER value. It returns 0 if no nodes remain after applying the XPath traversal on the document. It returns 1 if any nodes remain.

Example

The following example tests for the existence of the /Warehouse/Dock node in the XML path of the warehouse_spec column of the sample table oe.warehouses:

```
SELECT warehouse_id, warehouse_name
   FROM warehouses
   WHERE EXISTSNODE(warehouse_spec, '/Warehouse/Docks') = 1;

WAREHOUSE_ID WAREHOUSE_NAME
------------ ----------------------------------
           1 Southlake, Texas
           2 San Francisco
           4 Seattle, Washington
```

EXTRACT

Syntax

```
extract::=EXTRACT(XMLType_instance,XPath_string,namespace_string)
```

Purpose

EXTRACT is similar to the EXISTSNODE function. It applies a VARCHAR2 XPath string and returns an XMLType instance containing an XML fragment. The optional namespace_string must resolve to a VARCHAR2 value that specifies a default mapping or namespace mapping for prefixes, which Oracle Database uses when evaluating the XPath expression(s).

Example

The following example extracts the value of the /Warehouse/Dock node of the XML path of the warehouse_spec column in the sample table oe.warehouses:

```
SELECT warehouse_name, EXTRACT(warehouse_spec, '/Warehouse/Docks')
   "Number of Docks"
   FROM warehouses
   WHERE warehouse_spec IS NOT NULL;

WAREHOUSE_NAME       Number of Docks
-------------------- --------------------
Southlake, Texas          <Docks>2</Docks>
```

```
San Francisco             <Docks>1</Docks>
New Jersey                <Docks/>
Seattle, Washington       <Docks>3</Docks>
```

EXTRACTVALUE

Syntax

```
extractvalue::=EXTRACTVALUE(XMLType_instance,XPath_string,namespace_string)
```

Purpose

The `EXTRACTVALUE` function takes as arguments an `XMLType` instance and an XPath expression and returns a scalar value of the resultant node. The result must be a single node and be either a text node, attribute, or element. If the result is an element, then the element must have a single text node as its child, and it is this value that the function returns. If the specified XPath points to a node with more than one child, or if the node pointed to has a non-text node child, then Oracle returns an error. The optional `namespace_string` must resolve to a VARCHAR2 value that specifies a default mapping or namespace mapping for prefixes, which Oracle uses when evaluating the XPath expression(s).

For documents based on XML schemas, if Oracle can infer the type of the return value, then a scalar value of the appropriate type is returned. Otherwise, the result is of type VARCHAR2. For documents that are not based on XML schemas, the return type is always VARCHAR2.

Example

The following example takes as input the same arguments as the example for EXTRACT. Instead of returning an XML fragment, as does the EXTRACT function, it returns the scalar value of the XML fragment:

```
SELECT warehouse_name,
   EXTRACTVALUE(e.warehouse_spec, '/Warehouse/Docks')
   "Docks"
   FROM warehouses e
   WHERE warehouse_spec IS NOT NULL;

WAREHOUSE_NAME       Docks
-------------------- ------------
Southlake, Texas     2
San Francisco        1
New Jersey
Seattle, Washington  3
```

SYS_XMLAGG

Syntax

```
sys_xmlagg::=SYS_XMLAGG(expr,fmt)
```

Purpose

`SYS_XMLAGG` aggregates all of the XML documents or fragments represented by `expr` and produces a single XML document. It adds a new enclosing element with a default name ROWSET. If you want to format the XML document differently, then specify `fmt`, which is an instance of the `XMLFormat` object.

Example

The following example uses the SYS_XMLGEN function to generate an XML document for each row of the sample table `employees` where the employee's last name begins with the letter R, and then aggregates all of the rows into a single XML document in the default enclosing element ROWSET:

```
SELECT SYS_XMLAGG(SYS_XMLGEN(last_name))
   FROM employees
   WHERE last_name LIKE 'R%';

SYS_XMLAGG(SYS_XMLGEN(LAST_NAME))
--------------------------------------------------------------------
<ROWSET>
  <LAST_NAME>Raphaely</LAST_NAME>
  <LAST_NAME>Rogers</LAST_NAME>
  <LAST_NAME>Rajs</LAST_NAME>
  <LAST_NAME>Russell</LAST_NAME>
</ROWSET>
```

SYS_XMLGEN

Syntax

```
sys_xmlgen::=SYS_XMLGEN(expr,fmt)
```

Purpose

`SYS_XMLGEN` takes an expression that evaluates to a particular row and column of the database, and returns an instance of type `XMLType` containing an XML document. The `expr` can be a scalar value, a user-defined type, or an `XMLType` instance.

If `expr` is a scalar value, then the function returns an XML element containing the scalar value. If `expr` is a type, then the function maps the user-defined type attributes to XML elements. If `expr` is an XMLType instance, then the function encloses the document in an XML element whose default tag name is ROW.

By default the elements of the XML document match the elements of `expr`. For example, if `expr` resolves to a column name, then the enclosing XML element will be the same column name. If you want to format the XML document differently, then specify `fmt`, which is an instance of the `XMLFormat` object.

Example

The following example retrieves the employee email ID from the sample table `oe.employees` where the `employee_id` value is 205, and generates an instance of an `XMLType` containing an XML document with an `EMAIL` element.

```
SELECT SYS_XMLGEN(email)
   FROM employees
   WHERE employee_id = 205;

SYS_XMLGEN(EMAIL)
--------------------------------------------------------------------
<EMAIL>SHIGGINS</EMAIL>
```

WB_XML_LOAD

Syntax:

```
WB_XML_LOAD(control_file)
```

Purpose

This program unit extracts and loads data from XML documents into database targets. The `control_file`, an XML document, specifies the source of the XML documents, the targets, and any runtime controls. After the transformation has been defined, a mapping in Warehouse Builder calls the transformation as a pre-map or post-map trigger.

Example

The following example illustrates a script that can be used to implement a Warehouse Builder transformation that extracts data from an XML document stored in the file products.xml and loads it into the target table called books:

```
begin
wb_xml_load('<OWBXMLRuntime>'
||
'<XMLSource>'
||
' <file>\ora817\GCCAPPS\products.xml</file>'
||
'</XMLSource>'
||
'<targets>'
||
' <target XSLFile="\ora817\XMLstyle\GCC.xsl">books</target>'
||
'</targets>'
||
'</OWBXMLRuntime>'
);
end;
```

For more information on control files, see the *Oracle Warehouse Builder User's Guide*.

WB_XML_LOAD_F

Syntax

```
WB_XML_LOAD_F(control_file)
```

Purpose

`WB_XML_LOAD_F` extracts and loads data from XML documents into database targets. The function returns the number of XML documents read during the load. The `control_file`, itself an XML document, specifies the source of the XML documents, the targets, and any runtime controls. After the transformation has been defined, a mapping in Warehouse Builder calls the transformation as a pre-map or post-map trigger.

Example

The following example illustrates a script that can be used to implement a Warehouse Builder transformation that extracts data from an XML document stored in the file `products.xml` and loads it into the target table books:

```
begin
wb_xml_load_f('<OWBXMLRuntime>'
||
'<XMLSource>'
||
' <file>\ora817\GCCAPPS\products.xml</file>'
||
'</XMLSource>'
||
'<targets>'
||
' <target XSLFile="\ora817\XMLstyle\GCC.xsl">books</target>'
||
'</targets>'
||
'</OWBXMLRuntime>'
);
end;
```

For more information on the types handled and detailed information on `control_files`, see the *Oracle Warehouse Builder User's Guide*.

XMLCONCAT

Syntax

```
xmlconcat::=XMLCONCAT(XMLType_instance)
```

Purpose

`XMLCONCAT` takes as input a series of `XMLType` instances, concatenates the series of elements for each row, and returns the concatenated series. `XMLCONCAT` is the inverse of XMLSEQUENCE. Null expressions are dropped from the result. If all the value expressions are null, then the function returns null.

Example

The following example creates XML elements for the first and last names of a subset of employees, and then concatenates and returns those elements:

```
SELECT XMLCONCAT(XMLELEMENT("First", e.first_name),
   XMLELEMENT("Last", e.last_name)) AS "Result"
   FROM employees e
   WHERE e.employee_id > 202;

Result
--------------------------------------------------------------------
<First>Susan</First>
<Last>Mavris</Last>

<First>Hermann</First>
<Last>Baer</Last>

<First>Shelley</First>
<Last>Higgins</Last>
```

```
<First>William</First>
<Last>Gietz</Last>
```

XMLSEQUENCE

Syntax

```
xmlsequence:=xmlsequence(XMLType_instance XMLType)
```

Purpose

XMLSEQUENCE takes an XMLType instance as input and returns a varray of the top-level nodes in the XMLType. You can use this function in a TABLE clause to unnest the collection values into multiple rows, which can in turn be further processed in the SQL query.

Example

The following example shows how XMLSequence divides up an XML document with multiple elements into VARRAY single-element documents. The TABLE keyword instructs the Oracle Database to consider the collection a table value that can be used in the FROM clause of the subquery.

```
SELECT EXTRACT(warehouse_spec, '/Warehouse') as "Warehouse"
   FROM warehouses
   WHERE warehouse_name = 'San Francisco';

Warehouse
------------------------------------------------------------
<Warehouse?
  <Building>Rented</Building>
  <Area>50000</Area>
  <Docks>1</Docks>
  <DockType>Side load</DockType>
  <WaterAccess>Y</WaterAccess>
  <RailAccess>N</RailAccess>
  <Parking>Lot</Parking>
  <VClearance>12 ft</VClearance>
</Warehouse>


SELECT VALUE(p)
   FROM warehouses w,
   TABLE(XMLSEQUENCE(EXTRACT(warehouse_spec,'/Warehouse/*'))) p
   WHERE w.warehouse_name = 'San Francisco';

VALUE(p)
------------------------------------------------------------

<Building>Rented</Building>
<Area>50000</Area>
<Docks>1</Docks>
<DockType>Side load</DockType>
<WaterAccess>Y</WaterAccess>
<RailAccess>N</RailAccess>
<Parking>Lot</Parking>
<VClearance>12 ft</VClearance>
```

XMLTRANSFORM

Syntax

```
xmltransform::=XMLTRANSFORM(XMLType_instance,XMLType_instance)
```

Purpose

XMLTRANSFORM takes as arguments an XMLType instance and an XSL style sheet, which is itself a form of XMLType instance. It applies the style sheet to the instance and returns an XMLType. This function is useful for organizing data according to a style sheet as you are retrieving it from the database.

Example

The XMLTRANSFORM function requires the existence of an XSL style sheet. Here is an example of a very simple style sheet that alphabetizes elements within a node:

```
CREATE TABLE xsl_tab (col1 XMLTYPE);

INSERT INTO xsl_tab VALUES (
   XMLTYPE.createxml(
   '<?xml version="1.0"?>
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 >
      <xsl:output encoding="utf-8"/>
      <!-- alphabetizes an xml tree -->
      <xsl:template match="*">
        <xsl:copy>
          <xsl:apply-templates select="*|text()">
            <xsl:sort select="name(.)" data-type="text" order="ascending"/>
          </xsl:apply-templates>
        </xsl:copy>
      </xsl:template>
      <xsl:template match="text()">
        <xsl:value-of select="normalize-space(.)"/>
      </xsl:template>
    </xsl:stylesheet>  '));
```

The next example uses the xsl_tab XSL style sheet to alphabetize the elements in one warehouse_spec of the sample table oe.warehouses:

```
SELECT XMLTRANSFORM(w.warehouse_spec, x.col1).GetClobVal()
   FROM warehouses w, xsl_tab x
   WHERE w.warehouse_name = 'San Francisco';

XMLTRANSFORM(W.WAREHOUSE_SPEC,X.COL1).GETCLOBVAL()
--------------------------------------------------------------------------------
<Warehouse>
  <Area>50000</Area>
  <Building>Rented</Building>
  <DockType>Side load</DockType>
  <Docks>1</Docks>
  <Parking>Lot</Parking>
  <RailAccess>N</RailAccess>
  <VClearance>12 ft</VClearance>
  <WaterAccess>Y</WaterAccess>
</Warehouse>
```

Part V

Deployment and Execution Reference

This part contains the following chapters:

28

Scheduling ETL Objects

This chapter contains the following topics:

Editing a Schedule

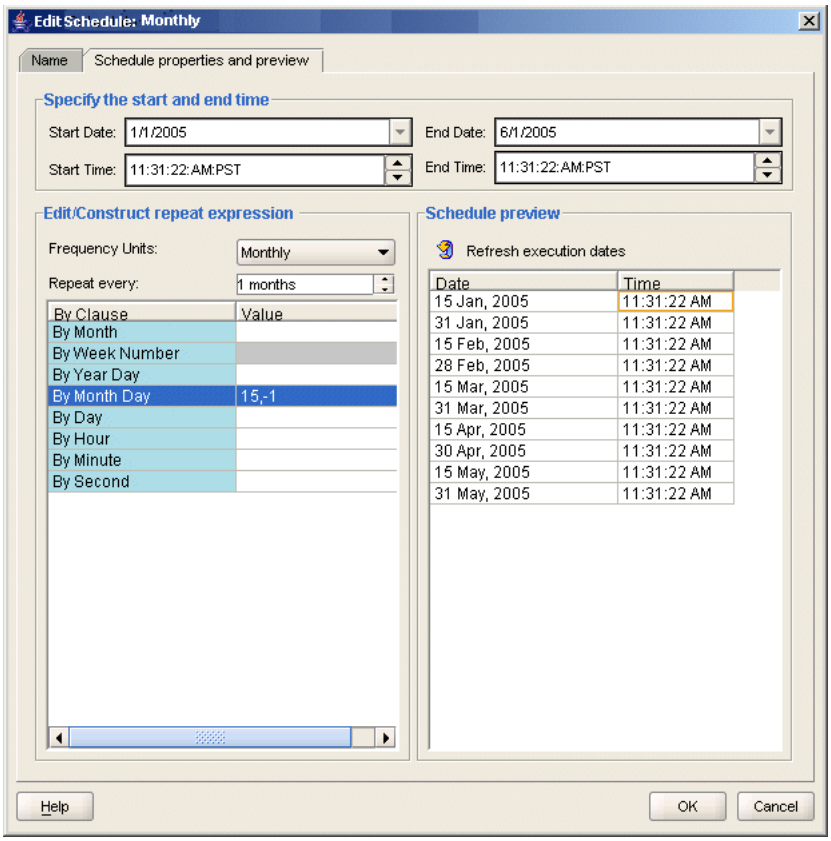
Figure 28–1 shows the schedule editor with the Start and End Dates and Times at the top of the editor.

The repeat expression appears in the lower left panel. Use the repeat expression to specify the Frequency Unit, Repeat Every, and one or more By Clauses.

The schedule preview appears in the lower right panel. The preview refreshes each time you press the <Enter> key or navigate to a new cell on the schedule editor. If you specify an invalid schedule, the preview displays an error message.

For examples of schedules you can define in Warehouse Builder, see Example Schedules on page 28-5.

Figure 28–1 The Schedule Editor



Start and End Dates and Times

The start and end dates and times define the duration for which the schedule is valid.

Begin by specifying the time zone. You can accept the default start date or specify a time in the future. Be sure to change the default end date which is the same as the default start date.

When working in the wizard, click **Next** to view the next page.

When working the Schedule Editor, the start date and time become the defaults for the By Clauses in the Repeat Expression. Notice in Figure 28–1 that the execution time in Schedule Preview corresponds to the Start Time.

Defining Schedules To Repeat

The repeat expression determines how often Warehouse Builder executes the schedule. Define the repeat expression by specifying the Frequency Unit, the Repeat Every value, and one or more By Clauses values.

When working in the wizard, note that ByClauses are not available. After you complete the wizard, you can open the schedule and set the ByClauses using the schedule editor.

Frequency Unit

The Frequency Unit determines the type of recurrence. The possible values are YEARLY, MONTHLY, WEEKLY, DAILY, HOURLY, MINUTELY, and SECONDLY.

Also, you can define schedules to run One Time or Immediately.

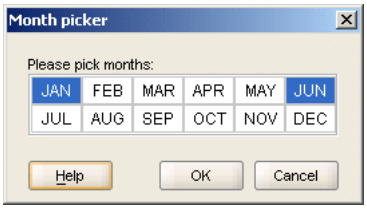
Repeat Every

The Repeat Every value specifies how often the recurrence repeats. The default value is 1 and the maximum value is 999. If you select YEARLY for the Frequency Unit and leave the Repeat Every value at 1, Warehouse Builder evaluates the schedule for every year included in the date range you specify in Start and End Dates and Times. For the same schedule, if you change Repeat Every to 2, Warehouse Builder evaluates the schedule only every other year within the specified date range.

By Clauses

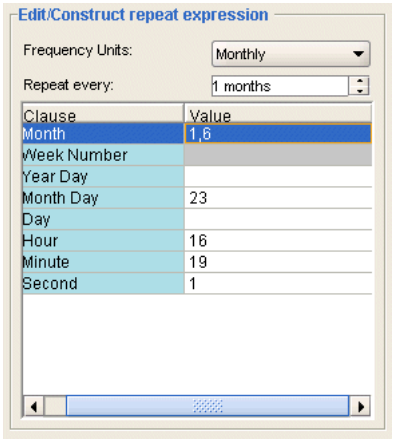
By Clauses enable you to define repeat expressions for complex schedules such as a schedule to run the first Friday of any month that contains 5 weeks. For each clause you can either type in values or click the Ellipsis button to view a selector dialog such as shown in Figure 28–2. If your goal is to know how to quickly type in values, first use the selector dialogs to learn what values are valid and also refer to Example Schedules on page 28-5.

Figure 28–2 Selector Dialog for Picking Months in a Year



When you use the selector dialogs and select **OK,** Warehouse Builder displays the results in the schedule editor as shown in Figure 28–3. In this way, you can use the selector dialogs to learn what values are valid.

Figure 28–3 Month Clause for January and June



You can define the following by clauses:

- By Month
- By Week Number
- By Year Day
- By Month Day
- By Day

- By Hour
- By Minute
- By Second

By Month

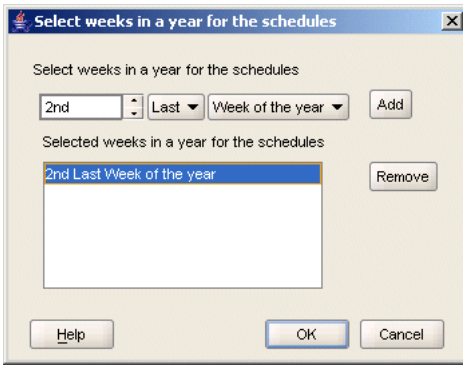
This specifies in which month or months the schedule is valid. If you type in values, use numbers such as 1 for January and 3 for March, or use three-letter abbreviations such as FEB for February and JUL for July.

By Week Number

Only when you select Yearly for the Frequency Unit can you schedule by the week number in the year.

You can either type in values or click the Ellipsis button to view the selector dialog. If you type in values, valid values include positive and negative integers from 1 to 52 or 53, depending on the year. For example, to set a schedule to run on the second to last week of the year, you can either type -2 or fill in the selector dialog as shown in Figure 28–2.

Figure 28–4 By Week Number Clause Set to Second To Last Week of the Year



The By Week Number clause follows the ISO-8601 standard, which defines the week as starting with Monday and ending with Sunday. Also, the first week of a year is defined as the week containing the first Thursday of the Gregorian year and containing January 4th.

Using this standard, a calendar year can have 52 or 53 weeks. Part of week 1 may be in the previous calendar year. Part of week 52 may be in the following calendar year. If a year has a week 53, part of it must be in the following calendar year.

As an example, in the year 1998, week 1 began on Monday December 29th, 1997. The last, week 53, ended on Sunday January 3rd, 1999. Therefore, December 29th, 1997, is in the first week of 1998 and January 1st, 1999, is in the 53rd week of 1998.

By Year Day

Use this clause to specify the day of the year as a number. A value of 1 equates to January 1st and 35 is February 4th. Valid values are and 1 to 366 and -366 to -1.

The negative values are useful for identifying the same dates year after year despite the occurrence of leap years. For example, the 60th day of the year is March 1st except for leap years when it is February 29th. To calculate the appropriate negative value, count backwards from the last day of the year. Therefore, the By Year Day for

December 31st is -1. December 30th is -2. To define a schedule for every March 1st, despite leap years, set By Year Day to -306.

By Month Day

This clause specifies the day of the month as a number. Valid values are 1 to 31 and -1 to -31. An example is 10, which means the 10th day of the selected month. Use the minus sign (-) to count backward from the last day. For example, if you set the By Month Day clause to -1, the schedule runs on the last day of every month. A value of -2 runs the schedule on the next to last day of every month.

By Day

This specifies the day of the week from Monday to Sunday in the form MON, TUE, and so on.

You can prefix the By Day values with positive and negative numbers. The numerical prefix you can use depends on the value you select for the Frequency Unit.

If you select Yearly as the frequency, you can prefix By Day with values that represent the weeks in a year, 1 to 53 and -53 to -1. Therefore, By Day set to 26Fri equates to the 26th Friday of the year. An entry of -1Mon when the frequency equals Yearly, equates to the last Monday of the year.

If you select Monthly as the frequency, you can prefix By Day with values that represent the weeks in a month, 1 to 5 and -5 to -1. In this case, an entry of -1Mon with frequency set to Monthly results in the last Monday of every month.

By Hour

This clause enables you to schedule by the hour. Valid values are 0 to 23 where 0 is midnight, 5 is 5 am, 13 is 1 pm, and 23 is 11 pm.

By Minute

Use this clause to schedule by the minute. Valid values are 0 to 59. As an example, 45 means 45 minutes past the hour.

By Second

Use this clause to schedule by the second. Valid values are 0 to 59. As an example, 30 means 30 seconds past the minute.

By Set Position

If your Oracle Database is version 10g Release 2 or higher, you can use this clause to schedule based on the position of items in a previously evaluated list of timestamps. Use other by clauses to return a list of timestamps. Then add the By Set Position clause to select one or more items from that list. It is useful for requirements such as running a job on the last workday of the month.

Valid values are 1 through 9999. A negative number selects an item from the end of the list (-1 is the last item, -2 is the next to last item, and so on) and a positive number selects from the front of the list. This clause is always evaluated last and only once in each frequency. The supported frequencies are MONTHLY and YEARLY.

Example Schedules

Use Table 28–1 as a guide for defining schedules.

Table 28–1 Example Repeat Expressions for Schedules

Schedule Description	Frequency Units	Repeat Every	By Clause
Every Friday.	weekly	1 week	By Day = FRI
Every other Friday.	weekly	2 weeks	By Day = FRI
Last day of every month.	monthly	1 month	By Month Day = -1
Second-to-last day of every month.	monthly	1 month	By Month Day = -2
First Friday of any month containing 5 weeks.	monthly	1 month	By Day = -5FRI
Last workday of every month	monthly	1 month	By Day =MON,TUE,WED,THU,FRI; By Set Pos=-1
On March 10th.	yearly	1 year	By Month = MAR By Month Day = 10
Every 12 days.	daily	12 days	n/a
Every day at 8 am and 5 pm.	daily	1 day	By Hour = 8,17
On the second Wednesday of every month.	monthly	1 month	By Day = 2 WED
Every hour for the first three days of every month.	hourly	1 hour	By Month Day = 1,2,3

29

Deploying Target Systems

After you design and configure the logical definitions of your target system, you can deploy and create the physical instance of your target. You can then start deployed mapping and process flow scripts to load or update your data. If you are working with a previously deployed system, you can view deployment history and plan an upgrade.

This chapter contains the following topics:

Creating Target Locations

You can deploy to locations that meet these criteria:

- The location type is appropriate for deployment, as described in "Selecting a Target Location Type".
- The location owner is registered with Warehouse Builder.

Warehouse Builder must be installed on the same system as the target location. If you are deploying to a remote location, then Warehouse Builder must be installed there as well as on the local system.

See Also: *Oracle Warehouse Builder Installation and Configuration Guide*

Selecting a Target Location Type

You can deploy to several different types of locations. Each location type has a different use:

- **Databases**: Targets for either relational or dimensional business intelligence systems, including objects such as tables and views, or dimensions and cubes.
- **Files**: Targets for storing data in comma-delimited or XML format.
- **Process Flows and Schedules**: Targets for managing ETL.

- **Business Intelligence**: Targets for metadata derived from Databases/Oracle modules.

See Also: "About Locations" on page 2-7

Granting Privileges to the Target Location

Some deployments require the owner of the target location to have more powerful privileges than Warehouse Builder grants when creating a new user:

- Upgrade action
- EUL deployment

A privileged database user can grant the additional privileges.

For ETL, the owner of the target location must have sufficient privileges to read data from the source location. If the source location is a database table, for example, the target must have `SELECT` privileges on the table.

Upgrade Action

The `GRANT_UPGRADE_PRIVILEGES` PL/SQL script grants the target user the necessary roles and privileges for the Upgrade action. Use this syntax:

```
@%OWB_ORACLE_HOME%/owb/rtp/sql/grant_upgrade_privileges username
```

Where:

`OWB_ORACLE_HOME` is the Oracle home directory for Warehouse Builder on the target system.

`username` is the owner of the target location.

For example, the following command grants privileges on a Windows system to the `SALES_TARGET` user.

```
@%OWB_ORACLE_HOME%\owb\rtp\sql\grant_upgrade_privileges sales_target
```

EUL Deployment

Discoverer locations require the EUL user to have the `CREATE DATABASE LINK` privilege.

Registering Locations

During the design process, you create logical definitions of locations. All modules, including their source and target objects, must have locations associated with them before they can be deployed.

Registering a location establishes a link between the Warehouse Builder repository and the locations of source data and deployed objects. You can change the definition of a location before it is registered, but not afterward. After the location is registered, you can only change the password. To further edit a location or one of its connectors, you must first unregister the location. Unregistering deletes the deployment history for the location.

Locations are registered automatically by deployment. Alternatively, you can explicitly register a location in the Control Center.

To register a location:

1. Open the Control Center Manager and select a location from the navigation tree.

2. From the File menu, choose **Register.**

   The Location dialog box appears.

3. Check the location details carefully.

   Click **Help** for additional information.

4. Click **OK**.

To unregister a location:

1. Open the Control Center Manager and select a location from the navigation tree.
2. From the File menu, choose **Unregister**.
3. Click **OK** to confirm the action.

Deploying From the Design Center Project Explorer

Deploying objects from the Design Center navigation tree is a simple, quick process of deployment. Use this method when you want Warehouse Builder to use the default deployment actions.

The default deployment action is determined by changes to the object design since the object was last deployed. For example, when you deploy an altered table, the default action is Upgrade. If the object has not changed or it does not have a default action, then it will not be redeployed.

Note: Progress messages appear at the bottom of the Design Center window. If you want to view detailed information, set the deployment preferences first.

From the Tools menu, click **Preferences**. The settings that control deployment output are listed under Deployment Process. Click **Help** for descriptions of the settings.

To deploy objects from the Design Center:

1. Select a deployable object from the Project Explorer.
2. From the Design menu, select **Deploy**.
3. To monitor the progress of the deployment, select **Job Monitor** from the Tools menu.

You can also deploy by right-clicking the object or using the Deploy toolbar icon.

Opening Control Center Manager

Control Center Manager provides access to the Control Center for the active configuration of a project. After opening Control Center Manager, you can access the deployable objects and the deployment jobs in your current project.

To open Control Center Manager:

1. From the Tools menu, select **Control Center Manager.**

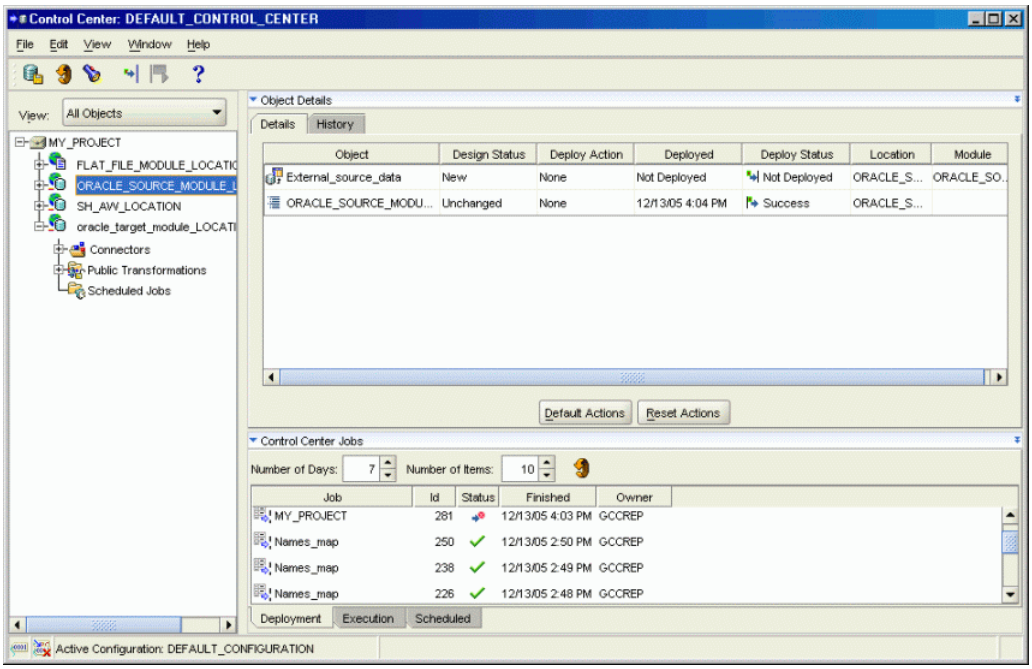
   The Connection Information dialog box appears if some connection details are missing. The default Control Center typically does not require a separate login.

2. Provide the password and any other missing information from the dialog box.

Click **Help** for more information.

Control Center Manager opens, as shown in Figure 29–1. For more information about the menus, toolbar, and panels, choose **Topic** from the Help menu.

Figure 29–1 Main Window in Control Center Manager



Deploying From Control Center Manager

Control Center Manager provides complete information about the current deployment status and history of objects in the current project. It enables you to override the default deployment options and monitor the progress of deployment jobs.

You can validate and generate objects from the Design Center, or Control Center Manager will validate and generate the objects during deployment.

Note: Numerous settings on the Preferences dialog box control the behavior of Control Center Manager. Additional settings control deployment.

From the Tools menu, click **Preferences**. The settings are listed under Control Center Monitor and Deployment. Click **Help** for descriptions of the settings.

To deploy objects from Control Center Manager:

1. Open Control Center Manager.
2. Expand the project tree to display the contents. You can choose a filter to display a partial list.
3. Select the objects that you want to deploy.

   You can select one or more individual objects, modules, or locations.

   The details about the selected objects appear in the Details tab.

4. Choose a deployment action for each object. You can choose the action either automatically or manually:

   - On the Details tab, click **Default Actions**.

     The default action depends on the type of object and whether the object has been deployed previously. For example, dimensions and schedules do not support the Upgrade action. You can use the Changed Objects filter on the deployment tree to find objects without an action.

   - On the Details tab, click the Deploy Action field, and select an action from the popup menu.

   The action appears in the Deploy Action column.

5. Click the Deploy icon on the toolbar.

Warehouse Builder must be open during generation of the job, but not during deployment. The Control Center Manager can be closed at any stage.

Reviewing the Deployment Results

You can monitor the progress of a job by watching the status messages at the bottom of the window and the Status column of the Control Center Jobs panel.

When the job is complete, the new deployment status of the object appears in the Details tab. You can review the results and view the scripts.

To view deployment details:

Double-click the job in the Job Details panel.

The Deployment Results window will appear. For a description of this window, choose **Topic** from the Help menu.

To view deployed scripts:

1. Open the Deployment Results window, as described in the previous steps.
2. Select the object in the navigation tree.
3. On the Script tab, select a script and click **View Code**, or just double-click the script name.

Deploying to Additional Locations

A repository may contain numerous target locations. You can deploy an object to only one target location at a time. However, you can deploy different modules (or entire projects) to different locations, or you can deploy the same modules to multiple locations. The procedures vary depending on whether the objects have already been deployed.

To deploy a different module to a different schema on the same system:

1. Create a target location for the new schema.
2. Create or edit the module, and associate it with the new location.
3. Use the same Control Center as the previous deployment.
4. Deploy as usual.

To deploy a module to a second schema on the same system:

Note: You do not need to create a new configuration to deploy to a second schema, but it will be more convenient if you are changing additional configuration parameters.

1. Create a location for the schema.
2. In the Project Explorer, double-click the module.

   The Edit Module dialog box opens.
3. On the Data Locations tab, move the new location from Available Locations to Selected Locations, then click **OK**.
4. In the Project Explorer, select the module and click the Configure icon.

   The Configuration Properties dialog box opens.
5. Under Identification, select the new location from the list for the Location property, then click **OK**.
6. Deploy as usual.

To deploy a module to a second schema on a different system:

1. Set up the target system:
   a. Install Warehouse Builder on the target schema.
   b. Use the Repository Assistant to create a repository and a user for the target schema.
2. In the local Design Center, create a new configuration and a new Control Center for the repository on the target system. Make this the default configuration.
3. Define a target location for the remote schema.
4. Deploy as usual.

Starting the ETL Process

ETL is the process of extracting data from the source locations, transforming it as specified in the mappings, and loading it into objects in the target location. In Warehouse Builder, you start ETL by starting a mapping or a process flow.

Starting a mapping or a process flow involves the following steps:

1. Generating the PL/SQL, SQL*Loader, or ABAP script, as needed.
2. Deploying the script, as needed.
3. Executing the script.

You can start the Extract, Transform, and Load process manually in Warehouse Builder in the following ways:

To start ETL in the Project Explorer in the Design Center:

1. Select a mapping or a process flow from the Project Explorer.
2. From the Design menu, select **Start**.

   You can also right-click the object and select **Start** from the pop-up menu.

To start ETL in Control Center Manager:

1. Open Control Center Manager and select a location from the navigation tree.
2. Select a mapping or process flow from the navigation tree.

   Note: Only one mapping or process flow can be started at a time.

3. From the File menu select **Start**.

   You can also right-click the object and select **Start** from the pop-up menu, or click the Start icon in the toolbar.

Scheduling ETL Jobs

You can use any of the following methods to schedule ETL:

- Use the scheduler in Warehouse Builder.

  Refer to Chapter 28, "Scheduling ETL Objects".

- Use a third-party scheduling tool.
- Submit the job to the Oracle job queue using the scheduling facilities in Oracle Enterprise Manager.

Runtime Preferences

This describes some of the Platform Properties you may wish to alter. Other properties can be altered by using the file <owb home>/owb/bin/admin/Runtime.properties.

property.RuntimePlatform.0.NativeExecution.SQLLoader.oracle_home_10g

This optional property defines the location of the Oracle10*g* home from which SQL*Loader will be executed. This property is selected when the version of target location starts with "10.". If this property is not set then the "oracle_home_default" property is used instead.

property.RuntimePlatform.0.NativeExecution.SQLLoader.oracle_home_9i

This optional property defines the location of the Oracle9*i* home from which SQL*Loader will be executed. This property is selected when the version of target location starts with "9.". If this property is not set then the "oracle_home_default" property is used instead.

property.RuntimePlatform.0.NativeExecution.SQLLoader.oracle_home_8i

This optional property defines the location of the Oracle8*i* home from which SQL*Loader will be executed. This property is selected when the version of target location starts with "8.". If this property is not set then the "oracle_home_default" property is used instead.

property.RuntimePlatform.0.NativeExecution.SQLLoader.oracle_home_default

This optional property defines the location of the default Oracle home from which SQL*Loader will be executed. This property is selected when no other explicit property is set. If this property is not set then the ORACLE_HOME environment variable of the Control Center is used, otherwise the Control Center's own home is used.

property.RuntimePlatform.0.NativeExecution.SQLLoader.sqlldr_exe_10g

This optional property defines the location or name of the SQL*Loader executable. This property is selected when the version of target location starts with "10.". If this property is not set then the "sqlldr_exe_default" property is used instead.

The value can specify either the executable terminal name or a full path to the executable. A full path name will be constructed from the specified Oracle home, the "bin" directory and the terminal name.

property.RuntimePlatform.0.NativeExecution.SQLLoader.sqlldr_exe_9i

This optional property defines the location or name of the SQL*Loader executable. This property is selected when the version of target location starts with "9.". If this property is not set then the "sqlldr_exe_default" property is used instead.

The value can specify either the executable terminal name or a full path to the executable. A full path name will be constructed from the specified Oracle home, the "bin" directory and the terminal name.

property.RuntimePlatform.0.NativeExecution.SQLLoader.sqlldr_exe_8i

This optional property defines the location or name of the SQL*Loader executable. This property is selected when the version of target location starts with "8.". If this property is not set then the "sqlldr_exe_default" property is used instead.

The value can specify either the executable terminal name or a full path to the executable. A full path name will be constructed from the specified Oracle home, the "bin" directory and the terminal name.

property.RuntimePlatform.0.NativeExecution.SQLLoader.sqlldr_exe_default

This optional property defines the location or name of the SQL*Loader executable. This property is selected when no other explicit property is set. If this property is not set then the standard terminal name for the execution platform is used, for example, "sqlldr" on UNIX and "SQLLDR.EXE" on Windows.

The value can specify either the executable terminal name or a full path to the executable. A full path name will be constructed from the specified Oracle home, the "bin" directory and the terminal name.

property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_10g

This optional property defines the location of the Oracle10*g* home from which SQL*Plus will be executed. This property is selected when the version of target location starts with "10.". If this property is not set then the "oracle_home_default" property is used instead.

property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_8i

This optional property defines the location of the Oracle8*i* home from which SQL*Plus will be executed. This property is selected when the version of target location starts with "8.". If this property is not set then the "oracle_home_default" property is used instead.

property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_9i

This optional property defines the location of the Oracle9*i* home from which SQL*Plus will be executed. This property is selected when the version of target location starts with "9.". If this property is not set then the "oracle_home_default" property is used instead.

property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_default

This optional property defines the location of the default Oracle home from which SQL*Plus will be executed. This property is selected when no other explicit property is set. If this property is not set then the ORACLE_HOME environment variable of the Control Center is used, otherwise the Control Center's own home is used.

property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_10g

This optional property defines the location or name of the SQL*Plus executable. This property is selected when the version of target location starts with "10.". If this property is not set then the "sqlldr_exe_default" property is used instead.

The value can specify either the executable terminal name or a full path to the executable. A full path name will be constructed from the specified Oracle home, the "bin" directory and the terminal name.

property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_9i

This optional property defines the location or name of the SQL*Plus executable. This property is selected when the version of target location starts with "9.". If this property is not set then the "sqlldr_exe_default" property is used instead.

The value can specify either the executable terminal name or a full path to the executable. A full path name will be constructed from the specified Oracle home, the "bin" directory and the terminal name.

property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_8i

This optional property defines the location or name of the SQL*Plus executable. This property is selected when the version of target location starts with "8.". If this property is not set then the "sqlldr_exe_default" property is used instead.

The value can specify either the executable terminal name or a full path to the executable. A full path name will be constructed from the specified Oracle home, the "bin" directory and the terminal name.

property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_default

This optional property defines the location or name of the SQL*Plus executable. This property is selected when no other explicit property is set. If this property is not set then the standard terminal name for the execution platform is used, for example, "sqlplus" on UNIX and "SQLPLUS.EXE" on Windows.

The value can specify either the executable terminal name or a full path to the executable. A full path name will be constructed from the specified Oracle home, the "bin" directory and the terminal name.

property.RuntimePlatform.0.delete_all_job_on_complete

This Boolean property can be used to delete the audit-record(s) of all jobs that complete successfully. Its default value is false. If set to true, then the audit-execution record and any map-audit records will be automatically deleted if the job completes successfully.

property.RuntimePlatform.0.logfile_max_size

Can be given an integer number which represents the maximum number of trace messages that will be logged into the Control Center Service logfile. If this property is set then a new log-file will be created when this value is reached. The value must be at least 1000. The default value is 0 and this means that there is no maximum size and a new logfile will only be created when the Control Center Service is started.

property.RuntimePlatform.0.max_number_logfiles

This integral property represents the number of logfiles that will be automatically retained in the <owb-home>/log subdirectory. This property must be at least 2 and is 10 by default. When a new logfile is created, the system will ensure that at most this number of logfiles will be retained in the log subdirectory.

property.RuntimePlatform.0.purge_delay_hours

The audit details of any Deployment or Execution with start time older than the specified number of hours may be purged, regardless of job status. If not supplied, the default is 24 hours. The minimum value is 1 hour.

property.RuntimePlatform.0.purge_minimum_minutes

No audit details of any Deployment or Execution with start time under the specified number of minutes old may be purged, regardless of job status. If not supplied, the default is 10 minutes. The minimum value is 1 minute.

property.RuntimePlatform.0.recovery

This Boolean property can be used to control whether deployments and executions are recovered when the Service starts up. The default value is true. This means that any deployments/executions that are still busy will be restarted when the service starts up. A value of false will mean that such deployments/executions are not recovered and are deactivated.

property.RuntimePlatform.0.platform_net_host_name

This property defines the host name that should be used when making inward connections to the Control Center. On RAC this would normally identify the entire cluster, not a node within the cluster. The property is used for Process Flows and Schedules where a database link is required to connect back to the Control Center schema.

If this property is not set then the host name of the node running the Control Center service is used.

property.RuntimePlatform.0.platform_net_service_name

This property defines the TNS name that should be used when making inward database connections to the Control Center. The property is used for Process Flows and Schedules where a database link is required to connect back to the Control Center schema.

If this property is not set then the Control Center service connection information is used.

30

Auditing Deployments and Executions

Auditing deployment and execution information can provide valuable insight into how your target is being loaded and what you can do to further optimize mapping and process flow settings and parameters. Also it can give you immediate feedback on deployment information to make you aware of the deployment history of an object. Auditing and deployment information is available to you using Oracle Warehouse Builder Repository Browser.

This section provides information:

About the Repository Browser

The Oracle Warehouse Builder Repository Browser is a browser-based tool that generates reports data stored in Oracle Warehouse Builder repositories. Using the Repository Browser, you can view:

- Detailed information about the design of a repository. Reports are generated from data stored in the Warehouse Builder repositories.
- Reports that provide access to both high-level and detailed ETL runtime information. This information includes the following:
  - Timings for each mapping and process flows
  - Details of activities for each process flow
  - Error details
  - Deployment information to manage separate target environments

As an alternative to using the Repository Browser, you can access the same information through the Warehouse Builder public views. Launch a SQL*Plus session and query the public views. Refer to the Oracle Warehouse Builder API and Scripting Reference for a list of public views.

Viewing Audit Reports

Audit reports provide information about deployment and ETL jobs. Each time you deploy an object or start a job, the details are stored in the repository. You can access this information in these environments:

- Control Center Manager
- Repository Browser

The Repository Browser provides the information in the form of predefined reports. The reports are displayed in your default Web browser. The Repository Browser Listener must be running.

To use the Repository Browser:

1. From the Design Center Tools menu, choose **Repository Browser**.
2. On the connection page, log in with your Warehouse Builder user name and password.
3. Choose **Control Center Reports.**

You can also open the browser when the Design Center is closed.

Opening the Repository Browser

Opening the Repository Browser is a multistep process:

1. Before you can open the Repository Browser, the Repository Browser Listener must be started as described in Starting the Repository Browser Listener on page 30-2.
2. When the Repository Browser Listener is running, you can start the Repository Browser in a number of ways as described in Starting the Repository Browser on page 30-3.
3. The Repository Browser opens to the Login page where you log in to a repository as described in Logging in to a Repository on page 30-3.

> **Note:** In order to open the Repository Browser, you must have the ACCESS_PUBLICVIEW_BROWSER system privilege. You automatically have this privilege when you are the owner of the repository you want to browse. When you are not the owner of the repository, contact your database administrator who can give you this privilege.

Starting and Stopping the Repository Browser Listener

Before you can open the Repository Browser, the Repository Browser Listener must be started.

Starting the Repository Browser Listener

From the **Start** menu, select **Programs,** then the name of the Oracle home directory into which you installed Warehouse Builder**,** then **Warehouse Builder,** then **Administration,** and then **Start OWB Browser Listener.**

Stopping the Repository Browser Listener

From the **Start** menu, select **Programs,** then the name of the Oracle home directory into which you installed Warehouse Builder**,** then **Warehouse Builder,** then **Administration,** and then **Stop OWB Browser Listener.**

Starting the Repository Browser

Before you can start the Repository Browser, the Repository Browser Listener must be started as described in Starting the Repository Browser Listener on page 30-2. Once the Listener is running, you can start the Repository Browser in any one of the following ways:

- From the **Start** menu, select **Programs,** then the name of the Oracle home directory into which you installed Warehouse Builder**,** then **Warehouse Builder,** and then **Repository Browser.**
- From the menu of the Design Center of the Warehouse Builder, select **Tools**, and then **Repository Browser**.
- From within any web browser, type in the location of the Repository Connection page.

  You can identify this location, by saving it after starting the Repository Browser using another method of starting the Repository Browser, or you can ask your Oracle Builder Administrator for the location.

Regardless of which approach you take, once you start the Repository Browser, the browser opens the Repository Connection page from which you log in to the Repository Browser.

Logging in to a Repository

To login to a repository, specify the connection information for the repository you would like to access. If you do not know the connection information, contact your Warehouse Builder database administrator.

Specify the first Repository Browser page that you want to open:

- When you want start by browsing through the Repository using the **Repository Navigator** page, select **Design Repository**.
- When you want to start by viewing Repository Browser Control Center reports using the Control Center **Reports** page, select Control Center.

The Design Center

When you start the Repository Browser select Design Center as described in Starting the Repository Browser, the Repository Browser opens to the Repository Navigator page. From this page you can discover information about the design of Oracle Warehouse Builder Repositories such as:

- Object Reports
- Object Properties
- Object Lineage
- Object Impact Analysis

Repository Navigator

You use the Repository Navigator page to search the metadata of a repository and to access metadata main properties, lineage, impact analysis, and list of related reports and Control Center reports for the repository.

Search

Search by object type, or name, or by both.

- To search by object type, select the type of object for which you want to search in the **Search By Type** list. The search result is a list of all objects of this type.
- To search by name, type the name of the object in the Search field. You can search for just the first character(s) of the name in which case, the search result is a list of objects whose names begin with those characters.

Click **Go** to start your search.

The Repository Browser displays the results of the search in a new page called the **Repository Search Results** page. You can also search for the new objects from a Repository Search Results page.

All

A navigator tree for the repository.

The use of the columns is described in the following table.

Column Head	Description:
Focus	Click an icon in this column to change the focus of the tree.
Name	The name of an item in the tree. Click the plus sign (+) or minus sin (-) next to an item in the tree to expand or collapse the tree. Click the Name of the object to open the Object Properties page for that object.
Report	Click an icon in this column to open the Object Reports page for the related item.
Lineage	Click an icon in this column to open the Object Lineage page for the related item.
Impact	Click an icon in this column to open the Object Impact Analysis page for the related item.

Refresh

Click to refresh your view of the repository.

The tree collapses when you refresh the data. If you had navigated or focused to a specific area before refreshing, you need to navigate or focus again to the desired node in the tree.

Related Links

Click **Control Center - Reports** to open the Control Center Reports page from which you can select a deployment, execution, or management report.

Object Reports

The Object Reports page provides access to the predefined Design Center reports for the object that you selected in the Repository Navigator. Use these reports to examine your metadata.

Click a Report name to display a report.

The following types of reports are available:

- Summary Reports
- Detailed Reports
- Implementation Reports
- Lineage and Impact Analysis Reports and Diagrams

Summary Reports

The type of information displayed in a summary report is determined by the object selected. For example, a Table Summary Report lists all tables in the module. A Materialized View Summary Report lists all materialized views in the module. Header information that identifies the module is also displayed. Selecting the name of an item displays the detailed report for that item.

Summary reports are available for the following objects:

Advanced Queue
Collection
Cube
Dimension
External Table
File
Function
Materialized View
Procedure
Sequence
Table Library
Table Function
Table
Transform Map
View

Detailed Reports

The type of information displayed in a detailed report is determined by the object selected. Detailed reports provide comprehensive information about an item. For example, a Detailed Table Report lists information about the table columns, keys, foreign keys, and physical configuration parameters.

Detailed reports include:

Detailed Advanced Queue Report
Detailed Collection Report
Detailed Connector Report
Detailed Cube Implementation Report
Detailed Dimension Implementation Report
Detailed Dimension Report
Detailed External Table Report
Detailed File Module Report
Detailed File Report

Detailed Function Library Report
Detailed Function Report
Detailed Installation Report
Detailed Location Report
Detailed Materialized View Report
Detailed Module Report
Detailed Object Report
Detailed Process Flow Activity Report
Detailed Process Flow Module Report
Detailed Process Flow Package Report
Detailed Process Flow Package Report
Detailed Process Flow Report
Detailed Process Transition Report
Detailed Project Report
Detailed Record Report
Detailed Repository Report
Detailed Runtime Location Report
Detailed Sequence Report
Detailed Table Function Report
Detailed Table Report
Detailed Transform Map Report
Detailed Transform Map Report
Detailed View Report

Implementation Reports

Implementation Reports can be run on Dimensions and Cubes. They provide information on how physical objects are used to implement logical objects.

Lineage and Impact Analysis Reports and Diagrams

Lineage and Impact Analysis Reports and Diagrams are available for Cubes, Dimensions, Materialized Views, Tables, Views, and Records.

Impact Analysis Reports

Impact Analysis Reports list all items belonging to the subject of the report. The name of the mapping and the name of the item that it is mapped to is also displayed. The report provides a one-step impact analysis for all items related to the selected item.

For example, if you want a list of all the columns in a table used as sources in any mappings, use this report.

Lineage Reports

Lineage Reports are similar to Impact Analysis Reports. They list items that are used as targets in a mapping.

Lineage and Impact Analysis Diagrams

A Lineage Diagram graphically displays all the objects and transformations that are used to make up the subject of the Diagram. Lineage can be performed at either the object level or the item level. At the Object Level, the diagram can contain Tables, Views, Materialized Views, Dimensions, Cubes, Records, and Operators. At the item level the diagram can contain Columns, Measures, Fields, Operator Parameters, and Level Attributes.

- The Lineage Diagram is displayed with the subject on the right side of the screen.

- An Impact Analysis Diagram is identical except it shows all objects and transformations that might be affected by a change to the subject. The subject is displayed on the left side of the screen.

Lineage and Impact Analysis diagrams are created based on a Dependency Index. In order for the data displayed in the diagram to be current, the index must be refreshed.

Object Properties

The Object Properties page displays the properties of the object that you selected in the Repository Navigator.

From this page you can go to the Object Reports, Object Lineage, or Object Impact Analysis pages by clicking on the corresponding link on the left side of the page.

See also: Summary Reports on page 30-5 and Detailed Reports on page 30-5

Object Lineage

The Object Lineage page displays information about how object that you selected in the Repository Navigator is used.

From this page you can go to the Object Properties, Object Reports, or Object Impact Analysis pages by clicking on the corresponding link on the left side of the page.

See also: Lineage and Impact Analysis Reports and Diagrams on page 30-6

Object Impact Analysis

The Object Impact Analysis page displays the Impact Analysis diagram for the object that you selected in the Repository Navigator. The Impact Analysis diagram is a graphical representation of the objects on which the definition of the selected object depends. As such, it represents the potential impact of a change in the definition of the selected object.

From this page you can go to the Object Reports, Object Properties, or Object Lineage pages by clicking on the corresponding link on the left side of the page.

See also: Lineage and Impact Analysis Reports and Diagrams on page 30-6

Control Center Reports

The Control Center of the Repository Browser provides the following types of reports: Deployment Reports, Execution Reports, and Management Reports.

Note: You can access the Design Object Reports from any Control Center Reports by clicking the **Design Repository: Navigator** link on the report page.

Deployment Reports

Top-level deployment reports are:

- Deployment Schedule Reports that show basic Warehouse attributes and display a node-tree giving details of all deployments in time order.

- Object Summary Reports that show basic Warehouse attributes and list deployed objects (Processes, Maps and Data Objects) in type/name order with details of their latest deployment.
- Locations Reports that show all Locations into which objects have been deployed.

From these top-level deployment reports, you can accessDeployment Error Detail Reports and Deployment Reports that supply details about the deployment of a specific Process, Map, or Data Object.

Execution Reports

Top-level execution reports

- Execution Schedule Reports that show basic Warehouse attributes and display a node-tree giving details of all Process Runs (and top-level Map Runs) in time order.
- Execution Summary Reports that show basic Warehouse attributes and lists executed Processes (and top-level Maps) in type/name order.

From these top-level execution reports, you can access other reports that allow you to:

- Monitor jobs using Execution Reports (sometimes called Execution Detail Reports) that show the execution job details of a given Process Run or Map Run; Execution Job Reports that show details of logical errors for a given target detected during the execution of Map Runs; and Job Error Diagnostic Reports that show basic details of runtime errors and target details, and, when possible, list source and target column details.
- Display diagnostics using Error Table Execution Reports that show details of logical errors for a given target detected during the execution of Map Run; Trace Reports that show details of source and target values plus data errors detected during the execution of Map Runs; and Job File Reports that show basic Process or Map attributes, list log and data files associated with the Process or Map Run, and display the contents of the selected file.
- Rerun jobs using Job Start Reports that show Process or Map identification properties (including latest deployment and latest execution dates), list all execution parameters for the Process as specified by the latest deployment, and assign parameter default values from the latest deployment specification.

Management Reports

The main Repository Browser management report is the Service Node Report that displays and enables you to manage service node information for the RAC system.

Also, from a Locations Report (a top-level deployment report) you can access the Location Validation Report that shows basic Location attributes, current Control Center connection details, and current Location connection details.

Deployment Reports

Top-level deployment reports are Deployment Schedule Reports, Object Summary Reports, andLocations Reports. From these top-level deployment reports, you can accessDeployment Error Detail Reports and Deployment Reports that supply details about the deployment of a specific Process, Map, or Data Object.

Deployment Schedule Report

The Deployment Schedule report is a top-level Control Center report that shows basic Warehouse attributes and displays a node-tree giving details of all deployments in time order.

You use Deployment Schedule reports to view run details, and access Data Object, Map, and Process Deployment reports. With a Deployment Schedule Report, you can:

- Expand deployments to show run details.
- Filter deployments on date range.
- Set a date range for which you want to view deployments.
- Refresh report to show up-to-date deployment details.
- When you have sufficient privileges, you can purge selected deployment audit details.

Location Deployment Schedule Report

A Location Deployment Schedule Report is similar in appearance to a Deployment Schedule Report except that it only shows the deployments for a specific location and does not offer you the opportunity to purge audit details.

Deployment details report

A report with the columns described in the following table.

Column Name	Description
Select	Click to select this node in the deployment tree. This functionality is used in conjunction with the purge facility on a Deployment Schedule Report.
Focus	Click the icon in this column to change the focus of the tree to this node.
Name	A tree that represents all of the items in this deployment report. To expand a node, click its + (plus) icon. To collapse a node, click its - (minus) icon.
Dep	A number that identifies a deployment run.
Type	The type of item.
Obj Status	The status of the object.
Date	The date of deployment.
Dep Status	The status of the deployment.
Related Information	Other related information including a link to a related Deployment Error Detail Report, if appropriate.

Locations Report

This deployment report shows all Locations into which objects have been deployed.

Within this report, you can:

- Sort Locations on name and latest deployment time.
- When you have sufficient privileges, you can un-register selected Locations.

- When you have sufficient privileges and a link appears in the Validation column, you can open a related Location Validation Report in order to test and update connection details for a Location.

Object Summary Report

An Object Summary Report shows basic Warehouse attributes and lists deployed objects (Processes, Maps and Data Objects) in type/name order with details of their latest deployment.

Within this report, you can:

- Sort execution runs on name, type, location, latest deployment time, object status.
- Filter objects on type and status.

Location Object Summary Report

A Location Object Summary Report is similar to an Object Summary Report except that it also includes a Location parameters section. When you have sufficient privileges, you can update certain Web Server Base parameters, if applicable.

Deployment Report

This one help topic is displayed for Data Object Deployment Report, Map Deployment Report, and Process Deployment Report

This deployment report supplies details about the deployment of a specific Process, Map, or Data Object.

When the item is a Process, this report shows basic Process attributes and lists all deployments of the Process and its sub-Processes in time order. When the item is a Map, this report shows basic Map attributes and lists all deployments in time order. When the item is a Data Object, this report shows basic Data Object attributes and lists all deployments of the Data Object and its second-class Data Objects in time order.

Within this report you can:

- Sort deployments on deployment time.
- Filter deployments on deployment status.

Deployment Error Detail Report

Shows details of a specific deployment error and lists all of the messages for the deployment error.

Within this report, you can:

- Sort the error messages by message number.
- Filter the error messages by severity.

Execution Reports

The top-level execution reports are Execution Schedule Reports and Execution Summary Reports.

From these top-level execution reports, you can accessError Table Execution Reports, Job Error Diagnostic Reports, Trace Reports, Execution Job Reports, Job File Reports, Job Start Reports, andExecution Reports.

Execution Schedule Report

This execution report shows basic Warehouse attributes and displays a node-tree giving details of all Process Runs (and top-level Map Runs) in time order.

Within this report, you can:

- Focus on details for one Process Run.
- Expand Process Run to show activity run details.
- Filter Process Runs on execution name, execution status and date range (for example. to display only runs with 'busy' status).
- Use the calendar icon for date picker available to set start and end of date range.
- Refresh report to show up-to-date execution run details.
- When you have sufficient privileges, you can purge selected Process Run execution audit details.

Execution Summary Report

This execution report shows basic Warehouse attributes and lists executed Processes (and top-level Maps) in type/name order.

Within this report, you can:

- Sort execution runs on name, type, latest execution time, execution status.
- Filter Processes (and Maps) on type and execution status

Error Table Execution Report

This execution report shows details of logical errors for a given target detected during the execution of Map Runs.

Within this report, you can:

- Sort logical errors on map type, map name, execution start time, rule type, rule usage.
- Filter logical errors on map name, rule type and rule usage.
- When you have sufficient privileges, you can use the Purge Error Table to remove selected logical errors.

Trace Report

This execution report (also called the Map Run Trace Report) shows details of source and target values plus data errors detected during the execution of Map Runs.

Within this report, you can:

- Sort files on rowkey, table name.
- Filter diagnostic trace on execution severity and source or target.

> **Note:** Trace diagnostic are available one when the Map Run is executed with a particular setting of the Audit Level runtime parameter. Use this trace facility with care since it can generate a large volume of audit data.

Execution Job Report

An Execution Job Report shows detail information about the execution of either a Process Run or a Map Run.

Execution Job Report for a Process Run

When the Execution Job Report is for a Process Run, it shows basic Process Run execution details, lists execution parameters, lists activity (Map and sub-Process) details in time order, and lists error messages.

Within an Execution Job Report for a Process Run, you can:

- Hide or show activity details to show Map Run details.
- Refresh report to show up-to-date execution run details.
- Abort the Process Run.

Execution Job Report for a Map Run

When the Execution Job Report is for a Map Run, it shows basic Map Run execution details, including source and target Data Objects, lists execution parameters, lists map step details in time order, lists error messages, lists logical error details, and displays the contents of the SQL Loader log file (if applicable).

Within an Execution Job Report for a Map Run, you can:

- Hide or show map step details, including source and target Data Objects.
- Refresh report to show up-to-date execution run details.
- Sort logical errors on error table, map step, rule type, rule usage.
- Abort the Map Run.
- When your role has sufficient privileges, you can purge Error and Trace audit details for the Map Run and purge Error Table to remove selected logical errors

Job File Report

This execution report shows basic Process or Map attributes, lists log and data files associated with the Process or Map Run, and displays the contents of the selected file.

Within a Job File Report, you can:

- Sort files on file type, creation time.
- View the contents of any selected file.

Job Start Report

This execution report shows Process or executable Map identification properties, including latest deployment and latest execution dates, lists all execution parameters for the Process or executable Map as specified by the latest deployment, and assign parameter default values from the latest deployment specification.

Within a Job Start Report, you can:

- Sort execution parameters on name, category.
- Change values of any input parameter where permitted.
- Change the default Execution Name as necessary.
- Reset all parameter settings to their default values.
- Apply basic validation to parameter values.

- Start the Process or Map Run, which means it is scheduled for execution immediately.
- Navigate to the Deployment Report for latest deployment details of Process or Map.
- Navigate to the Execution Run Report for latest execution of current Process or Map.

Execution Report

An execution report (sometimes called an Execution Detail Report) shows all of the execution run details of a given Process, a given Map, or all of the Map Runs for which a given Data Object is a source or target.

When the Execution Report is for a Process, the report shows basic Process or Map attributes and lists the Process or Map Runs in time order.

Within an Execution Report, you can:

- Sort the Process or Map Runs on execution start time.
- Hide or show a Process or Map Run to show activity run details.
- Filter Process or Map Runs on execution status and execution severity.

Job Error Diagnostic Report

This execution report (sometimes also referred to as the Run Error Diagnostic Report) shows basic details of runtime error, shows target details, and lists source and target column details, where possible. Note that some column values are displayed only when your role has appropriate privilege

Within this report, you can sort column details on source/target category, source/target name, rowkey, column name.

Management Reports

The top-level management report is the Service Node Report. From this report you can open a Location Validation Report.

Service Node Report

This management report display and enables you to manage service node information for the RAC system. Specifically, it shows basic Warehouse attributes, lists details and status of all service nodes currently used in the RAC system, generated from the underlying system tables, lists service nodes available to the RAC system which are currently not in use, and shows the net service name to be used to access the runtime repository.

Within a Service Node Report, you can:

- Sort service nodes on instance number, instance name, runtime version.
- Update an instance number when the node is not enabled or active.
- Set or unset an enabled setting. (Note that you can never change an active setting as it is maintained by the RAC system.)
- Remove selected service nodes that are not enabled or active from being used by the RAC system.
- Add a node to the service, from the list of available nodes.

- Set the runtime repository net service name.
- Refresh report to show up-to-date service node details.

(Note that you can add, remove or update node details only if you have sufficient privilege.)

Location Validation Report

This management report is shows basic Location attributes, current Control Center connection details, and current Location connection details.

Within a Location Validation Report, you can:

- Test the Location connection
- Update Location connection details

Common Repository Browser Tasks

The following scenarios are examples of some typical actions performed by a user of the Repository Browser:

- Identifying Recently-Run Warehouse Builder Processes
- Identifying Why a Process Run Failed
- Comparing Process Runs
- Discovering Why a Map Run Gave Unexpected Results
- Identifying Recently-Made Deployments
- Identifying the Data Objects that are Deployed to a Specific Location
- Identifying the Data Objects that are Deployed to a Specific Location
- Identifying the Map Runs that Use a Specific Deployed Data Object
- Discovering the Default Deployment-Time Settings of a Deployed Process
- Rerunning a Process
- Monitoring a Process Run
- Aborting a Process Run
- Removing the Execution Audit Details for a Process
- Removing Old Deployment Audit details
- Unregistering a Location
- Updating Location Connection Details for a Changed Database Environment
- Updating Service Node Details in a Changing RAC Environment

Identifying Recently-Run Warehouse Builder Processes

1. Open the Execution Schedule Report to see the latest Processes runs.
2. Filter the information shown by using execution name, execution status and date range, as required.
3. Note any Process runs which are reported as having errors or not having completed.

4. Expand the tree structure for any Process run identified in Step 3 to see details of its activities (that is, any of its sub-processes and maps).

Identifying Why a Process Run Failed

1. Open the Execution Schedule Report and note the Process run which is marked as having errors.
2. Click the **Run Execution Report** link which opens a Execution Report that provides details of the Process run.
3. Note any Map runs which are reported as having errors or not having completed.
4. Click the **Run Execution Report** link which opens a Execution Report that provides details of any Map run identified in Step 3.
5. For any process-level or map-level error messages, click the **Run Error Diagnostic Report** link which opens a Job Error Diagnostic Report that displays more details of the error, including source data values.

Comparing Process Runs

1. Open the Execution Summary Report to see list of all Processes.
2. Click the Process name to see its Execution Report.
3. Compare the results of previous runs, using hide/show feature to reveal further details as required.
4. To see details of all of the activities of a Process, click the **Run Execution Report** link against any Process run which opens a Execution Report for that Process.

Discovering Why a Map Run Gave Unexpected Results

1. Open the Execution Schedule Report and note the Process Run which contains the required Map Run.
2. Click the **Run Execution Report** link which opens a Execution Report that provides details of the Process Run.
3. Click the **Run Execution Report** link which opens a Execution Report that provides details of the Map Run.
4. If the Map Run had `Audit Level` runtime parameter set to `Complete`, select the Trace tab link to see its Trace Report.
5. Filter trace lines by error and source or target as required, and note any unexpected source or target actions.
6. For error messages, click the **Run Error Diagnostic Report** link which opens a Job Error Diagnostic Report that displays more details of the error, including source data values.
7. Click **Purge Error** and **Trace Lines** to remove all details or errors and trace for this Map Run, if they are no longer required.

   If purging, a confirmation screen will be shown, requesting that the action be confirmed

Identifying Recently-Made Deployments

1. Open the Deployment Schedule Report to see the latest deployments.

2. Filter the information shown by using date range, as required.

   Note any deployments which are reported as having errors or not having completed.

3. Expand the tree structure for any deployment to see details of its components (that is, units and deployed objects).

4. For error messages, click the **Deployment Error Detail Report** link to display the related Deployment Error Detail Report.

Identifying the Data Objects that are Deployed to a Specific Location

1. Open the Locations Report to see the registered Locations.

2. Click the Location name link to see its Object Summary Report.

3. Filter the information shown by using object type and object status as required.

4. Click the **Name** link for a Data Object to see its Deployment Report.

   Details are shown for all deployments of this Data Object and its second-class Data Objects.

Identifying the Map Runs that Use a Specific Deployed Data Object

1. Open the Object Summary Report to see list of all deployed objects.

2. Filter the information shown by using object type and object status as required.

3. Click the **Name** link for a data object to see its Deployment Report.

4. Select the Execution Report tab to display an Execution Report which is a a summary of how and when the data object was used in a Map Run.

5. Click the related **Run Execution Report** link to display an Execution Report which shows details of any Map Run.

Discovering the Default Deployment-Time Settings of a Deployed Process

1. Open the Object Summary Report to see all deployed Processes.

2. Click the **Process Name** link to see its Deployment Report.

3. Select the Start Report tab for the given Process to display a Job Start Report for the Process.

   The execution parameters have the default deployment-time settings

4. Change any of the input parameter values, as required

5. Click **Start** to execute the new Process Run.

Rerunning a Process

1. Open the Execution Schedule Report to see list of all Process Runs.

2. Click the **Run Execution Report** link to display the Execution Report for the given Process Run.

3. Click the appropriate related link to display the Job Start Report for the given Process.

   The execution parameters have the default deployment-time settings.

4. Change any of the input parameter values, as required.
5. Click **Start** to execute a new Process Run.

Monitoring a Process Run

1. Open the Execution Schedule Report to see the executing Process Runs.
2. If necessary, use the Execution Status filter to display only currently executing Process Runs.
3. Click **Refresh** as required, to follow the progress of the Process Runs.
4. Click the **Run Execution Report** link to display the Execution Report which shows the details of a given Process Run.
5. Click **Refresh** as required, to follow the progress of the Process Run.
6. For Process Runs known to the Workflow system, click the **Related information** link to switch across to the Oracle Workflow Monitor and follow its progress in a graphical display – use the browser's Back button to return to the current report page.

Aborting a Process Run

1. Open the Execution Schedule Report to see the executing Process Runs.
2. Click the **Run Execution Report** link to display the Execution Report that shows details of a given Process Run.
3. Click **Stop** to abort the given Process Run.
4. Click **Refresh** as required, to follow the progress of the Process Run as its execution is terminated.

Removing the Execution Audit Details for a Process

1. Open the Execution Schedule Report to see the latest Processes Runs.
2. Filter the information shown by using execution name.
3. Select all the executions which are to be removed, and click **Purge Selected Audit Details**.

   A confirmation screen will be shown, requesting that the action be confirmed.

Removing Old Deployment Audit details

1. Open the Deployment Schedule Report to see the latest deployments.
2. Filter the information shown by using date range.
3. Select all the deployments which are to be removed, and click the **Purge Selected Audit Details**.

   A confirmation screen will be shown, requesting that the action be confirmed.

Unregistering a Location

1. Open the Locations Report to see the registered Locations.
2. Select the Location which is to be unregistered, and click **Unregister Selected Locations**.

A confirmation screen will be shown, requesting that the action be confirmed.

Updating Location Connection Details for a Changed Database Environment

1. Open the Locations Report to see the Locations.
2. Select the Location which is to be validated, and click the **Validation** link.

   The Location Validation Report will be displayed, showing the connection details of the Location and the Control Center
3. Change the service description values, as necessary, and click **Update Details**.
4. Click **Test Connection** to validate the current connection settings for the Location.

   Note that the results of Location connection tests are not maintained beyond the current session,

Updating Service Node Details in a Changing RAC Environment

1. Open the Service Node Report to see the settings which currently describe the RAC system.
2. Update details and usage of the Service Nodes, then click **Update Node Details** for the requested changes to be made.
3. **Add** or **Remove** Service Nodes, as required.
4. Click **Refresh** to see the current settings of the RAC system.
5. Set the Net Service Name by which the Control Center may be accessed, as necessary.

Part VI

Reference for Managing Metadata

This part contains the following chapters:

31

Managing Metadata Dependencies

The Metadata Dependency Manager enables you to detect and resolve the impact of the changes made to the object definitions or the metadata in the Warehouse Builder repository.

This chapter contains the following topics:

Introduction to the Metadata Dependency Manager

The Metadata Dependency Manager generates lineage and impact diagrams for any data object. A lineage diagram traces the process flows for an object back to the data source and displays all objects along that path. An impact diagram identifies all the objects that are derived from selected object.

This type of information can help you in many circumstances. For example, you can use these diagrams to:

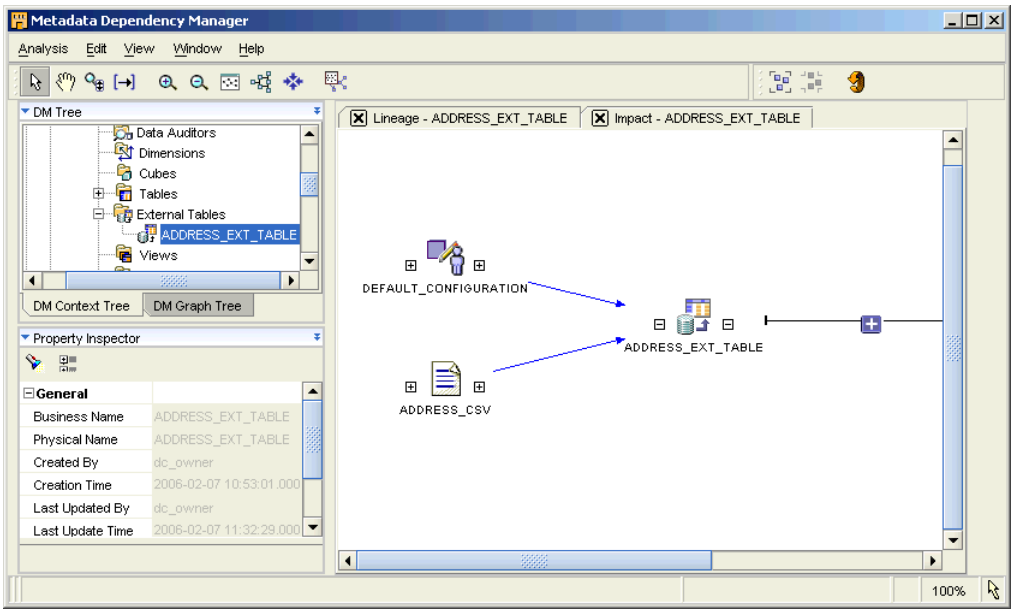
- Comply with government regulations or other audits that require tracking of financial data.
- Modify the system design because of changes in the source data.
- Modify the system design because of new requirements for reporting and analysis.
- Assess the impact of design changes in a pluggable map that is used throughout a project.

The Dependency Manager enables you to plan your project by previewing the impact of the changes or future changes. If you are planning to introduce changes to your source systems, you can use the Dependency Manager to gauge the impact of that change on your warehouse design. Or, if the change has already been introduced, then you can plan the time required to update your ETL design and rebuild your data warehouse.

Figure 31–1 shows the Metadata Dependency Manager. Like other windows in Warehouse Builder, the Dependency Manager has menus, toolbars, a navigator, a property inspector, and a canvas. The canvas displays one or more diagrams.

For specific information about these components, choose **Topic** from the Help menu.

Figure 31–1 Metadata Dependency Manager



Usage Scenario

Source databases often undergo changes even after a data warehouse has been designed and built. These changes imply a corresponding change, or impact, in your Warehouse Builder design or metadata. Warehouse Builder enables you to reimport the modified source definitions into the repository. However, your original warehouse design may no longer remain valid with the re-imported definitions, and you may need to make changes to the design and fix the inconsistencies.

You need to first find out how the warehouse design is affected by the changes in source and then determine all the design objects that are dependent upon the sources must synchronize the metadata so that all the affected design objects are updated to reflect the changes in the source. After this process is complete, you can redeploy the updated design to rebuild your data warehouse and synchronize the data.

In this scenario, a company retrieves all its statistical reports from a flat file named `CUSTOMERS`. Over a period of time, the file definition needs to be changed to generate reports on additional parameters. The statistical analysis database has also recently migrated from Oracle 9*i* to Oracle Database 10*g*.

The designers at this company first need to synchronize the modified metadata definitions for this file in the design repository with all the design objects and mappings that are based on its data. The `CUSTOMERS` flat file is part of multiple mappings in the ETL design, and any changes to its definitions invalidate all them. Manually tracking all the areas impacted by the `CUSTOMERS` flat file and updating the definitions is a process prone to errors.

The Dependency Manager enables them to identify all of the objects that are affected by this change to the data source. They can be sure that each related design object is updated so that the ETL designs remain valid. After finishing this process, they can redeploy the affected tables and the updated ETL design objects to their data warehouse.

What are Lineage and Impact Analysis Diagrams?

Lineage and Impact Analysis (LIA) diagrams show the relationships among objects managed by Warehouse Builder. These relationships are constructed by mappings and process flows. The lineage diagram for a particular object shows its source objects, and the impact diagram shows its targets.

Lineage and impact are mirror images of each other. If Object A is part of the lineage diagram of Object B, then Object B is part of the impact diagram of Object A. When you read a diagram from left to right, you are seeing impact. When you read it from right to left, you are seeing lineage.

For example, you might have a mapping that extracts data from a file and loads it into a table by way of an external table. This is the relationship:

```
flat_file > external_table > table
```

Figure 31–2 shows a lineage diagram of an external table named ADDRESS_EXT_TABLE. ADDRESS_CSV is a flat file, and it is part of the lineage of ADDRESS_EXT_TABLE. Thus, any change to ADDRESS_CSV will impact ADDRESS_EXT_TABLE.

Figure 31–2 Lineage Analysis Diagram for ADDRESS_EXT_TABLE

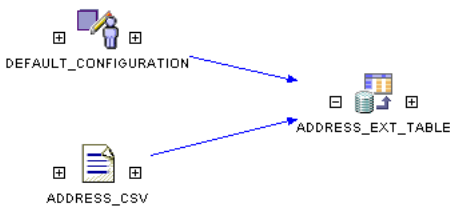
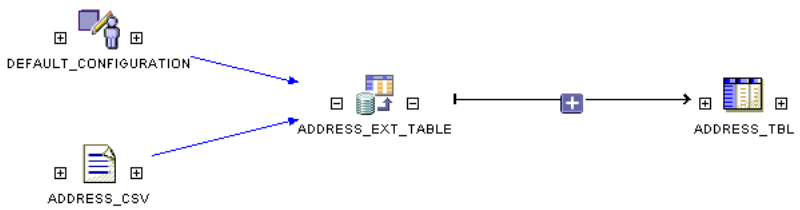


Figure 31–3 shows an impact diagram of ADDRESS_EXT_TABLE, which includes the ADDRESS_TBL. Any change to ADDRESS_EXT_TABLE will impact ADDRESS_TBL. ADDRESS_EXT_TABLE is part of the lineage of ADDRESS_TBL.

Figure 31–3 Impact Analysis Diagram for ADDRESS_EXT_TABLE

You can easily see both the lineage and the impact of an object just by clicking the plus signs (+) on either side of the object icon, as shown in Figure 31–4.

Figure 31–4 Lineage and Impact Analysis Diagram



Generating an LIA Diagram

You can generate an LIA diagram from the Project Explorer in the Design Center or by first opening the Metadata Dependency Manager.

To generate a diagram from the Design Center:

1. Expand the Project Explorer until you see the object that you want to analyze.
2. Right-click the object and choose **Lineage** or **Impact** from the popup menu.

   The Metadata Dependency Manager will open with the diagram.

The Lineage and Impact commands are also available from the View menu.

To generate a diagram from the Metadata Dependency Manager:

1. From the Design Center Tools menu, choose **Metadata Dependency Manager**.

   The Metadata Dependency Manager is displayed.
2. Expand the DM Context Tree until you see the object that you want to analyze.
3. Right-click the object and choose **Show Lineage** or **Show Impact** from the popup menu.

   The diagram will be displayed on the canvas.

Modifying the Display of an LIA Diagram

Your initial selection of an object and a diagram type simply determine the initial starting point and the direction that the diagram branches from that object. You can modify an LIA diagram in the following ways:

- Drag-and-drop another object onto the diagram.
- Click the plus (+) and minus (-) signs next to an object icon to expand or collapse a branch of the diagram.
- Use the grouping tool to collapse a section of the diagram into a single icon, as described in "Using Groups in an LIA Diagram" on page 31-5.
- Double-click an object to display its attributes, as described in "Displaying an Object's Attributes" on page 31-5.

Using Groups in an LIA Diagram

Groups enable you to organize the objects in a complex diagram so that they are easier to locate and edit. By reducing the number of objects in a diagram, you can more easily focus on the objects currently of interest.

To create a group:

1. Select a group of objects by dragging and dropping a box around them.
2. Click the Group Selected Objects tool.

   The Group Selected Data Objects dialog box is displayed.
3. Enter a name for the group.

   The selected objects are collapsed into a single folder icon.

To display the individual objects in a group, double-click the folder icon. You can work on these objects in the same way as ungrouped objects.

To ungroup the objects, select the group and click the Ungroup Selected Object tool.

Displaying an Object's Attributes

You can expand an object icon in a diagram so that you can examine its attributes. Figure 31–5 shows two expanded icons whose column attributes are connected by a mapping.

To expand an icon, double-click it. To reduce it to an icon, click the down arrow in the upper right corner.

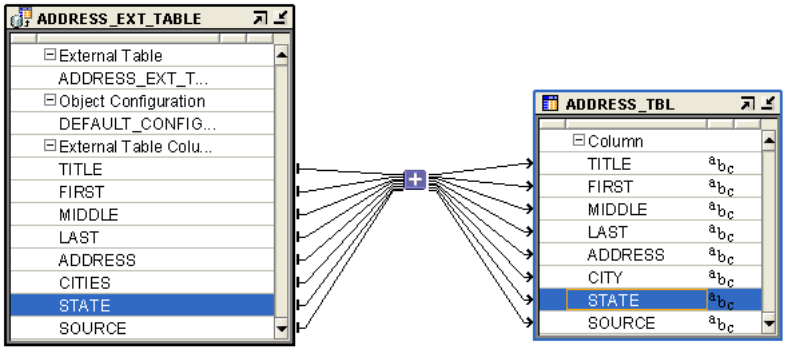
To generate an LIA diagram for an attribute:

1. Generate an LIA diagram for an object.
2. Double-click the icons to display their attributes.
3. Right-click an attribute and select **Show Lineage** or **Show Impact**.

   The attributes along the lineage or impact path for the selected attribute are highlighted in a different color.

You may use this detailed information for auditing or when planning to propagate changes.

Figure 31–5 Expanded Icons in an LIA Diagram



Modifying Objects in the Dependency Manager

The LIA diagrams identify all of the objects that may be invalidated by a change to one or more objects. With this knowledge, you can examine the affected objects and modify them as necessary.

You can make these changes within the Dependency Manager, using either of these methods:

- Object Editor: Provides extensive editing capabilities by opening the standard editor for the object.
- Propagate Changes tool: Provides basic editing capabilities such as changing the name, data type, or precision for a single column or attribute.

To open the object editor:

Right-click the object icon in a diagram, and choose **Open Editor** from the popup menu.

To use the Propagate Changes tool:

1. Double-click the object icon in a diagram.

   The icon will expand to show its attributes.

2. Right-click the attribute that you want to change, and choose **Propagate Change** from the popup menu.

   The Propagate Change dialog box is displayed.

3. Make your changes to the attributes in the New Value column and select the Propagate box.

4. Click **OK**.

Metadata Dependency Manager

The Metadata Dependency Manager is a graphical interface that represents the potential impact of a change in the definition of an object. It has the following components:

- Menu Bar
- Toolbars
- DM Tree
- Property Inspector
- Canvas
- Bird's Eye View

Menu Bar

The Metadata Dependency Manager menu bar provides commands for performing various tasks. Some of these commands are also available on the toolbars.

Analysis

The Analysis menu contains the following commands:

- **Close**: Closes the Dependency Manager.

- **Export Diagram**: Exports the active diagram to the local file system as an SVG or JPEG file.
- **Print Options**: Provides Print Setup, Preview, and Print options for printing the diagram.

Edit

The Edit menu contains the following commands:

- **Open Editor**: Opens the Editor for the currently selected object.
- **Hide**: Removes the selected objects from the canvas. Use the Refresh command on the View menu to restore the hidden objects.
- **Select All**: Selects all objects displayed on the canvas.
- **Propagate Changes**: Displays the Propagate Changes dialog box for a selected attribute. Use it to change the value of an attribute and to propagate that change to all objects downstream. For example, you can select a column in a table object and change its name, data type, and so forth.
- **Group Selected Objects**: Creates a group containing the selected objects on the canvas. A folder icon represents all objects in the group. Double-click the icon to display the individual objects in the group. Grouping enables you to reduce clutter on the canvas when there are many objects.
- **Ungroup Selected Objects**: Eliminates the selected group so that all objects are represented individually.
- **Group By Module**: Automatically groups all objects by module. A folder icon represents the module and all objects in the module. Double-click the icon to display the individual objects in the group.
- **Ungroup Modules**: Eliminates the module groups so that all objects are represented individually.

View

The View menu contains the following commands:

- **Toolbars**: Displays or hides the Graphic tools or the Edit tools.
- **Mode**: Sets the pointer for one of these actions:
  - **Select**: Selects one or more objects on the canvas.
  - **Pan**: Moves the entire diagram on the canvas.
  - **Interactive Zoom**: Expands the diagram as you move the pointer down, or shrinks the diagram as you move the pointer up.
  - **Navigate Edge**: Selects the next object in the flow.
- **Zoom**: Displays a list of percentages that expand or shrink the diagram.
- **Fit in Window:** Automatically chooses a size for the diagram so that it fits on the canvas.
- **Auto Layout**: Organizes the objects and displays the diagram at its default size.
- **Center:** Centers the diagram on the canvas.
- **Show Full Impact**: Generates the full impact diagram of the selected object.
- **Show Full Lineage**: Generates the full lineage diagram of the selected object.

- **Show Lineage**: Displays the next level of objects in the lineage diagram of the selected object.
- **Hide Lineage**: Hides the lineage of the selected object.
- **Show Impact**: Displays the next level of objects in the impact diagram of the selected object.
- **Hide Impact**: Hides the impact of the selected object.
- **Expand**: Expands the selected icon in the diagram. The expanded icon shows the details of the object, such as the columns in a table or the attributes in a dimension.
- **Expand All**: Expands all the object icons in the diagram.
- **Collapse**: Collapses the expanded icon for a selected object.
- **Collapse All**: Collapses all the expanded icons on the canvas.
- **Refresh**: Refreshes the Dependency Manager diagram to reflect recent changes in the repository.

Window

The Metadata Dependency Manager Window menu contains commands for toggling between displaying and hiding the following windows:

- **Property Inspector**
- **Bird's Eye View**
- **Tree View**

Toolbars

The Metadata Dependency Manager provides two toolbars as shortcuts to frequently used commands:

- **Graphic toolbar**: Provides icons for commands on the View menu. Refer to "View" on page 31-7 for descriptions of these commands.
- **Edit toolbar**: Provides icons for commands on the Edit menu. Refer to "Edit" on page 31-7 for descriptions of these commands.

Bird's Eye View

Use the Bird's Eye View to quickly change the portion of the diagram currently displayed on the canvas. This view displays a miniature version of the diagram on the canvas, with a scrollable box that represents the dimensions of the canvas. Drag the box to the area of the diagram currently of interest to you.

DM Tree

Use the DM Tree to change the content of the canvas. The DM Tree has these tabs:

- **DM Context Tree**: Lists objects in the current project. Right-click an object, and use the popup menu to generate a new diagram or to add the object to the current diagram. You can also drag-and-drop an object onto the current diagram.
- **DM Graph Tree**: Lists the current diagrams. You can close a diagram on the canvas, then use this list to redisplay the diagram without regenerating it.

Property Inspector

Use the Property Inspector to view an object's properties.

To change the current object in the Property Inspector, right-click the object on the canvas and select **Update Property Inspector** on the popup menu.

For a description of a property, select the property in the Inspector. The description appears at the bottom of the window.

Canvas

Use the canvas to display one or more lineage and impact diagrams. Each diagram is displayed on a separate tab.

You can use these techniques to create and manipulate diagrams on the canvas:

- To open a new diagram, right-click an object in the DM Context Tree and choose **Show Lineage** or **Show Impact** from the popup menu.
- To close a diagram, click the **X** on the tab. You can redisplay the diagram from the DM Graph Tree until you close the Metadata Management window.
- To add an object to the canvas, drag-and-drop it from the DM Context Tree.
- To display more of the lineage of an object, click the plus sign (+) to the left of its icon. To hide the lineage, click the minus sign (-) to the left.
- To display more of the impact of an object, click the plus sign (+) to the right of its icon. To hide the impact, click the minus sign (-) to the right.
- To display the details of an object, double-click it. You can then select an individual property and edit it by choosing **Propagate Changes** from the Edit menu.
- To edit an object, right-click its icon and choose Open Editor from the popup menu.

Propagate Change Dialog Box

Use this dialog box to make changes to an attribute and to propagate those changes to the repository.

Propagate

Identifies whether the change to an attribute value is propagated to the repository. Select this box to propagate the changes.

Attribute

Lists the attributes for the selected object, such as the columns of a table.

Existing Value

Displays the current values of the attributes, such as the data type.

New Value

Enter a new value for one or more attributes.

32

Managing Metadata Changes

Oracle Warehouse Builder provides a metadata snapshot feature that enables you to backup and restore your metadata, maintain a history of metadata changes, and compare different versions of the metadata.

This section contains the following topics:

This chapter describes the metadata change management feature using the Warehouse Builder GUI. For information about creating and managing snapshots using scripts, refer to the *Oracle Warehouse Builder API and Scripting Reference*.

About Metadata Snapshots

A snapshot contains all the information about its objects and the relationships for the object. While an object can only have one current definition in the repository, it can have multiple snapshots that describe it at various points in time. Warehouse Builder supports two types of snapshots:

- *Full snapshots* provide backup and restore functionality.
- *Signature snapshots* provide historical records for comparison.

Snapshots are stored in the database, in contrast to Metadata Loader exports, which are stored as separate disk files. You can, however, export snapshots to disk files.

Snapshots can be useful to both warehouse managers and designers. Managers can use full snapshots to perform large-scale actions, such as deploying a warehouse or restoring a warehouse to a previous point in history. Designers can create full snapshots of a particular component under development so that, if necessary, they can restore the component to its previous state. They can also use signature snapshots to track the changes made to a particular component.

When used with other Warehouse Builder facilities, such as MDL metadata imports and impact analyses, snapshots can help you manage your metadata. For example, you can use snapshots to determine the impact an MDL metadata import will have on the current metadata repository. With the knowledge gained by the comparison, you might import the metadata at a lower level of detail to avoid overwriting the definition of related metadata objects.

Snapshots are also used by Warehouse Builder to support the recycle bin, providing the information needed to restore a deleted metadata object.

Creating Snapshots

Metadata change management enables you to take snapshots that capture the content of your metadata repository, or specific objects in your repository, at a given point in time. You can use a snapshot to detect and report changes in your metadata.

Note: A snapshot of a collection is not a snapshot of just the *shortcuts* in the collection but a snapshot of the actual objects.

You can create snapshots of any objects that you can access from the Project Explorer.

Opening the Create Snapshot Wizard

To open the Create Snapshot wizard and create a snapshot:

1. In the Design Center Project Explorer, select all the components you want to include in the snapshot. You do not need to select required or child objects, because the wizard does that for you.

   For example, if you select a collection containing two cubes, then both the cubes are included in the snapshot. You can optionally add the dimensions, source tables, and so forth automatically.

2. Right-click and select **Snapshot**, then **New** from the pop-up menu.

   or

   From the Design menu, select **Snapshot**, then **New**.

   The Welcome page of the Create Snapshot wizard is displayed.

3. Click **Next.** The Name page of the wizard is displayed.

   Specify a name and the type of snapshot. You can also provide a description (optional).

4. Click **Next** to open the Components page of the wizard.

   The page displays the components whose snapshot will be taken. If a component is a folder level object, such as a module, then select the Cascade option to include the subcomponents.

5. Click **Next** to open the Dependency page.

   Specify the depth of dependency to include dependent objects while taking the snapshot.

6. Click **Next**. The Finish page is displayed and it provides the details of the snapshot.

7. Click **Finish** to create the snapshot.

Adding Components to a Snapshot

After you create a snapshot, you can add more components. Keep in mind, however, that when you add components, you are changing the historical record provided by the snapshot, so its creation date changes.

To update a snapshot, use the Add to Snapshot wizard.

Opening the Add to Snapshot Wizard

To open the Add to Snapshot wizard and add new components:

1. In the Design Center Project Explorer, select all the components you want to add to the snapshot. For example, you can select tables, mappings, and dimensions from an Oracle module.
2. From the Design menu, select **Snapshot**, then **Add to Existing**.

   or

   Right-click and select **Snapshot**, then **Add to Existing** from the pop-up menus.

   The Welcome page of the Add to Snapshot wizard is displayed. The Welcome page lists the steps of the wizard.
3. Click **Next**.

   The Snapshot page is displayed. From the list, select the snapshot to which you want to add the components.
4. Click **Next**.

   The Components page is displayed. The page displays the components to be added to the snapshot. If a component is a folder level object, such as a module, then select the Cascade option to include the subcomponents.
5. Click **Next**.

   The Finish page displays the details of the components to be added.
6. Click **Finish** to add the components to the snapshot.

Managing Snapshots

You can manage your snapshots from the Metadata Change Management window in the Design Center. To open this window, select **Change Manager** from the Tools menu.

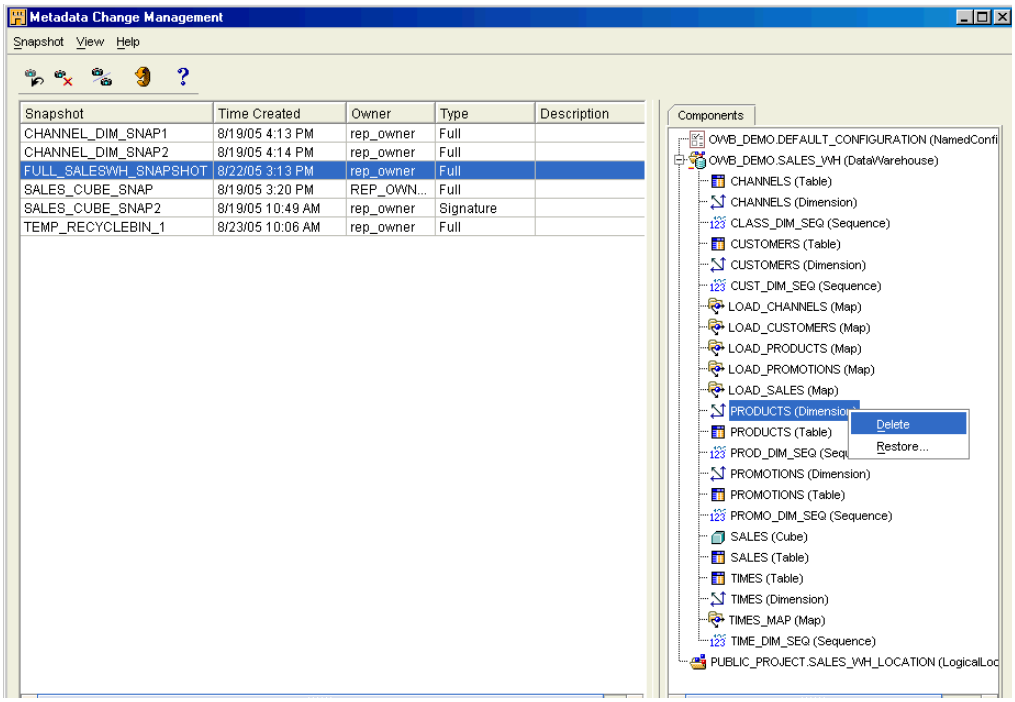
The Metadata Change Management window contains a menu bar and a tool bar. You can initiate most tasks in several different ways, either by using the menus, clicking the tools, or right-clicking a snapshot or a component.

You can perform the following activities from the Metadata Change Management window:

- Managing Snapshot Access Privileges on page 32-4
- Comparing Snapshots on page 32-4
- Restoring Repository Objects From Snapshots on page 32-6
- Exporting and Importing Snapshots on page 32-7
- Deleting Snapshots on page 32-7

Figure 32–1 shows the Metadata Change Management window.

Figure 32–1 Metadata Change Management Window



Managing Snapshot Access Privileges

You can control access to snapshots just like any other Warehouse Builder object. By default, everyone has full access rights to the snapshots.

To change access privileges:

1. In the left section of the Metadata Change Management window, right-click the name of the snapshot.
2. Click **Security** from the menu.

   The Snapshot Privilege Management dialog box is displayed.
3. For each role and user, select the privileges you wish to grant and clear the privileges you wish to deny. Click **Help** for additional information.

Comparing Snapshots

You can compare two snapshots or a snapshot with a current repository object. The results list all objects and identify which objects are identical, which objects appear only in one place, and which objects appear in both but are not the same.

If you take a snapshot of a parent object and a child object changes, a snapshot comparison will show the parent object as having changed. For example, if you create a Cascade snapshot of a project, all the modules in that project are its children. If any of the modules change, a comparison of the snapshot to the repository will show that the project has changed.

The Metadata Change Manager uses Universal Object Identifiers (UOIDs) as the basis for all comparisons. When you delete an object and re-create it with the same metadata, the object has a different UOID although it may be identical to the deleted object in all other aspects. When you compare the re-created object to its original version, the results show that all the metadata has changed. Likewise, objects will not

match if they have been deleted and re-created during an import or Intelligence Object derivation.

Comparing Two Snapshots

Use one of the following methods:

1. From the Metadata Change Management window, select **Compare** from the Snapshot menu.

   Or

   Select the two snapshots you want to compare, right-click, and select **Compare** from the menu.

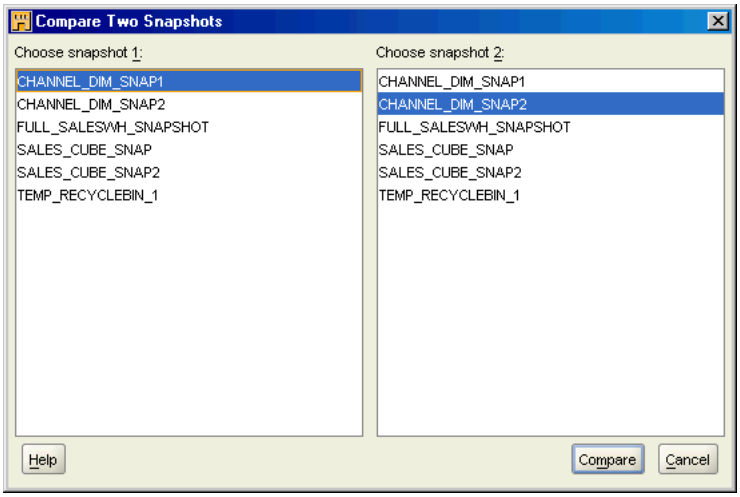
   The Compare Two Snapshots dialog box is displayed, as shown in Figure 32–2. Snapshots that were selected in the Metadata Change Management window are also selected in the Compare Two Snapshots dialog box.

2. If you have not selected two snapshots for comparison yet, select one from each list.

3. Click **Compare.**

   The Snapshot Comparison window displays the differences between the two objects. If there are none, then a message informs you that the objects are the same. For more information, click **Help**.

Figure 32–2 Compare Two Snapshots Dialog Box



Comparing a Repository Object with a Snapshot Component

To compare the current version of an object with a snapshot version:

1. Select the object you want to compare from the Design Center Project Explorer.

2. Right-click, select **Snapshot**, and then Compare.

   The Choose Snapshot dialog box is displayed with a list of all the snapshots containing versions of that object.

3. Select the snapshot you want to compare with the current repository object.

4. Click **OK.**

The Snapshot Comparison window displays the differences between the two objects. If there are none, then a message informs you that the objects are the same.

Converting a Full Snapshot to a Signature Snapshot

You can convert full snapshots to signature snapshots when they are no longer needed for backup. Conversion preserves the snapshot history and results in significant space savings in your repository.

Note: Converting a full snapshot to a signature snapshot means that you can no longer use that snapshot to restore metadata objects.

To convert a full snapshot:

1. Open the Metadata Change Management window, as described in "Managing Snapshots" on page 32-3.
2. Select the snapshot that you want to convert.
3. From the Snapshot menu, choose **Convert to Signature**.
4. On the Warehouse Builder Warning box, click **Yes**.

Restoring Repository Objects From Snapshots

You can replace the current definition of an object in the repository with the snapshot image of that object. You can only use full snapshots; you cannot restore objects from signature snapshots. You can restore all components or only selected components of the snapshot.

Note: When you restore a collection from a snapshot, you restore both the collection and the actual objects.

Similarly, when you restore a container, all its child objects are also restored.

To restore only the collection or selected objects within a container, use the Components tab.

To restore objects from a snapshot:

1. Save any work that you have done in the current session.
2. Open the Metadata Change Management window, as described in "Managing Snapshots" on page 32-3.
3. Select the snapshot that contains the version of the objects you want to restore.
4. To restore all objects, right-click the snapshot and select **Restore** from the menu.

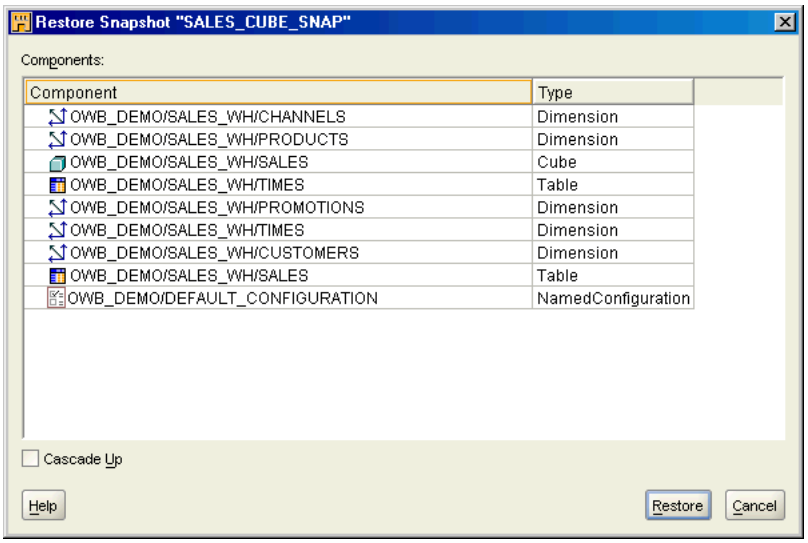
   or

   To restore selected components, select them on the Components tab, right-click, and choose **Restore** from the menu.

   The Restore Snapshot dialog box is displayed, as shown in Figure 32–3.
5. Review the objects in the list and verify that the correct objects will be restored.

6. Click **Restore**.

Figure 32–3 Restore Snapshot Dialog Box



Exporting and Importing Snapshots

You can export a full snapshot to an MDL file and use this file to re-create metadata objects either in the same database or a different one. You cannot export signature snapshots or individual components of a snapshot.

To export a snapshot:

1. Open the Metadata Change Management window and select the full snapshot you want to export.
2. From the Snapshot menu, choose **Export**.

   The Metadata Export dialog box is displayed.
3. Click **Help** for information about completing the dialog box.

To import a snapshot:

1. Open the Design Center and choose **Import** from the Design menu.
2. Choose **Warehouse Builder Metadata** from the menu.

   The Metadata Import dialog box is displayed.
3. Click **Help** for information about completing the dialog box.

The imported snapshot will be listed in the Metadata Change Management window.

Deleting Snapshots

You can delete snapshots or components within a snapshot from the Metadata Change Management window.

To delete a snapshot:

1. Open the Metadata Change Management window and select the snapshot you want to delete.
2. Right-click the snapshot and select **Delete** from the menu.

Or

From the Snapshot menu, select **Delete.**

Warehouse Builder displays a delete confirmation box.

3. Click **Yes** to delete the selected snapshot from the repository.

To delete components from a snapshot:

1. Open the Metadata Change Management window and select the snapshot from which you want to delete components.
2. On the Components tab, select the components, right-click, and select **Delete** from the pop-up menu.

   Or

   From the Snapshot menu, select **Delete.**

   Warehouse Builder displays a delete confirmation dialog box.
3. Click **Yes** to delete the selected component from the snapshot.

33

Importing and Exporting with the Metadata Loader (MDL)

The Metadata Loader (MDL) enables you to populate a new repository as well as transfer, update, or restore a backup of existing repository metadata. You can also take snapshots of your metadata and use them for backup, compare, and restore purposes.

This section contains the following topics:

Overview of Import and Export Using Metadata Loader

Warehouse Builder provides several features that enable you to copy and move metadata for the purposes of backup, history management and version management. You can import and export metadata for any type of object on the Project Explorer, Global Explorer, and Connection Explorer using the Metadata Loader (MDL) utility. You can then move exported files into a third-party version control tool such as Oracle Repository, ClearCase, or SourceSafe. You can enter annotations for your MDL export file to keep track of the information contained in the file. The MDL enables you to copy or move metadata objects between repositories, even if those repositories reside on platforms with different operating systems.

You can also perform metadata change management by taking snapshots of your metadata using Warehouse Builder Design Center or OMB Plus scripting. Snapshots enable you to capture definitions of metadata objects using Warehouse Builder scripts. Use snapshots for metadata backup and version management. For more information about metadata change management, see Chapter 32, "Managing Metadata Changes".

This section contains the following topics:

Metadata Loader Utilities

The Metadata Loader consists of the following two utilities:

- Metadata Export Utility

  Use the Metadata Export Utility to export metadata from a Warehouse Builder repository.

- Metadata Import Utility

  Use the Metadata Import Utility to import metadata into a Warehouse Builder repository.

MDL uses its own format, and the Metadata Import Utility only reads files of MDL format (files created by the Metadata Export utility). The Metadata Loader file is a formatted ZIP file.

Metadata Export Utility

The Metadata Export Utility extracts metadata objects from a repository and writes the information into a ZIP format file. This ZIP file has a .mdl extension and contains the following files:

- Metadata Loader XML file

  This file contains the objects extracted from the repository and formatted in XML. It has the same name as the of the ZIP file, but with the extension .mdx.

- Catalog

  The catalog file is called `mdlcatalog.xml` and it contains internal information about the Metadata Loader XML file.

The Metadata Export Utility enables you to specify a file name and a path for the exported MDL file. For example, you export the repository metadata into a file called `sales.mdl`. When you unzip this MDL ZIP file, you obtain two files. The file `sales.mdx` contains the repository objects. The file `mdlcatalog.xml` contains internal information about the MDL XML file.

You can export an entire project, collections, public objects, locations, or any subset of objects. If you export a subset of objects, the MDL exports definitions for each object that you have selected and the parent objects to which the subset belongs. This enables the MDL to maintain the tree relationships for those objects during metadata import.

For example, if you export a single dimension, the export file contains definitions for:

- The dimension
- The module to which the dimension belongs
- The project to which the module belongs

If you are exporting a subset of objects, make sure you export all referenced objects and import them as well. You can export the objects referenced by a set of objects by selecting the **Export All Dependencies** option on the Metadata Export dialog. For example, if you export a table `DEPT` and it contains a foreign key reference to the table `EMP`, you can choose to export `EMP` along with `DEPT`.

Metadata Import Utility

The Metadata Import Utility reads the metadata information from an exported MDL file and creates, replaces, or merges the metadata objects into a repository. It imports information belonging to exported metadata objects such as table columns and their constraints, data loading configuration parameters, and named attribute sets. The Metadata Import Utility enables you to import repository objects even if the references for those objects cannot be satisfied.

You can use the Metadata Import Utility to import objects into a project or a collection. The Metadata Import Utility only reads files created by the metadata export utility.

If the MDL file being imported was created using an earlier version of Warehouse Builder, the Metadata Import Utility automatically upgrades it to the current version of Warehouse Builder. For more information on the automatic upgrade of MDL files, see "Upgrading Metadata from Previous Versions" on page 33-15.

Uses of Metadata Loader

Use the metadata loader to perform any of the following tasks:

- **Backup metadata:** The MDL is an important part of your disaster recovery strategy. You can export a file with your existing repository metadata as a backup and use that exported file to restore a repository if necessary.
- **Seed a new repository:** You can export data from an existing repository and use it as the basis for a new repository.
- **Migrate a repository:** You can export the metadata from a previous Warehouse Builder version to a file and then import the file into a newer version of Warehouse Builder. You need to do this when you upgrade to a newer version of Warehouse Builder.
- **Automatically upgrade from a previous version of MDL data file:** When you import metadata from an MDL data file that has been created using a previous version of Warehouse Builder, the MDL automatically upgrades the data file to the current version.
- **Copy metadata:** A multiple user development environment can result in multiple copies of the same metadata. The MDL makes it easy to load single set metadata into more than one repository.

Accessing the Metadata Loader

You can access the Metadata Loader using one of the following:

- Design Center

  When you use the menu in the Design Center, a graphical interface guides you through the export or import process as described in"Using Metadata Loader with Warehouse Builder Design Center" on page 33-5.

- OMB Plus or Scripting

  OMB Plus provides the following commands to export and import metadata: OMBEXPORT, OMBIMPORT, OMUEXPORT, and OMUIMPORT. Use the OMBUIMPORT and the OMPUEXPORT commands to display the Metadata Loader user interface from OMB Plus. For more information on importing metadata using the OMBIMPORT and OMBEXPORT commands, see "Using Metadata Loader with OMB Plus" on page 33-18.

Multiple Session Concurrency and MDL

The Warehouse Builder repository allows multiple clients to access the same repository schema concurrently. Warehouse Builder uses locks to allow only one client to change repository objects. While an object is locked, other clients can only view it as it existed after the last transaction instigated by any user is saved.

When replacing or merging objects, the MDL acquires locks on the primary objects that exist both in the repository and in the MDL file. Primary objects include, but are

not limited to, projects, modules, tables, dimensions, cubes, mappings, views, and flat files. Secondary objects, such as columns and mapping attributes, are not locked. If locks cannot be obtained because other users are locking the primary objects, then the import fails. Therefore, you must be able to hold locks for primary objects that you are importing.

If the target repository is not an Oracle 10*g* database and the MDL import affects too many objects in the repository, the MDL automatically switches to single user mode. This means that no other users can log on to the repository until after the MDL import completes. Single-user mode allows the MDL to avoid the performance degradation that results from using a large number of locks. In single-user mode, the MDL is less likely to deplete the repository enqueue resources. If other users are logged into this repository when MDL attempts to switch to single-user mode, MDL cannot switch to single-user mode and subsequently fails.

Tip: To ensure a successful metadata import, you may need to be the sole client accessing the repository.

The MDL saves changes made to the repository after a successful metadata import (any import with no error messages, including imports with only information or warning messages). The MDL also executes a rollback after an unsuccessful import.

Metadata Loader Log File

Whenever you export or import repository metadata, the MDL writes diagnostic and statistical information to a log file. You can specify the location of the log file when you invoke the MDL.

The log file enables you to monitor and troubleshoot export and import activities in detail and contains the following information:

- Name of the data file
- Start time and end time of the export or import
- Time taken for the export or import in hours, minutes, and seconds (in hh:mi:ss format) or milliseconds
- Object types exported or imported
- Number of objects of each object type exported or imported

  The import log file also displays the total number of objects that have been added, replaced, skipped, and deleted.

- Status messages

  Status messages provide information about the import or export process. They are of the following types:

  - **Informational:** Provides information about the import or export, such as missing metadata objects, whether or not objects were imported, and any reasons why objects were not imported or exported.
  - **Warning:** Cautions you about the import or export of an object but does not indicate a failed or aborted export or import. A warning notifies you of the possibility of unexpected results that could occur as a result of the export or import.

- **Error:** Indicates that the MDL export or import was aborted and did not complete successfully. The error message provides a brief description of the reason for the failure.

About Metadata Loader Results

When you use the Metadata Loader Export or Import utilities, you can view the results of a successful export or import task. You can use the Metadata Export Results dialog or the Metadata Import Results dialog to ensure that all of the objects were exported or imported. To view the results dialog, click **Show Statistics** on the Metadata Export Progress dialog or the Metadata Import Progress dialog.

The results dialog contains the following information:

- The name of the project exported or imported (if applicable).
- The number of objects of each type exported or imported.
- The number of objects of each object type skipped.

  Details about the number of skipped objects is displayed only when you import metadata.

Using Metadata Loader with Warehouse Builder Design Center

You can use the Design Center to run the Metadata Loader utilities. The Design Center provides a graphical interface that guides you through the process of exporting and importing metadata.

This section contains the following topics:

- Exporting Metadata Using Warehouse Builder Design Center on page 33-5
- Importing Metadata Using Warehouse Builder Design Center on page 33-8
- Upgrading Metadata from Previous Versions on page 33-15

Exporting Metadata Using Warehouse Builder Design Center

You can use Design Center to export objects from a Warehouse Builder repository into an MDL file. This includes objects that are part of the Project Explorer, Connection Explorer, and Global Explorer. The information related to the exported objects, such as table columns and their constraints, data loading configuration parameters, and named attribute sets, are also exported.

Note that all the objects you select for export should belong to the same explorer. For example, you cannot export a table from the Project Explorer and a public transformation from the Global Explorer at the same time. You can export them in two separate steps.

Before Exporting Metadata

Before you attempt to export metadata, ensure you have the following:

- **Required access privileges:** The user preforming the export must have READ privileges on any object that you want to export. You also need to have READ privileges on folder objects. If you do not have READ privileges on a folder object, such as projects or modules, the folder object and all objects that it contains will not be exported. During an export, the Metadata Export Utility skips objects for which you do not have READ privileges. It logs information about the list of

objects that have not been exported due to lack of security privileges in the Metadata Loader Log File.

By default, Warehouse Builder provides READ privileges on all the repository objects to all registered users. If you want to export security information such as users, roles, role assignments, and object privileges, see "Export Advanced Options Dialog" on page 33-7.

To ensure that you are exporting the most up-to-date metadata, verify that you are the sole client accessing the repository. For more details, see "Multiple Session Concurrency and MDL" on page 33-3.

- **Sufficient disk storage:** If you lack sufficient disk space on the machine to which you export the metadata, the export fails. Your destination machine must be able to contain the entire metadata file. The export utility cannot save portions of the metadata file.

To export metadata from a repository using the Design Center:

1. From the Design Center, select the object or objects you want to export. You can select multiple objects by holding down the Ctrl key and selecting the objects.

   You can export individual objects such as tables or groups of objects. When you export projects nodes, or modules, you also export the objects they contain. When you export collections, you also export the objects they reference.

2. From the **Design** menu, select **Export** and then **Warehouse Builder Metadata**.

   If you made changes to the repository metadata prior to running the export utility, the Warehouse Builder Warning dialog is displayed. Click **Save** to save changes or **Revert** to revert to the previously saved version.

   If you have not made any changes to the repository metadata after last saving the design, Warehouse Builder displays the Metadata Export Dialog.

Metadata Export Dialog

The Metadata Export dialog displays the names and the types of objects being exported. It also contains the following:

- **Annotations:** Use this field to enter any comments about the file that contains the exported objects.
- **File Name:** Warehouse Builder displays a default path and file name for the export file. You can retain this default or specify a directory and file name. Type the name of the export file to create or click **Browse** to locate a directory or file. The file name extension commonly used is .mdl.
- **Log File:** Use this field to specify the file name and path for the log file that stores diagnostic and statistical information about the export. For more information about log files, see "Metadata Loader Log File" on page 33-4.
- **Export all object dependencies:** Select this option to export all the dependencies of the objects being exported. For example, when you export a table, the location to which the table is deployed is also exported.

> **Note:** Public objects such as locations, public transformations, public experts, public icon sets, or public data rules belong to a project called PUBLIC_PROJECT. You can export the PUBLIC_PROJECT and its objects if the selected exported objects have a dependency on the public objects and if you select the **Export all object dependencies** option.

- **Advanced:** Use the Advanced button to export additional metadata such as user-defined properties, security information, and additional languages. For more information on the advanced options, see "Export Advanced Options Dialog" on page 33-7.

Click **Export** to export the metadata for the selected objects. The Metadata Export Progress dialog is displayed. For more information on the contents of this dialog, see "Metadata Progress Dialog" on page 33-7.

Metadata Progress Dialog The Metadata Progress dialog displays a progress bar that indicates the progress of the metadata export, import, or upgrade activity. If the export or import is successful, a message indicating this is displayed just above the progress bar. An error in the process is displayed too.

To view detailed information about the metadata export or import, click **Show Details**. The message log is displayed. The message log contains the following information:

- Start time of the export or import
- Names and types of objects exported or imported
- Warning or error messages
- End time of the export or import
- Location of the export or import log file
- Total export or import time in hh:mi:ss or milliseconds

You can hide the message log by clicking **Hide Details**.

To view details about the exported or imported objects, click **Show Statistics**. The Metadata Export Results dialog is displayed. For more information about this dialog, refer "About Metadata Loader Results" on page 33-5.

Once the export or import completes, the Close button is enabled. To exit the Metadata Export or Metadata Import Utility, click **Close**.

Export Advanced Options Dialog

Use the Export Advanced Options dialog to export any of the following:

- Additional language metadata
- User-defined definitions
- Security information

This dialog contains two sections: Languages and Administration.

Languages

The **Base Language** field displays the base language of the Warehouse Builder repository. Warehouse Builder exports data in the base language.

You can specify additional languages to export for the objects that contain translations for their business names and descriptions. The **Available Languages** list displays the list of languages that are installed in the Warehouse Builder repository. To export additional languages, select the language and click the arrows to move the language from the Available Languages list to the **Selected Languages** list. You can choose multiple languages at the same time by holding down the Ctrl or Shift key while making your selection.

> **Note:** The Available Languages list will contain language entries only if you installed additional languages in the repository.

For example, the Warehouse Builder repository has the base language as American English and additional languages Spanish and French. While exporting metadata from the repository, you can select French as the additional language. The Metadata Export Utility then exports the base language of the object, American English, and the additional language French for objects that contain a French translation. Note that additional languages will be exported for an object only if they contains translations for the business names and descriptions.

Administration

You can export additional metadata if you have administrator privileges. The options you can choose to export additional metadata are as follows:

- **Export user-defined definition:** Select this option to export the user-defined definitions for objects that contain any user-defined properties. This option is enabled only if you have created any user-defined properties for the objects being exported.
- **Export security information:** Select this option to include security information such as object privileges or role assignments made to users. For more information about Warehouse Builder security, refer to the *Oracle Warehouse Builder Installation and Administration Guide*.

After you specify the options on the Export Advanced Options dialog, click **OK** to close this dialog and return to the Metadata Export dialog.

Importing Metadata Using Warehouse Builder Design Center

You can use the Design Center to import metadata. The Metadata Import Utility also automatically upgrades metadata that was created using an earlier version of Warehouse Builder to the current version. For more information on upgrading metadata, see "Upgrading Metadata from Previous Versions" on page 33-15.

Before Importing Metadata

Before you attempt to import metadata, ensure you have the following:

- **Required access privileges:** To import metadata, the user performing the import must have the following privileges:
  - EDIT privilege on existing objects that are being replaced by the import.
  - CREATE privilege on existing folder objects under which new objects will be created by the import.

  By default, Warehouse Builder assigns the FULL_CONTROL privilege on all repository objects to registered users. The Metadata Import Utility skips objects for which the user importing metadata does not have the required privileges. The list

of objects that have not been imported due to security privileges are logged to the Metadata Loader Log File.

You can import security information such as users and roles as described in "Import Advanced Options Dialog" on page 33-12. When importing Warehouse Builder user metadata, if a corresponding database user does not exist for a Warehouse Builder user, the import will fail and an error message is written to the Metadata Loader Log File.

Because the Metadata Import Utility is altering the repository, the metadata objects must be locked prior to importing. For more details, see "Multiple Session Concurrency and MDL" on page 33-3.

- **A backup of your current repository:** Consider taking a backup of your existing repository (either in the form of an export or a metadata snapshot) before attempting a large or complex import. For more information on metadata snapshots, see Chapter 32, "Managing Metadata Changes".
- **Multiple Language Support base language compatibility:** The base language is the default language used in the repository and is set using the Repository Assistant during installation. You cannot alter this setting after installing the repository. For more information on setting the base language in a repository, refer to the Oracle Warehouse Builder Installation and Administration Guide.

To import objects from an export file using the Warehouse Builder Design Center:

1. From the Warehouse Builder Design Center, select **Design**, **Import**, and then **Warehouse Builder Metadata**.

   If you had made changes to the repository metadata prior to running the import utility, the Warehouse Builder Warning dialog is displayed. Click **Save** to save changes or **Revert** to revert to the previously saved version.

   If you have not made any changes to the repository metadata after last saving the design, Warehouse Builder displays the Metadata Import dialog.

Metadata Import Dialog

Use the Metadata Import dialog to specify the information required to import Warehouse Builder metadata contained in an export file. Specify the following information on this dialog:

- File Name
- Log File
- Object Selection
- Import Option
- Match By

Click **Advanced** to import metadata for additional languages, security information, or user-defined properties. For more information on advanced options, see "Import Advanced Options Dialog" on page 33-12.

Click **Show Summary** to view a summary of the export file contents. For more information on the export file contents, see "File Summary Dialog" on page 33-13.

After specifying the options on the Metadata Import dialog, click **Import** to import the metadata from the MDL file. The Metadata Import Progress dialog that indicates the

progress of the import is displayed. For more information about this dialog, see "Metadata Progress Dialog" on page 33-7.

> **Note:** If the MDL file that you selected for import was created using an earlier version of Warehouse Builder, clicking **Show Summary**, **Advanced**, or **Import** displays the Metadata Upgrade dialog. This dialog enables you to automatically upgrade the selected MDL file to the current version of Warehouse Builder. For more information about this dialog, see "Metadata Upgrade Dialog" on page 33-16.

File Name Type the name of the MDL file or click **Browse** to locate the MDL file you want to import.

Log File Type the name of the log file, along with the path, that will store diagnostic and statistical information about the import. You can also click **Browse** to locate the log file. For more information on log files, see "Metadata Loader Log File" on page 33-4.

Object Selection The Metadata Import Utility enables you to select the objects that you want to import from the MDL file. The Object Selection section contains the following options:

- **Import all objects from file**

  Select this option to import all objects contained in the export file.

- **Import selected objects from file**

  Select this option to import only some of the objects contained in the MDL file. Click **Select Objects** to choose the objects that you want to import. The Import Object Selection dialog is displayed. This dialog contains two sections: Available and Selected. The Available section contains the primary objects such as projects, modules, tables, views, connections that are specified in the MDL file. Expand the nodes in this section to view the objects they contain. When you select a node, all the objects that it contains are included in the import. For example, if you select a module node, all the objects contained in the module are imported. Use the shuttle buttons to move the selected objects from the Available section to the Selected section.

  The MDL file being imported can also contain administrative objects. To import these administrative objects, the user performing the import must have administrative privileges. If the user performing the import does not have the required privileges:

  – If the MDL file contains some administrative objects, the Available section of the Import Object Selection Page dialog does not displays these objects.

  – If the MDL file contains only administrative objects, the Import Utility displays an alert informing that the user does not have the required administrative privileges to perform the import.

Import Option Use the Import Option section to select the import mode. You can select one of the following options for the import mode:

- **Create new metadata only**

  This option adds new objects to a repository. It is referred to as the create mode.

- **Update metadata (replace existing objects and create new metadata)**

This option is referred to as the update mode. Selecting this option adds new objects to a repository and replaces existing objects with those in the MDL file being imported.

- **Merge metadata (merge existing objects and create new metadata)**

  When you select this option, the MDL adds new objects and overwrites existing objects in the repository only if they differ from those in the MDL file. This option is referred to as the merge mode. The merge mode does not delete existing objects.

  Note: You cannot import metadata using the Merge mode for mappings, pluggable mappings, and data auditors.

- **Replace existing objects only**

  Selecting this option replaces existing objects in your repository but does not add new objects. When importing metadata objects, the Metadata Import Utility overwrites any existing metadata when you use this mode. This mode is called the replace mode.

When you import metadata using the Update or the Replace modes, the import completely replaces the child objects of existing objects so that the final object is exactly the same as the source object. Any existing children of a repository object that are not replaced or added are deleted. This occurs regardless of whether a child object occurs in a mapping or is a foreign, primary, or unique key column in a table or view.

For example, in the MDL export file, the CUST table contains three columns with the physical names: `last_name`, `first_name`, and `middle_init`. In the repository, the same table already exists, and contains four columns with the physical names: `last_name`, `first_name`, `status`, and `license_ID`. During a replace operation, the columns `last_name` and `first_name` are replaced, column `middle_init` is added, and column `status` and `license_ID` are deleted. The final result is that the CUST table in the Warehouse Builder repository contains the same metadata from the CUST table in the export file.

Tip: Using the replace and update modes can result in lost data constraints, metadata physical property settings, data loading properties, and mapping attribute connections. If you choose to use replace or update modes, ensure that you can restore your repository, from a backup, to that state it was in prior to importing metadata in replace or update mode.

Match By

When you use the metadata import utility, it first searches the repository for metadata objects that exist in the repository and compares them to those in the file you are importing. To compare metadata in the import file with the existing repository metadata, it uses the matching criteria. How the comparison is made is determined by the import mode and by the search method you choose.

The Match By section provides the following options for matching criteria:

- **Universal Identifier:** Searches your repository using the Universal Object Identifiers (UOIDs) of the objects you are importing. The Metadata Import Utility uses these UOIDs to determine whether an object needs to be created, replaced, or merged during the import operation. Use this method if you want to maintain

UOIDs across different repositories even when object names in the target repository have changed.

- **Names:** Searches your repository using the names of the objects that you are importing. Physical names are exported to the export file. The physical name determines whether an object needs to be created, replaced, or merged during an import operation. Use this method when object names in the target repository change, and you want to create new UOIDs for those objects.

  By default, the import utility searches by UOIDs.

Note: MDL import does not support merging existing mappings.

Import Advanced Options Dialog

Use the Import Advanced Options dialog to import any of the following:

- Additional language metadata
- User-defined properties
- Security information such as object privileges and role assignments to users

The Import Advanced Options dialog contains the following sections: Languages and Administration.

Languages

The Base Language displays the base language of the repository. By default, Warehouse Builder imports data in the base language.

You can specify additional languages to import. The Metadata Import Utility imports the translations of the object for business name and description. The Available Languages list displays the list of languages that are specified in the MDL file. For example, the MDL file contains the additional languages French, German, and Spanish. But your repository contains only Spanish and German as the additional languages. Then the Available Languages list displays only Spanish and German. Select the language you want to import and click the arrow to move the language to the Selected Languages list. You can choose multiple languages at the same time by holding down the Ctrl or Shift key while making your selection. For more information on importing metadata in additional languages, see "Import Different Base Languages" on page 33-14.

Administration

This option is available only if you have administrator privileges and the metadata exists in the MDL file being imported. This section enables you to import the following additional metadata:

- **User-defined definitions:** To import the user-defined definitions for the objects that contain any user-defined properties, select the **Import User-defined Definitions** option.
- **Security Grants:** Select **Import security information** to import security information such as object privileges and role assignments made to users.

If the MDL file contains any of these objects, then you can import this additional metadata.

When you import and MDL file into a new repository, if you want to inherit the security information from the old repository, you must import the security information

before you import other objects. To do this you need to be connected to the repository as a user with Warehouse Builder administrator privileges.

After you make your selections on the Import Advanced Options dialog, click **OK** to save your selections and return to the Metadata Import dialog.

Name Conflicts

Name conflicts can occur in one of the following cases:

- A different object with the same name already exists in the target repository.
- A different object with the same business name already exists in the target repository.
- A different object with the same UOID already exists in the repository

When a name conflict occurs, the MDL reports an error and aborts the import.

File Summary Dialog

The File Summary dialog contains a brief summary of the contents of the export file. The information on this page is divided into the following sections: File, Administration, and Statistics.

File

The File section contains the name of the data file, the creation timestamp, the name of the export user, the repository connection information, the version of the Warehouse Builder Design Center used for the export, and annotations.

Administration

The Administration section contains information about the users and roles. It also lists the following details:

- Base language of the export file
- Additional languages in the export file
- Whether security information were included in the export file
- Whether user-defined definitions were included in the export file

Statistics

The Statistics section contains details about the types of objects contained in the export file and the number of objects of each type.

Combining Import Modes and Matching Criteria

Each search method used as matching criteria can be combined with an import mode in several different combinations. Each combination can offer different results in the import process. The mode that you select determines how the metadata import utility will search for metadata objects in the repository prior to importing.

For example, if the search is by the name of a repository object in the export file, the Metadata Import Utility searches the repository for the object's name. If an object with the corresponding name is not found, the resulting actions are based on the import mode you select.

Table 33–1 describes what happens in the available import modes for repository objects that do not match the object names.

Table 33–1 Import Mode without Matching Names

Import Mode	Result
Create Mode	A new object is created.
Replace Mode	A warning message is written to the log file that the object cannot be replaced because it does not exist in the repository. The object is skipped.
Update Mode	A new object is created.
Merge Mode	A new object is created.

Table 33–2 describes what happens in the available import modes for repository objects that match the object names.

Table 33–2 Import Mode with Matching Names

Import Mode	Result
Create Mode	A message is written to the log file that the object already exists and the object is skipped.
Replace Mode	The object is replaced.
Update Mode	The object is replaced.
Merge Mode	The object is merged.

The MDL reads and processes the imported metadata and writes status and diagnostic information in the log file.

Import Different Base Languages

When you import metadata in multiple languages, the language settings in the target repository can be different from the language settings in the export file. For example, the target repository can have the base language as English and additional language as French and German. But the export file can have the base language as French and additional language as English and German. This section describes how the MDL handles these conditions.

Base Language of the MDL Import File Different From the Base Language of the Target Repository

When you import metadata, MDL compares the ISO identification of the base language in the import file with the ISO identification of the base language of the target repository. The ISO identification consists of the language ID followed by the locale, in the format *language_locale*. For example, en_US is American English and fr_FR is French.

If the base ISO identification languages are different, MDL displays a warning dialog informing you that the base languages are different and warns you that Oracle recommends that you import metadata with the same character set and base language. You have the option to continue with the import. Click **Yes** to continue with the import. Click **No** to cancel the import.

WARNING: Under certain circumstances, continuing to import metadata when the base languages are different could mean corruption of the metadata being imported.

It is recommended that you move metadata between repositories with the same character set and base languages.

If the base ISO identification languages are the same, but the locales are different, the Metadata Import Utility displays a warning dialog asking if you want to continue with the import. For example, the export file contains English and the base language of the repository is American English. Click **Yes** to import metadata. Click **No** to cancel the import.

Importing Supported Languages

During an import, MDL checks if the additional languages in the import file exist in the target repository. If the import file contains additional languages that do not exist in the target repository, and you specify that these additional languages are to be imported, the Metadata Import utility writes a warning message in the MDL log file stating that the additional languages are not installed in the repository.

Validation Rules Governing Import

When you import a set of definitions from exported metadata, the import utility can update existing definitions in a Warehouse Builder project. However, certain metadata definitions require attention to ensure that they are updated. The following are examples of some of the errors you can see:

- **Mapping Definitions.** The Metadata Import Utility will bind imported mapping operators to their physical objects if the associated objects exist in the repository. However, if the associated physical objects do not exist in the repository, the imported mapping operators are unbound. The Metadata Import Utility writes a warning message in the log file stating that the mapping operators are not bound. You must then synchronize the new mapping operators with the physical objects they represent.
- **Foreign Key Definitions.** It is possible that a source MDL file can contain foreign key references to unique or primary keys that are not in the target repository. If the referenced unique or primary keys for any foreign key appearing in the MDL file does not exist in the target repository, the MDL Import Utility writes a warning message in the log file. This message will state that the repository does not contain a referenced key for the foreign key.

Upgrading Metadata from Previous Versions

While importing metadata, the Metadata Import Utility automatically upgrades metadata created using previous versions of Warehouse Builder to Oracle Warehouse Builder 10*g* Release 2. You do not need to manually upgrade your metadata from a previous version of Warehouse Builder.

When you import an MDL file into Oracle Warehouse Builder 10*g* Release 2, Warehouse Builder detects the version used to create the file. If the MDL file was created using a version earlier than Oracle Warehouse Builder 10*g* Release 2, the Metadata Upgrade dialog is displayed. This dialog enables you to upgrade the MDL file to the current version. For more information on the contents of this dialog, see "Metadata Upgrade Dialog" on page 33-16.

If you import an .mdl file containing metadata for gateway Modules, such as DB2 or Informix, from an older version of Warehouse Builder, the file may not import the metadata into the corresponding source module folders in a project. The imported files are stored under the Others node in the Project Explorer. You need to manually copy the metadata for the gateway modules into the correct source module folders.

The production versions of Warehouse Builder from which metadata is automatically upgraded to Oracle Warehouse Builder 10*g* Release 2 are as follows:

- Oracle Warehouse Builder 2.0.4 and 2.0.5
- Oracle Warehouse Builder 2.1.1
- Oracle Warehouse Builder 3.0 and 3.1
- Oracle Warehouse Builder 9.0.2, 9.0.3, and 9.0.4
- Oracle Warehouse Builder 9.2
- Oracle Warehouse Builder 10*g* Release 1
- Oracle Warehouse Builder 10*g* Release 2

Metadata Upgrade Dialog

This dialog is displayed automatically when Warehouse Builder detects that an MDL file being imported was created using a version prior to Oracle Warehouse Builder 10*g* Release 2. Use the **File Name** field to specify the name of the file that stores the upgraded MDL file. You can also click **Browse** to locate a directory or MDL file.

Click **Upgrade** to upgrade the MDL file to the current version of Warehouse Builder. After the Upgrade completes, the Metadata Upgrade dialog is closed. Click **Cancel** if you do not want to upgrade the MDL file.

Changes to Repository Objects After Upgrading to Oracle Warehouse Builder 10*g* Release 2

When you upgrade from previous versions of Warehouse Builder to the current version of Warehouse Builder, the upgrade utility makes the following changes to objects in the repository:

- **My Project:** The sample project that is prepackaged in Warehouse Builder is renamed from My Project to `MY_PROJECT` to comply with physical name requirements.
- **External Processes:** External processes are upgraded to external process activities in a process flow. If you defined an external process in a mapping in a previous release, MDL Upgrade Utility redefines the object as an external process in a process flow.
- **Business Areas:** Business areas are upgraded to Collections. If you defined a business area in a previous release, the MDL File Upgrade Utility prefixes the module name to the business area name and redefines it as a collection. For example, a business area named `ORDERS` in a module named `REGION1` is upgraded to a collection named `REGION1_ORDERS`.
- **External process mappings:** External process mappings will be migrated to process flows.
- **Dimension and Cube Mapping Operators:** The mapping operators for dimensions and cubes are converted to table operators. These table operators use the physical tables created by the MDL Upgrade Utility for dimensions and cubes.

- **Dimensions:** An associated dimension table is created with the same name as the dimension. The table contains the columns, constraints, and attribute sets defined in the Dimension Editor Table Properties of the dimension in the previous release.
- **Mapping Display Sets for Dimension Hierarchies:** Any mapping sets originally created based on the named attribute set for a dimension hierarchy are removed. This is because Warehouse Builder no longer automatically creates and maintains display sets for dimension hierarchies.
- **Dimension Attributes:** For each level attribute upgraded, a dimension attribute with the same name is created, if it does not already exist in the dimension.
- **Cubes:** An associated cube table is created with the same name as the cube. The cube table contains columns, constraints, and attribute sets defined in the Cube Editor Table Properties of the cube in the previous release.
- **Cube Dimension Reference:** If a foreign key in the cube does not reference a unique key in the lowest level of a dimension, the dimension reference is not imported. A warning is written to the import log file.
- **Intelligence Objects and Reports:** In the previous release, intelligence objects and reports were available only using OMBPlus scripting. These objects are not upgraded.
- **Locations and Runtime Repository Connections:** Locations and runtime repository connections are moved out of the projects that own them so that they can be shared across the entire repository. Thus the statistics in the import log file will display an additional project for these objects.
- **Control Centers and Locations:** After an upgrade, there is no association between the locations and the control centers that they reference. You must review the control center details using the Edit Control Center dialog and select the locations associated with this control center.
- **Advanced Queues:** An associated queue table is created based on the property AQ queue table name. The queue table created by the MDL File Upgrade Utility contains a column whose data type is the object type for that advanced queue.
- **Advanced Queue Operator in a Mapping:** Mapping Advanced Queue operators are changed to contain only one attribute called PAYLOAD. For Mapping Advanced Queue operators that are used as a source, a new Expand operator is added after the Mapping Advanced Queue operator. For Mapping Advanced Queue operators that are used as a target, a new Construct operator is added before the Mapping Advanced Queue operator.
- **Mapping Operator Names:** The MDL Upgrade Utility ensures that the physical names and business names of all mapping operators are unique.
- **MIV Objects:** MIV objects are not upgraded.

The following warnings may appear as a result of standalone transformations from previous releases no longer being an independent function category, but a part of the Warehouse module:

- Transformation name renamed from "old name" to "new name".
- Transformation business name renamed from "old name" to "new name".

Upgrading from Oracle Warehouse Builder 2.1.1 Be aware of the following if you are upgrading from Oracle Warehouse Builder 2.1.1:

- SQL*Loader mappings with variables and parameters are converted to group attributes of the Data Generator stage component.

- Control characters in descriptions are replaced with spaces.
- Configuration names in free format in Warehouse Builder 2.1 (for example, indexes) are changed to conform to the database naming standards.
- All Warehouse Builder 2.1.1 file and data warehouse objects are upper-cased. All Oracle modules remain unchanged, except for modules whose names are upper-cased.
- Index columns must reference an actual entity, otherwise the upgrade generates a list of columns that do not satisfy this requirement. If you do not fix this error before upgrading, the upgraded file will fail on import into Warehouse Builder.
- All local variables are converted to global variables.
- DDL mappings are not upgraded because they are not supported in Warehouse Builder.

  Granular exported mappings have upgrade limitations. For example, if a mapping is used as a source by Fact 1 and Fact 2, then the order of the upgrade is as follows:

  1. Fact 1
  2. Fact 2
  3. Mapping

Checking for Warnings and Error Messages

After upgrading the metadata, check the log file for warnings and errors.

- If you receive warnings during the upgrade, the upgrade utility completes and logs the warnings. If you receive errors, the upgrade utility terminates and logs the errors.
- If warnings and errors are shown after an upgrade, search for the words Warning and Error in the log file to determine the problem.
- If an unexpected error occurs and the upgrade terminates, the log file contains the details. Check your log file or contact Oracle Support.

Using Metadata Loader with OMB Plus

Warehouse Builder provides a scripting language called OMB Plus that enables you to create, modify, delete, and retrieve object metadata from the repository. For more information on using OMB Plus and its commands, refer to the Oracle Warehouse Builder API and Scripting Reference.

You can import and export repository metadata using OMB Plus commands. The control file and its parameters are the same as were used when exporting or importing metadata using the previously available command-line interface.

Note that the MDL command-line interface is no longer supported in Oracle Warehouse Builder 10*g* Release 2 (10.2.0.2).

34

Extending the Warehouse Builder Repository

This chapter describes how to extend the Warehouse Builder repository by creating custom objects and custom properties. This chapter includes the following topics:

Extending the Warehouse Builder Repository With User Defined Objects

As a Warehouse Builder administrator, you may encounter scenarios that require you to add objects or properties to the existing Warehouse Builder repository. Any objects you add to the repository are known as user defined objects (UDOs) and any properties you add are known as user defined properties (UDPs).

For example, as you use Warehouse Builder in conjunction with other applications, you may want to document how the data and metadata you manage in Warehouse Builder interacts with other applications. To facilitate this documentation, you can introduce new metadata objects into the Warehouse Builder repository and associate those objects with existing Warehouse Builder objects. Warehouse Builder displays these custom objects in the Design Center with the icon of your choice, provides a basic editor, and includes the objects in lineage and impact analysis reports.

Administration users can extend the Warehouse Builder repository by adding new properties or by adding new objects.

- **Adding New Properties to Warehouse Builder Objects**: Each Warehouse Builder object has a pre-defined property set to which you can add UDPs. Consider doing this, if, for example, you want to add a property to contain design notes for metadata objects. Once you define them, UDPs behave like native properties.
- **Adding UDOs to the Repository**: You can introduce new objects to the repository by defining UDOs which follow Warehouse Builder rules for object locking, multiuser access, transactions, security, and snapshots. You can also import and export UDOs and UDPs using the Metadata Loader (MDL).

For the sake of clarity, this chapter refers to the objects native to the Warehouse Builder repository as *Warehouse Builder objects*. Any objects you introduce to the repository are UDOs and any new properties are UDPs.

About Oracle Metabase (OMB) Plus

To define custom objects and properties in Warehouse Builder, you must use the scripting utility Oracle Metabase (OMB) Plus. After you specify the UDOs and UDPs through scripting, you can then create instances of those objects either through scripting or using the graphical user interface (GUI).

OMB Plus is the flexible, high-level command line metadata access tool for Oracle Warehouse Builder. It is an extension of the Tcl programming language and you can use it on UNIX and Windows platforms. With Using OMB Plus Scripts to Specify UDOs and UDPs, you can write the syntactic constructs such as variable support, conditional and looping control structures, error handling, and standard library procedures. Using OMB Plus, you can navigate repositories and manage and manipulate metadata in repositories.

To supplement the information in these sections on using OMB Plus to specify UDOs and UDPs, please refer to the Oracle Warehouse Builder API and Scripting Reference.

Using OMB Plus Scripts to Specify UDOs and UDPs

OMB Plus scripts enable you to define new objects, add and remove properties, as well as view the attributes for existing objects. The syntax is case sensitive and must be in upper case. When you create UDOs and UDPs, follow the "Naming Conventions for UDOs and UDPs" on page 34-3.

OMBDEFINE

OMBDEFINE CLASS_DEFINITION enables you to create new objects in the repository.

To create a new module object, use the following command. This creates a new module called UD_TABLEMODULE:

```
OMBDEFINE MODULE CLASS_DEFINITION 'UD_TABLEMODULE' SET PROPERTIES (BUSINESS_NAME,
PLURAL_NAME) VALUES ('Table Module', 'Table Modules')
```

OMBREDEFINE

OMBREDEFINE CLASS_DEFINITION enables you to redefine Warehouse Builder objects to include custom properties.

To create a UDP on the Dimension object, issue the following statement. This adds a new property called UD_DOCID to class definition DIMENSION:

```
OMBREDEFINE CLASS_DEFINITION 'DIMENSION'
     ADD PROPERTY_DEFINITION 'UD_DOCID' SET PROPERTIES (TYPE, DEFAULT_VALUE)
     VALUES ('INTEGER', '100')
```

The following command adds a new property for notes for the COLUMN type. Since columns exist in tables, views, materialized view, external table and sequences, the following command adds the property to columns for all of those metadata objects:

```
OMBREDEFINE CLASS_DEFINITION 'COLUMN'
     ADD PROPERTY_DEFINITION 'UD_COL_NOTE' SET PROPERTIES (TYPE, DEFAULT_VALUE)
     VALUES ('STRING', 'notes')
```

When you create and save a new property, OMB Plus performs the following validations:

- A user access check ensures that you have single-user access to the entire repository.

- A name space check ensures that you did not define two identically named properties within the same class hierarchy.
- A property value check ensures that you defined default values consistent with the data types you specified.

To change the name or the default value of a given property, issue a command like the following:

```
OMBREDEFINE CLASS_DEFINITION 'TABLE' MODIFY PROPERTY_DEFINITION 'UD_TBL_NOTE'
     SET PROPERTIES (DEFAULT_VALUE, BUSINESS_NAME)
     VALUES ('99', 'Table Note')
```

To delete a UDP, issue a command such as

```
OMBREDEFINE CLASS_DEFINITION 'TABLE' DELETE PROPERTY_DEFINITION 'UD_TBL_NOTE'
```

which deletes the UD_TBL_NOTE property from the Table class. Deleting a UDP is a destructive and a highly deprecated action because it cannot be undone. It renders irretrievable all custom property values made for this property definition in your repository.

OMBDESCRIBE

You can use OMBDESCRIBE on any class definition to view the attributes for a metadata element. Among other tasks, use OMBDESCRIBE to list the UDPs for a given object type. For instance, the following command lists the UDPs for dimensions:

```
OMBDESCRIBE CLASS_DEFINITION 'DIMENSION' GET PROPERTY_DEFINITIONS
```

You can also use OMBDESCRIBE to inspect the properties of a property definition. For instance, for a UDP called `UD_DOCID`, you can know its data type, default value, and business name with the following command:

```
OMBDESCRIBE CLASS_DEFINITION 'DIMENSION' PROPERTY_DEFINITION  'UD_DOCID'
     GET PROPERTIES (TYPE, DEFAULT_VALUE, BUSINESS_NAME)
```

Naming Conventions for UDOs and UDPs

It is mandatory to include the prefix UD_ while naming UDOs and UDPs. This ensures that the names of UDOs and UDPs are not identical to the names of predefined objects in the repository.

> **Note:** UDPs named with the prefix UDP_ are still valid. However, with the current release, it is preferred to use the prefix UD_ while naming UDOs as well as UDPs.

Adding New Properties to Warehouse Builder Objects

To define new properties on Warehouse Builder objects, complete the following steps:

1. Carefully plan the new additions to the repository.

   If possible, you should define all user-defined properties into the Warehouse Builder repository before allowing end users to access it. In doing so, you avoid the task of supplying values for UDPs on existing objects.

2. Log in to Warehouse Builder as the repository owner and in single user mode.

If another user is logged on through the GUI or OMB Plus, Warehouse Builder prevents you from running commands that alter the structure of the repository.

If already logged into the Warehouse Builder Design Center, you can launch OMB Plus from Window on the main menu. To ensure that no other users access the repository, connect to the repository and issue the command OMBSWITCHMODE SINGLE_USER_MODE.

3. Use the command OMBREDEFINE on the Warehouse Builder object you want to add a custom property.

   For examples on how to use this command, see the section OMBREDEFINE on page 34-2.

4. To view the changes in OMB Plus, use the command OMBDESCRIBE.

5. Once you create the UDP using scripting, users can view and assign values to the new property in the GUI.

6. Notify users that they can log in to Warehouse Builder.

Creating UDPs: An Example

Complete the following steps to create UDPs for an object:

1. Log in to the Warehouse Builder client as an administrator.

2. Open the OMB Plus client.

3. Ensure that you are in the single user mode. You can verify this with the command OMBDISPLAYCURRENTMODE. If you are in multiple user mode, switch to single user mode by using the command.

   ```
   OMBSWITCHMODE SINGLE_USER_MODE
   ```

4. In the OMB Plus client, type the following to create four UDPs for the object View:

   ```
   OMBREDEFINE CLASS_DEFINITION 'VIEW' \
   ADD PROPERTY_DEFINITION 'UDP_OWNER' SET PROPERTIES \
   (TYPE, DEFAULT_VALUE, BUSINESS_NAME) VALUES \
   ('STRING', 'REP_OWNER', 'Object Owner')

   OMBREDEFINE CLASS_DEFINITION 'VIEW' \
   ADD PROPERTY_DEFINITION 'UDP_FILE' SET PROPERTIES \
   (TYPE, DEFAULT_VALUE) VALUES ('FILE', 'C:\\vw.sql')

   OMBREDEFINE CLASS_DEFINITION 'VIEW' \
   ADD PROPERTY_DEFINITION 'UDP_LINK' SET PROPERTIES \
   (TYPE, DEFAULT_VALUE) VALUES ('URL', 'http://www.oracle.com')

   OMBREDEFINE CLASS_DEFINITION 'VIEW' \
   ADD PROPERTY_DEFINITION 'UDP_VERSION' SET PROPERTIES \
   (TYPE, DEFAULT_VALUE) VALUES ('DATE', '2006-1-7')

   OMBSAVE
   ```

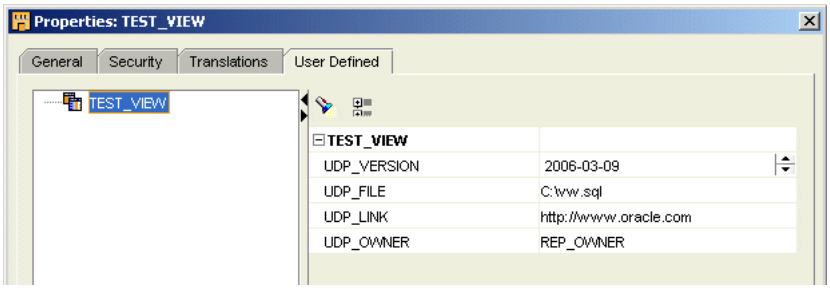
   This creates the following UDPs: UDP_OWNER, UDP_FILE, UDP_LINK, and UDP_VERSION.

Note: The valid UDP types are integer, string, float, double, date, timestamp, boolean, long, file, and url.

---

5. From the Project Explorer create a view in any module.
6. Open the property inspector for the view by right-clicking the view and selecting **Properties**.
7. Click the User Defined tab and select the view in the left-hand panel. The newly created UDPs are visible as shown in Figure 34–1.

Figure 34–1 UDPs for TEST_VIEW



Note: You can modify the values of any of the UDPs from the GUI.

To remove a UDP from the repository, use the DELETE clause. For example, to delete UDP_VERSION, use the following command:

```
OMBREDEFINE CLASS_DEFINITION 'VIEW' DELETE PROPERTY_DEFINITION 'UDP_VERSION'
```

Adding UDOs to the Repository

UDOs are objects that you define and add to the repository in addition to existing Warehouse Builder objects.

A UDO consists of a primary object called the module. This module acts as the topmost container holding other objects within. A module can contain folders, first class objects (FCOs), and second class objects (SCOs). Similarly, a folder can contain other folders, FCOs, and SCOs, while an FCO can contain one or more SCOs.

The objects in a UDO exhibit a parent-child relationship. The module is the parent. An FCO within a module is a child element of the module. Similarly, an SCO within an FCO is a child element of the FCO. For example, an Oracle module in the Warehouse Builder is a parent module. A table within this module is an FCO and a column within the table is an SCO.

To define new objects for the repository, complete the following steps:

1. Carefully plan the new additions to the repository.

   Before you begin, fully review the remainder of this chapter and become familiar with the necessary scripting commands.
2. Log in to Warehouse Builder as an administrator and in single user mode.
3. Design the UDO based on the steps described in "Writing Scripts to Define UDOs".
4. Use the OMBDEFINE command to create a new module, and FCOs and SCOs within that module. Use the OMBREDEFINE command to set the properties for these objects.

   Once you create the UDO using scripting, you can use the GUI to create and edit the objects it contains.

5. Log in to the Warehouse Builder GUI and view the new objects as described in "Working with UDOs and UDPs" on page 34-11.

   Verify that the new objects display as intended.

6. (Optional) Assign a new icon to the user defined object as described in"Creating New Icons for Objects in Warehouse Builder" on page 34-13.

7. Notify users that they can log in to Warehouse Builder.

Writing Scripts to Define UDOs

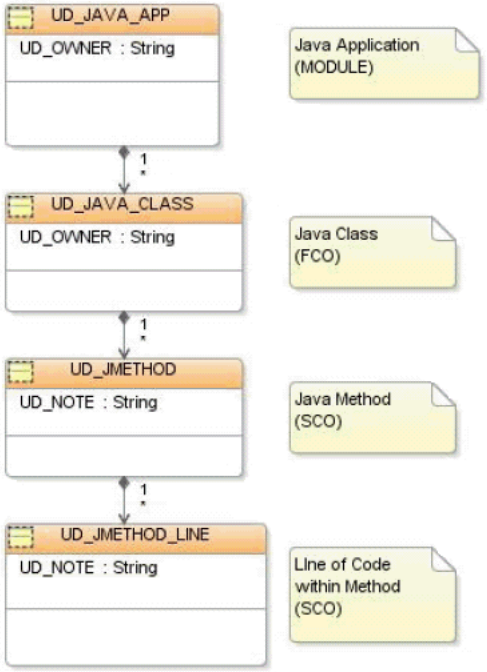
To define a UDO, write a script that completes the following steps:

- **Create a user defined module**: This will be the parent module.
- **Define the object type**: Define the module as a folder that can contain other objects.
- **Define FCOs and SCOs**: Create FCOs and SCOs for the UDO, and assign physical names to these objects. For example, `UD_WORKBOOK` is a valid physical name. You can also indicate a business name and plural name, both of which Warehouse Builder displays in the Design Center and in editors. Continuing the previous example, `Workbook` and `Workbooks` are likely entries for the business name and plural name respectively. If you do not specify these values, Warehouse Builder defaults to the physical name.
- **Define object properties**: Set the properties for all the objects you create. Some properties, such as Name, Business_Name, Plural_Name, and Description, are assigned automatically to any newly created object.
- **Add component definition**: All parent objects need to be assigned a component definition. The child elements have to be added to the component definition. The component definition determines the lifetime of child elements. For example, a column cannot exist if the parent table is deleted.
- **Define association types**: Create association types to indicate the types of relationships a UDO can have with Warehouse Builder objects and other UDOs. You need to perform this step only if you want end users to later relate the UDO to specific instances of objects. For instance, in your script you could associate the UDO with tables and views. In the Design Center, end users could then relate instances of the UDO with specific tables and views. Warehouse Builder displays these relationships in impact and lineage analysis reports.
- **Assign icons (optional)**
- **Save the changes**

Creating UDOs: An Example

This section provides an example to create an UDO that is modeled on a Java application. The application is a module. This module contains classes (FCOs) and those classes contain methods (SCOs). Within a method, you can model the lines of code. From a business standpoint, this is of interest because a particular line of code in an application may be impacted by a change in a database table if it is used within a SQL (JDBC) statement. The structure of the UDO is as shown in Figure 34–2.

Figure 34–2 Structure of the UDO



To create the UDO, perform the following steps:

1. Log in to the Warehouse Builder client as an administrator and open the OMB Plus window. Make sure that you are logged in the single user mode.
2. First create a module class and set properties for this module:

   ```
   OMBDEFINE MODULE CLASS_DEFINITION 'UD_JAVA_APP' \
   SET PROPERTIES (BUSINESS_NAME, PLURAL_NAME) \
   VALUES ('Java Application', 'Java Applications')
   ```

   This defines the class and sets certain properties that all Warehouse Builder objects have. BUSINESS_NAME is the user-friendly name for an object. If the Naming mode preference for the Design Center is switched into Business mode, then the value set for BUSINESS_NAME is displayed for the object. PLURAL_NAME is the label that is used to show where multiple instances of an object are shown, such as the label used for a tree node in the Design Center that contains several instances of the object.
3. Now create a folder with the same name as the module so that the module assumes the role of a folder:

   ```
   OMBDEFINE FOLDER_DEFINITION 'UD_JAVA_APP'
   ```

4. Now create an FCO:

   ```
   OMBDEFINE FIRST_CLASS_OBJECT CLASS_DEFINITION \
   'UD_JCLASS' SET PROPERTIES (BUSINESS_NAME, PLURAL_NAME) \
   VALUES ('Java Class File', 'Java Class Files')
   ```

5. Add the FCO as a child of the folder class:

   ```
   OMBREDEFINE CLASS_DEFINITION 'UD_JAVA_APP' \
   ADD CHILD_TYPE 'UD_JCLASS'
   ```

6. Create a component definition for the FCO:

```
OMBDEFINE COMPONENT_DEFINITION 'UD_JCLASS'
```

7. Add the component definition to the folder definition:

```
OMBREDEFINE FOLDER_DEFINITION 'UD_JAVA_APP' \
ADD 'UD_JCLASS'
```

8. Create an SCO and set its properties:

```
OMBDEFINE SECOND_CLASS_OBJECT \
CLASS_DEFINITION 'UD_JMETHOD' \
SET PROPERTIES (BUSINESS_NAME, PLURAL_NAME) \
VALUES ('Method', 'Methods')
```

9. Add the SCO as a child of the FCO:

```
OMBREDEFINE CLASS_DEFINITION 'UD_JCLASS' \
ADD CHILD_TYPE 'UD_JMETHOD'
```

10. Add the SCO to the component definition:

```
OMBREDEFINE COMPONENT_DEFINITION 'UD_JCLASS' \
ADD 'UD_JMETHOD'
```

11. Create an SCO and set its properties:

```
OMBDEFINE SECOND_CLASS_OBJECT \
CLASS_DEFINITION 'UD_JMETHOD_LINE' \
SET PROPERTIES (BUSINESS_NAME, PLURAL_NAME) \
VALUES ('Java Method Line', 'Java Method Lines')
```

12. Add this SCO as a child of the initially created SCO:

```
OMBREDEFINE CLASS_DEFINITION 'UD_JMETHOD' \
ADD CHILD_TYPE 'UD_JMETHOD_LINE'
```

13. Add this SCO to the component definition:

```
OMBREDEFINE COMPONENT_DEFINITION 'UD_JCLASS' \
ADD 'UD_JMETHOD_LINE'
```

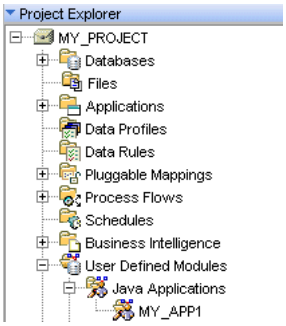
This creates an UDO with the following characteristics:

- A module folder called UD_JAVA_APP
- An FCO, UD_JCLASS, within the module
- An SCO, UD_JMETHOD, which is the child of UD_JCLASS
- An SCO, UD_JMETHOD_LINE, which is the child of UD_JMETHOD

You can access the UDO from the Project Explorer under the User Defined Modules icon as shown in Figure 34–3. To create a new instance of the UDO, right-click the UDO and select New. You can create new modules and SCOs as well as edit these modules and SCOs.

Note: For the newly created UDO to be visible in the Project Explorer, you must shut down Warehouse Builder completely, and then start it up again, saving any changes if prompted.

Figure 34–3 Accessing UDO from the Project Explorer



Associating UDOs with Objects

UDOs can be associated with other objects. By creating these associations, you can use UDOs just like any other object when doing Lineage and Impact Analysis.

Associating a Java Application with a Table

This example associates the SCO, UD_JMETHOD, with one or more tables. This is modeling the fact that a method could be referencing tables in JDBC calls.

To associate the Java method to table, use the command:

```
OMBDEFINE ASSOCIATION_DEFINITION 'UD_XJMETHOD2TABLE' \
SET PROPERTIES (CLASS_1,CLASS_2,ROLE_1,ROLE_2 \
,ROLE_1_MAX_CARDINALITY,ROLE_1_NAVIGABLE) \
VALUES ('UD_JMETHOD','TABLE','TABLEUSED','JM2TABLE' \
,'INFINITE','true') ADD DEPENDENCY_DEFINITION 'DATAFLOW'
```

CLASS_1 and CLASS_2 can be any classes (FCO or SCO). At least one of the classes should be a user defined class. The other class can be either a user defined class or one of the main Warehouse Builder classes, such as table or column. In this example, the association is between the UDO, UD_JMETHOD, and table.

Role_1 and Role_2 are the names you use to identify Class_1 from the point of view of this association. A class may have multiple associations and it plays a role in each one of them.

MAX_CARDINALITY enables you to limit the number of objects of that class which participate in that association. For example, consider the association between uniqueKey and foreignKey. The max_cardinality of uniqueKey is 1, because a given foreignKey object can be associated with at most one uniqueKey object. MAX_CARDINALITY can be set to any positive integer, or the reserved word INFINITE.

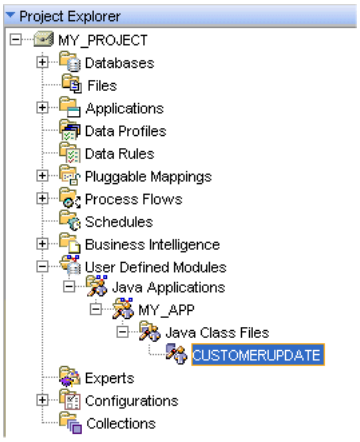
ROLE_1_NAVIGABLE is used by Lineage/Impact analyzer. If set to TRUE, it means that the analyzer can traverse the association in either direction between Class_1 and Class_2. If the property is set to FALSE, it means that the analyzer can traverse the relationship from Class_1 to Class_2, but not the other way around.

DEPENDENCY_DEFINITION is a mandatory parameter of an association and must always be set to DATAFLOW.

To associate the UDO to a table, complete the following steps:

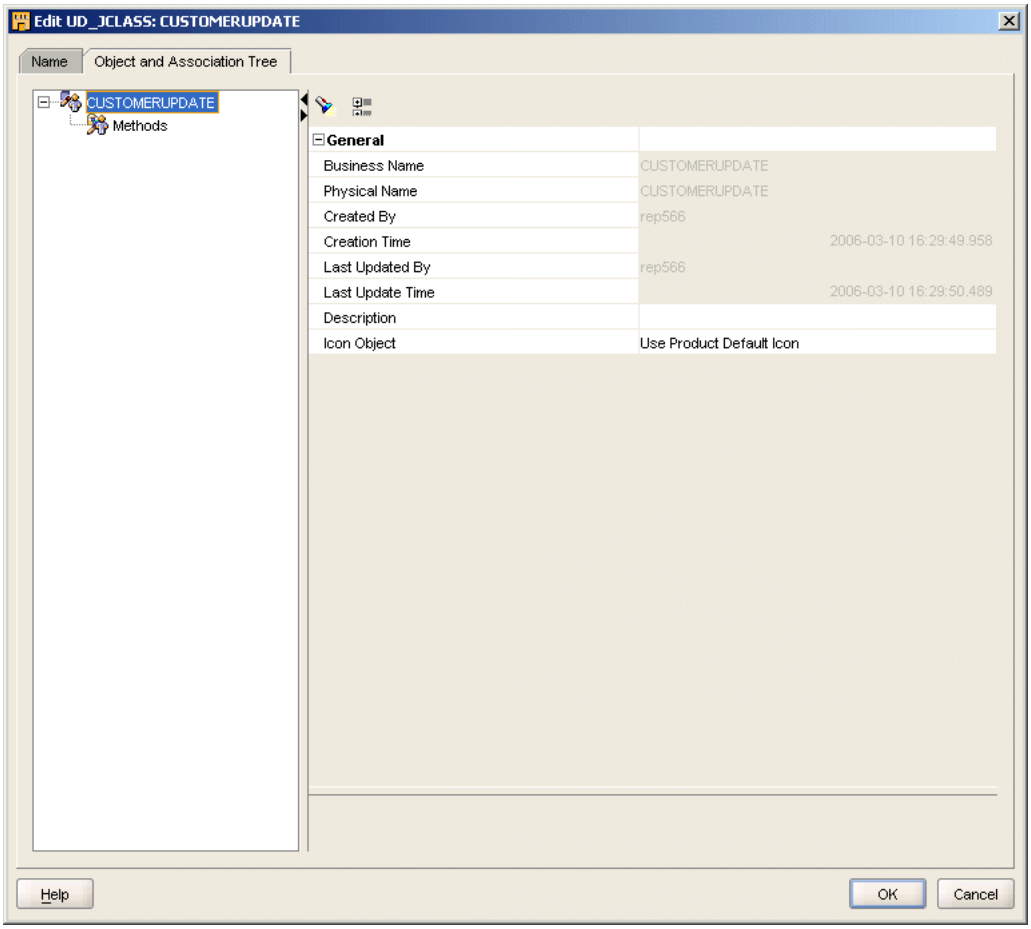
1. In the Project Explorer, expand the node User Defined Modules.
2. Right-click Java Applications and select **New**.
3. Specify a name for the application.

4. Right-click the node Java Class Files and select **New**.
5. Specify a name for the Java class.



6. Right-click the object you just created, and select **Open Editor** to open the UDO Editor.
7. Click the Object and Association Tree tab and select CUSTOMERUPDATE. You can see the properties for the FCO, UD_JCLASS, as shown in Figure 34–4.

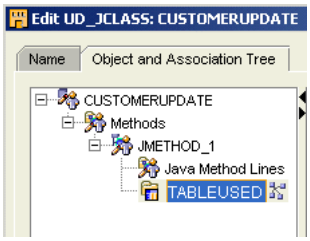
Figure 34–4 Editor for UD_JCLASS



8. Right-click **Methods** and select **Create**.

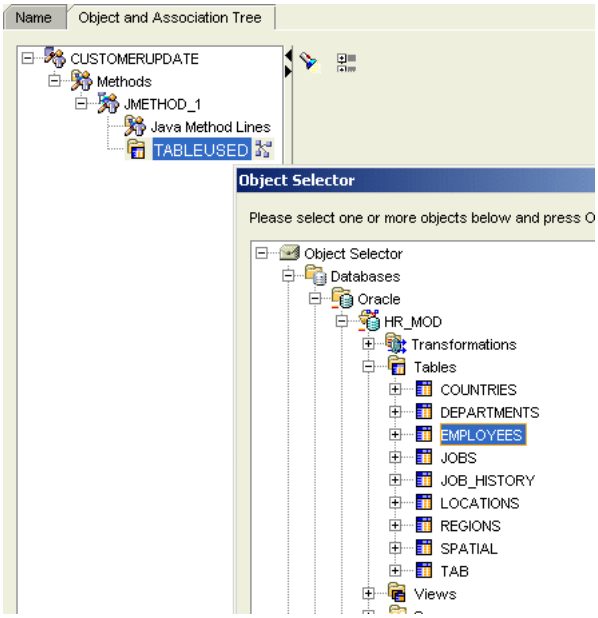
   An SCO called JMETHOD_1 is created.

9. JMETHOD_1 contains two nodes: Java Method Lines, which is the child SCO, and TABLEUSED, which is the value specified for ROLE_1 when the association UD_XJMETHOD2TABLE was created.



10. Right-click TABLEUSED and select **Reference**.

    The Object Selector dialog appears, and enables you to select the table to which you want to connect the UDO.



Working with UDOs and UDPs

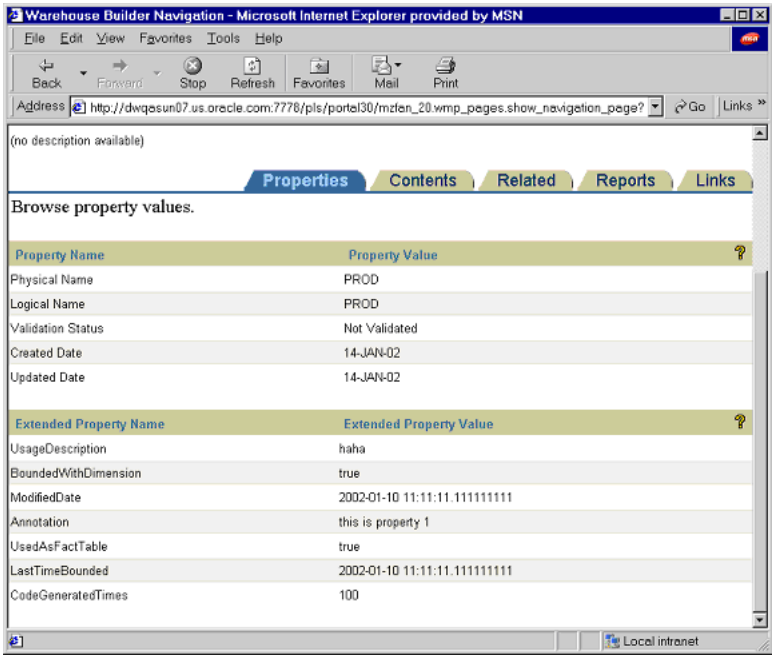
In the GUI, you can view UDOs and UPDs in the Project Explorer and in the Warehouse Builder Design Browser. However, in the Project Explorer, you can also edit the UDOs and UDPs.

Warehouse Builder Design Browser

Warehouse Builder Browser is a metadata management and reporting portal for Warehouse Builder. This browser displays objects, object properties, object relationships, including UDOs and UDPs. The browser also displays lineage and impact analysis reports for all objects including UDOs.

If you define a UDP for a given object, the browser lists the UDP name and values as Extended Property Name and Extended Property Value as shown in Figure 34–5.

Figure 34–5 Sample Properties Sheet with User-Defined Properties



Propagating UDOs and UDPs to Other Repositories

The primary method for propagating changes from one repository to another is using MDL. The MDL enables you to export and import the metadata definition of the UDP and its contents.

Exporting UDOs and UDPs

You can export UDOs and UDPs as any other object.

In the MDL Control file, the option is `DEFINITIONFILE=`*`filename`* to export the metadata definition. For example:

```
## Sample Export file
USERID=UserName/Password@HostName:PortID:OracleServiceName
#
DEFINITIONFILE=Drive:\DirectoryName\filename.mdd

FILE=Drive:\DirectoryName\filename.mdl
LOG=Drive:\DirectoryName\filename.log
```

Importing UDOs and UDPs

You can import UDPs from the command line only. During import, MDL updates the UDPs for all objects. In the MDL Control file, the option is DEFINITIONFILE=filename to import the metadata definition. For example:

```
## Sample Import file
USERID=UserName/Password@HostName:PortID:OracleServiceName
#
DEFINITIONFILE=Drive:\DirectoryName\filename.mdd

FILE=Drive:\DirectoryName\filename.mdl
LOG=Drive:\DirectoryName\filename.log
```

You can import UDPs using one of the following search criteria:

- **Universal ID**: The metadata definition contains a Universal Object ID (UOID). The UOID uniquely identifies objects across repositories. If you import the MDL file by UOID, then MDL looks up the metadata definition by UOID. If the metadata definition name in the source MDL file is different from the metadata definition in the repository, then MDL renames it.
- **Physical Name**: MDL looks up the metadata definition by physical name.

Regardless of the import mode, MDL either adds the metadata definition if it does not exist in the repository, or updates the metadata definition if it already exists. MDL will not delete metadata definitions in the repository.

When updating the metadata definition, MDL only renames the object if the names are different (search criteria is by UOID), and updates the default value. MDL does not change the data type.

Creating New Icons for Objects in Warehouse Builder

Icons are graphics that visually suggest the availability of a function or type of an object to end users. There are many types of objects pre-defined inside Warehouse Builder, each with their own icon. You may want to change the icon associated with an existing object or instance of an object to something more recognizable. For example, you could visually highlight a particular table by altering its icon. Additionally, for UDOs, you may want to change the default icon to something representative of the object. You can create your own icons using a graphics editor or third party software.

In the Design Center, you can associate icon sets with an instance of an object. Or, use the OMB Plus scripting language to associate icon sets at the object type level (and thus inherited by any instance of that object).

> **Note:** You can assign new icons to most Warehouse Builder objects with the exception of pre-defined objects like public transformations and public data rules and DEFAULT_CONFIGURATION, DEFAULT_CONTROL_CENTER, and OWB_REPOSITORY_LOCATION.

Every object in Warehouse Builder has a set of icons of varying sizes to represent it throughout the various editors and toolbars. Each icon set includes a canvas icon, palette icon, and a tree icon as described in Table 34–1. When you define a new icon set, follow the sizing guidelines. If you specify a new icon with an incorrect size, Warehouse Builder automatically resizes it, which may distort your intended design.

Table 34–1 Icon Sets

Type	Description
Canvas Icon	Represents instances of objects in the canvas of an editor. For example, it displays a table icon in the canvas of the Mapping Editor or in a Lineage Report. The correct size is 32 x 32 pixels in GIF or JPEG format
Palette Icon	Represents types of objects in editor palettes. For example, it displays the table operator in the Mapping Editor operator palette. The correct size is 18 x 18 pixels in GIF or JPEG format
Tree Icon	Represents types and instances of objects in navigation trees such as the Project Explorer in the Design Center. The correct size is 16 x 16 pixels in GIF or JPEG format

Creating Icon Sets

To create a new icon set, complete the following steps:

1. Log in to the Warehouse Builder client as an administrator.
2. In the Global Explorer, right-click the Icon Sets node and select **New**.
3. The Create Icon Set dialog is displayed. For details on the values to be entered on this page, see "Create Icon Set Dialog" on page 34-14.

Create Icon Set Dialog

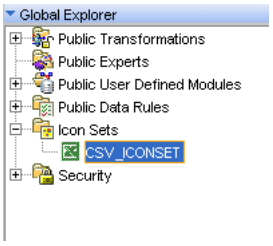
The Create Icon Set dialog enables you to specify values for the Icon Set. Enter the following values and click **OK** to define a new Icon Set:

- **Name**: The name of the Icon Set.
- **Group**: You can assign the Icon Set to any of the groups available in the list. This is useful when you are creating a large number of icon sets and want to organize them into different groups.
- **Description**: A description of the Icon Set.
- **File Name**: Navigate and select the image that you want to assign to the new Icon Set. You need to select an image for Canvas, Palette, and Tree Icon.

> **Note:** You can use any image of your choice to represent the new icon.

The newly created icon set will be available under Icon Sets in the Global Explorer as shown in Figure 34–6.

Figure 34–6 User Defined Icon



Assigning New Icon Sets to Warehouse Builder Objects

You can change the default icon setting for any object in Warehouse Builder by editing its properties. In any of the Explorers in the Design Center, right-click an object and select **Properties** to assign a new icon set to an object. When you save your changes, Warehouse Builder displays the new icon set throughout the user interface.

To revert back to the original icon, select Use Product Default Icon.

For more information on icon sets, see "Creating New Icons for Objects in Warehouse Builder" on page 34-13.

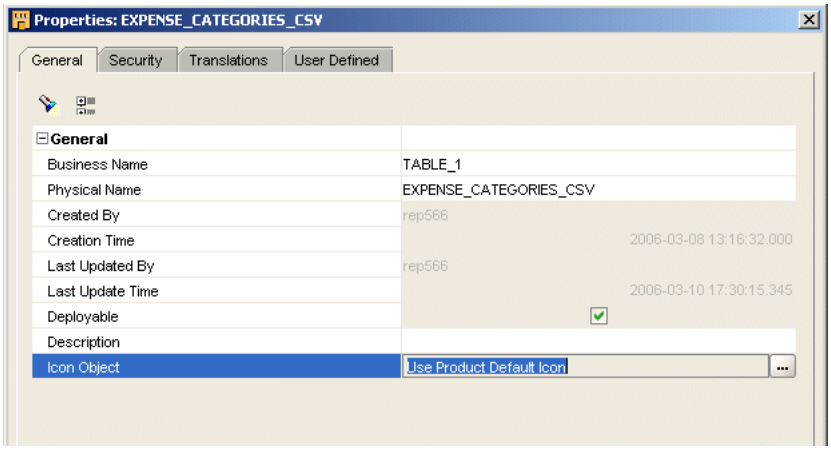
Assigning a New Icon Set to an Object

Complete the following steps to assign a newly created icon set, CSV_ICONSET, to an instance of an object.

1. Right click any instance of an object and select **Properties** to open the Property Inspector.
2. On the General tab, click the field **Use Product Default Icon**.

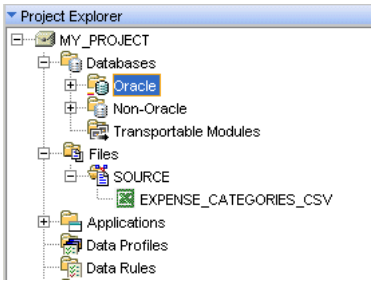
   An Ellipsis button is displayed in this field as shown in Figure 34–7.

Figure 34–7 Properties Inspector for EXPENSE_CATEGORIES_CSV



3. Click the Ellipsis button to open the Icon Object dialog.
4. Select CSV_ICONSET and click **OK**.
5. CSV_ICONSET is now assigned to the instance EXPENSE_CATEGORIES_CSV as shown in Figure 34–8.

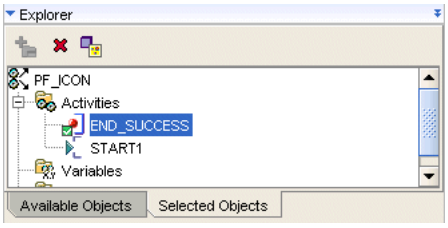
Figure 34–8 Icon Set for EXPENSE_CATEGORIES_CSV



Assigning New Icon Sets to Activities in Process Flow

You can assign new icon sets to the activities in a process flow. The Process Flow Editor contains the Select Icon button, which gets enabled when you select an activity, as shown in Figure 34–9.

Figure 34–9 The Select Icon button on the Process Flow Editor



To assign a new icon set to an activity:

1. Select the activity and click the Select Icon button. The Select Icon Set dialog displays.
2. Select an icon from the list and click **Select**.

The icon set is assigned to the activity.

For information on process flows, see Chapter 7, "Designing Process Flows".

Assigning New Icon Sets to Tasks in Expert Editor

You can assign new icon sets to the tasks in Expert Editor using the Select Icon button available on the Expert Editor. This is similar to assigning icon sets to activities in a process flow.

To assign a new icon set to a task:

1. Select the task and click the Select Icon button. The Select Icon Set dialog displays.
2. Select an icon from the list and click **Select**.

The icon set is assigned to the task.

For information on experts, see Oracle Warehouse Builder API and Scripting Reference.

Part VII

Additional Usages

This part contains the following chapters:

35

Implementing Best Practices

If you are a knowledgeable Warehouse Builder user, you can design solutions that simplify routine tasks and implement best practices. You can develop these solutions, called experts, in the Expert Editor.

This chapter contains the following topics:

What Are Experts?

Experts are solutions that advanced users develop to automate routine or complex tasks using best practices. To develop experts, you should have a working knowledge of Warehouse Builder, the Warehouse Builder scripting language, and Tool Command Language (Tcl), which is an open-source programming language.

For example, a common activity is extracting data from a flat file and loading that data into a table in Oracle Database. To accomplish this activity, users might take the following steps, in which they navigate a variety of user interfaces in Warehouse Builder:

1. Define a flat file module.
2. Identify the source file.
3. Specify the data format.
4. Define an external table.
5. Define an Oracle database module and location.
6. Define a mapping.

7. Validate, generate, and deploy all objects.
8. Execute the mapping.

To help users with this activity, you could design an expert that calls all the necessary user interfaces, provides customized instructions, and prompts users for input. In an expert, the steps are defined by **tasks** and the order of execution is defined by **transitions**.

Experts are reusable, shareable, and can access all areas of Warehouse Builder including user interfaces and the OMB Plus scripting language. Experts can also call Java programs.

See Also:

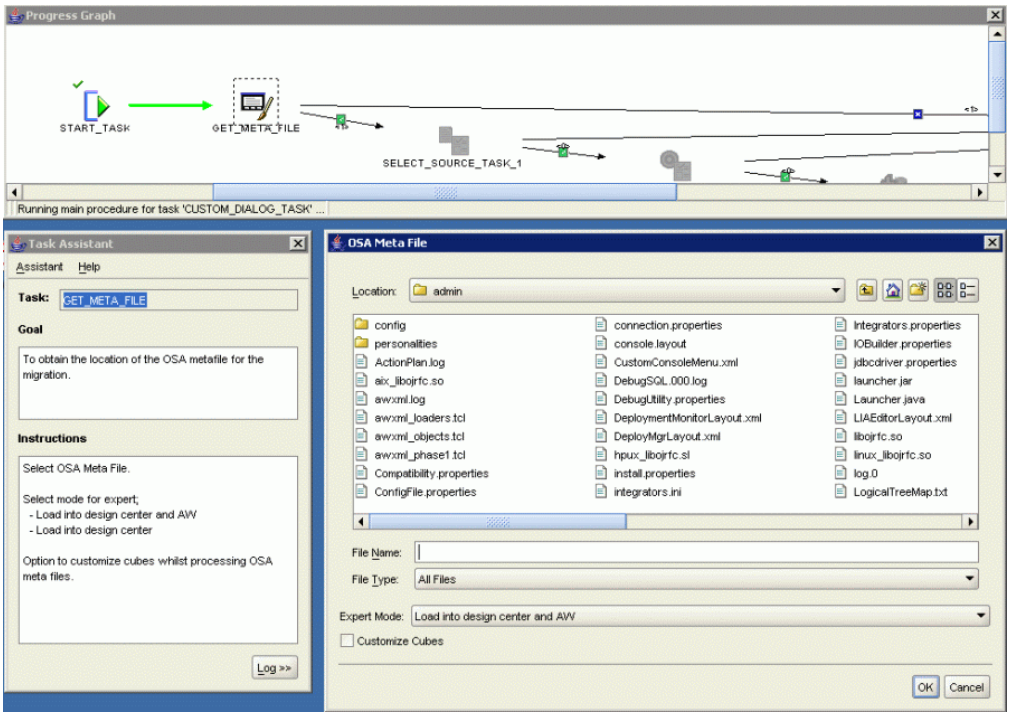
- *Oracle Warehouse Builder Scripting Reference* for information about the scripting language.
- The Tcl Developer Xchange web site at `http://www.tcl.tk` for information about Tcl.

User's View of an Expert

Figure 35–1 shows how an expert might look to a user. Three separate windows are displayed:

- **Progress Graph**: Displays the expert in the same format as it appears on the editing canvas, but the executed transitions and the current task are highlighted. This window serves the same purpose as bread-crumbs, tracking the user's progress through the expert. It can also be a helpful debugging tool. You can set a property on the expert that controls whether the progress graph is displayed or hidden.
- **Task Assistant**: Displays the name of the current task, its goal, and instructions for using it. You provide this information on the property sheets for the task as part of its definition. You can set a property on the expert that controls whether the task assistant is displayed or hidden.
- **Task Display**: Tasks that obtain information from the user display various types of graphical user interfaces. Among these tasks are those that display Warehouse Builder components, such as the Object Editor, and those that provide basic functions, like the file selector shown in the figure. In this example, the user selects a file, and the filename is passed as an input parameter to the next task. You can also store the value in a variable to use later.

Figure 35–1 Execution of an Expert

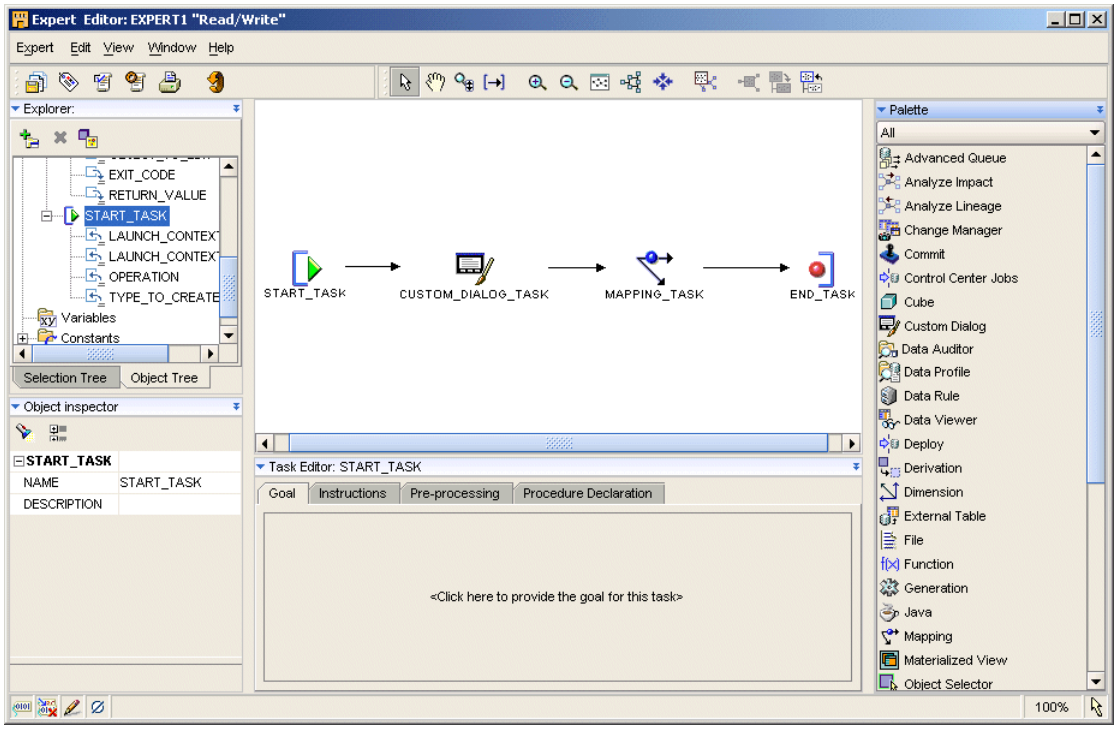


Developer's View of an Expert

The Expert Editor provides the canvas and the palette that you need to create, modify, and deploy experts.

Figure 35–2 shows the Expert Editor with a sample expert. Tasks appear as icons on the canvas. The arrows connecting the tasks are transitions. Transitions identify the order in which the tasks are executed.

Figure 35–2 Expert Editor



How Are Experts Different From Process Flows?

The Expert Editor is very similar to the Process Flow Editor. If you have created process flows, you will be able to adapt your knowledge very quickly to developing experts. However, there are important differences as well as important similarities.

- **Unit of Work**: In a process flow, the unit of work is an activity. In an expert, the basic unit of work is a task.
- **Transitions**: Both process flows and experts use transitions to connect the basic units of work. Transitions can be conditional.
- **Variables**: Both process flows and experts enable you to define local variables to pass values from one task to another.
- **End tasks**: Process flows have success, warning, and failure end activities, but experts have a single End task.
- **Subprocesses**: You can design a process flow to open other process flows, and you can design an expert to open other experts. In this use, they are called nested experts.
- **Code**: A process flow generates XML that conforms to the XPDL workflow standard. An expert generates Tcl.

How to Create Experts

Before you begin the process of developing an expert, you should compose a plan that answers the following questions

- What job do you want the expert to accomplish?

  For example, refresh the data in a schema.

- What are the steps that you would take in Warehouse Builder to accomplish this job?

  For example, identify the locations of the source data files and the target schema, execute a mapping or process flow, and so forth.

- How flexible do you want the expert to be? That is, where can users provide input into the expert?

  Experts can run without any user input at all, or they can prompt the user for input at every step. For example, an expert can refresh either a particular schema or the user's choice of a schema.

To define an expert, complete the following tasks:

1. Creating an Expert Object on page 35-5
2. Adding Tasks to the Expert Canvas on page 35-6
3. Adding Transitions to the Expert Canvas on page 35-7
4. Passing Data Values Among Tasks on page 35-8
5. Validating, Generating, and Starting Experts on page 35-9
6. Creating a Development Environment on page 35-10
7. Publishing Experts on page 35-10

Creating an Expert Object

Experts are organized in modules within a project.

> **Note:** Administrators can develop experts directly in the Global Explorer. Simply follow the steps below in the Global Explorer instead of in the Project Explorer. Administrators can also publish experts, as described in "Publishing Experts" on page 35-10.

To create an expert:

1. In the Project Explorer, right-click **Experts** and select **New** from the popup menu.

   The Create Expert Module dialog box is displayed.

2. Enter a name and description for the module.

   Click **Help** for additional information.

3. Select the **Proceed to Wizard** box.

   The Create Expert dialog box is displayed.

4. Enter a name and description for the module.

   The Expert Editor is displayed. You can begin developing the expert.

To edit an expert module:

1. In the Project Explorer, expand the Experts folder, then expand the module for the expert.
2. Double-click the expert.

   The Expert Editor is displayed.

Adding Tasks to the Expert Canvas

Tasks represent units of work for the expert. When you design an expert in Warehouse Builder, you select tasks from the Expert Editor palette and drag them onto the canvas. The Expert Editor initially has the Start and the End tasks already positioned on the canvas.

To add a task to an Expert:

1. Select a task from the palette and drag it onto the canvas.

   Or

   From the Expert menu, choose **Add**, then choose a category from the popup menu. Select a task from the category.

   The editor displays the task on the canvas.

2. In the **Task Editor**, complete the tabs for the various properties of that task.

   These properties vary according to the type of task. All tasks have the Goal, Instructions, Pre-processing, and Post-Processing properties. Some tasks such as the Custom Dialog and OMB tasks include additional properties.

   For information about these properties, click **Help** from the popup menu on the Task Editor title bar.

3. Use the Explorer and the Object Inspector to enter values for the parameters of the task.

   Tasks have input parameters and output parameters. Refer to "Passing Data Values Among Tasks" on page 35-8 for methods of providing values to the input parameters.

4. To change the name of the task or add a description, right-click the icon on the palette and select **Edit Details** from the popup menu.

   The Edit Task dialog box is displayed. Click **Help** for more information.

Adding Nested Experts to the Expert Canvas

You can create experts that perform relatively simple tasks, then use them as building blocks in developing more complex experts. When you add an expert to the canvas, it is called a **nested expert**. The nested expert functions as a single task, with one incoming transition and one or more outgoing transitions.

The existing flow becomes the **parent graph**, and the nested expert becomes a **child graph** on the canvas.

To add a nested expert:

1. From the Expert menu, select **Add**, then **Nested Experts**.

   Or

   Drag-and-drop experts from the Available Objects tab of the Explorer to the canvas.

   The Add Nested Experts dialog box is displayed.

2. Expand the module folders, then select one or more experts from the list. Click **OK**.

   The expert appears as a single icon on the canvas.

3. Draw transitions from a task to the expert, and from the expert to a task.

To view a nested expert:

The graphic toolbar provides icons for viewing the contents of a nested expert.

- To enlarge the expert icon so that you can see the individual tasks, click Expand Child Graph on the toolbar. This view of the nested expert is similar to the Bird's Eye View of the canvas.

  To shrink the nested expert back to an icon, click the arrow on the right side of its title bar.

- To fully examine the tasks and transitions in the nested expert, click Visit Child Graph on the toolbar. The canvas displays only the nested expert, and its properties are shown in the Explorer, Object Details, and Task Editor windows. You have read-only access to a nested expert.

  To shrink the nested icon back to a component in the larger expert, click Return to Parent Graph on the toolbar.

To edit a nested expert:

Open the nested expert in its own Expert Editor and make your changes. These changes are reflected in the parent expert as soon as you save the changes, with one exception: Changes to the expert parameters.

If you make changes to the parameters of the nested expert, you must delete and add the nested expert in the parent.

Adding Transitions to the Expert Canvas

Transitions indicate the sequence for executing the tasks in an expert. You can use conditional transitions to execute a task based on the completion state of the preceding task. By using conditional transitions, you can develop an expert that handles errors gracefully and provides alternative actions when users cancel a task.

A single task can have multiple outgoing transitions, but only one will be executed. If more than one transition evaluates to true, then only the first one is executed.

To connect two tasks with a transition:

1. Add one or more tasks to the canvas, following the steps in "Adding Tasks to the Expert Canvas" on page 35-6.
2. Click the Select tool on the toolbar.
3. Position the pointer over the previous task so that it has an arrow shape. Click and drag the pointer to the next task.

   Notice that a plus sign appears under the pointer, then changes to a circle as the pointer approaches a next task.

   The editor displays an arrow between the two tasks, assigns a default name to the transition, and displays the transition in the Explorer and in the Object Selector.
4. If you want execution of the next task to be conditional, then assign a condition to the transition in the Object Details window. Table 35–1 describes the conditions.
5. After you have connected all the tasks, click the Auto Layout tool to arrange the tasks in the order of execution and spaced evenly across the canvas.

Table 35–1 Types of Conditions for Transitions

Condition	Continuation to the Next Task
Success	Only if the preceding task ends in success.
Error	Only if the preceding task ends in error.
Cancel	Only if the user cancels the preceding task.
Complex Condition	Only if the conditions you specified are true.

Passing Data Values Among Tasks

Tasks have both input parameters and output parameters.

- Input parameters affect the execution of the task. You can provide these values, or you can design the expert so that the user provides them.
- Output parameters are returned by the task. These values are the result of the work performed by the task.

You can pass data values among tasks either by binding the parameters or by using constants and variables.

Binding Input Parameters

You can bind the input parameters of a task to the output parameter of another task, to a global constant, or to a global variable. Binding is the easiest method of passing values to input parameters.

To bind the parameters:

1. Create the source and the target tasks.
2. In the Expert Explorer, select an input parameter for the target task.
3. In the Object Details window, click the **Binding From** field and select an output parameter, a variable, or a constant from the popup menu.

Using Constants

You can use the following predefined constants in any task:

- `OMB_CURRENT_PROJECT`: Stores the name of the current project.
- `OMB_CURRENT_SESSION`: Stores the identifier of the current session.
- `OMB_CURRENT_USER`: Stores the user ID of the current user.

These constants appear in the Explorer under the Constants folder.

Using Variables

You can use custom variables to store the values of output parameters. This is particularly useful when using Custom Dialog tasks, because the `GUI_RETURN_VALUE` output parameter is an array that functions like a hash table. It must be parsed before the individual values can be passed to input parameters. You may also want to use variables when passing a single value to multiple tasks, or when passing a value further down the flow.

To create a variable:

1. In the Explorer, select the Variables folder.
2. Click the **Create** icon at the top of the Explorer.

   A variable named `VARIABLE` appears in the Variables folder.
3. Select `VARIABLE` and use the Object Details window to change its name, data type, and other parameters.

Or, you can use Tcl commands to declare variables in the Pre-processing or Post-processing tabs of the Task Editor.

To assign a value to a variable:

Use a Tcl variable assignment statement in the Pre-processing or Post-processing tabs of the Task Editor. The following are some examples:

This example assigns the value of the `RETURN_VALUE` output parameter to a variable named `THIS_OBJECT_NAME`.

```
set THIS_OBJECT_NAME $RETURN_VALUE;
```

The next example assigns the value of the first component (`COMP1`) of the `GUI_RETURN_VALUE` parameter to a variable named `THIS_TABLE`.

```
set THIS_TABLE $GUI_RETURN_VALUE(COMP1)
```

To use a variable:

You can bind an input parameter of a task to a variable. The variables appear in the popup menu in the Object Details window.

You can also use the variables in any Tcl commands on the Pre-processing and Post-processing tabs of any task, and on the Main tab of an OMB task.

Validating, Generating, and Starting Experts

You can validate, generate, and start an expert from the Expert Editor or from the Design Center.

To validate, generate, or start an expert from the Expert Editor:

From the Expert menu, choose the **Validate**, **Generate**, or **Start** command.

or

Click the Validate, Generate, or Start icon from the toolbar.

The Message tab of the Compilation Results window displays any validation errors and warnings. When you select a message, Warehouse Builder selects the errant task or transition on the canvas, the Explorer, and the Object Details window, so that you can correct the problem.

To validate, generate, or start an expert from the Design Center:

1. In the Project Explorer, expand the Experts folder for the project, then expand the module folder.
2. Right-click the expert, then choose the **Validate**, **Generate**, or **Start** command from the popup menu.

   or

   Select the expert, then choose the **Validate**, **Generate**, or **Start** command from the Design menu.

Creating a Development Environment

Experts have several settings that affect their behavior while running. You will probably want to set them one way while you are developing an expert, and another way when you or other users are using the expert to accomplish real work.

To set the expert parameters for development:

1. In the Expert Explorer, select the expert at the very top of the tree.
2. In the Object Details window, select the following settings:
   - **Show Progress Graph**
   - **Show Task Assistant**
   - **Show Log Window**
   - **Save All Before Start**
   - **Logging**

The two logging parameters (Show Log Window and Logging) display the scripting messages in the Task Assistant when you run the expert. These messages may help you diagnose any problems.

Publishing Experts

The settings that simplify testing and debugging experts are not appropriate when running the expert to accomplish work.

To set the expert parameters for use:

1. In the Expert Explorer, select the expert at the very top of the tree.
2. In the Object Details window, deselect the following settings:
   - **Show Log Window**
   - **Save All Before Start**
   - **Logging**
3. Select the following settings:
   - **Close Windows on Execution**
   - **Finish Dialog on Completion**
   - **Run Standalone**
   - **Close Assistant on Completion**
   - **Revert to Saved on Error**
4. Enter a name for the expert in the Menu Item Display String field.

To publish an expert:

If you have administrative privileges in Warehouse Builder, you can make experts available to other users in any of these places:

- **Global Explorer**: Copy and paste an expert from the Project Explorer to the Global Explorer into a module in the Public Experts folder.
- **Project Explorer**: Right-click a folder and choose **Add/Remove Experts Here**.
- **Design Center Menu**: On the Tools menu, choose **Add/Remove Experts Here**.

- **Windows Program Menu**: Create a batch file that can be run from the Windows Program menu, as described in "Running an Expert From a Batch File" on page 35-11

Experts that appear in the Public Experts folder are available to all users for all projects. Users can scan the contents of the Public Experts folder to find experts relevant to their objectives in Warehouse Builder.

Alternatively, you can make experts more accessible to end users by customizing the menus in the Design Center to include public experts. For example, if you designed an expert for creating a common type of mapping, you can customize the right-click menu for the mapping folder to include this specialized expert.

Running an Expert From a Batch File

You can create a batch file that enables users to run an expert without opening the Warehouse Builder Design Center. Take these steps, which are explained in more detail in the following paragraphs:

1. Create a Tcl script that starts the expert.
2. Create a batch file that opens OMB Plus and passes it the name of the Tcl file.
3. For Windows platforms, create a shortcut to the batch file on the Start menu or on the desktop.

To create a Tcl script:

Use a text editor to create a script containing the following Warehouse Builder scripting commands:

- `OMBCONN`: Connects a user to a Warehouse Builder repository.
- `OMBCC`: Sets the context to the location of the expert.
- `OMUSTART EXPERT`: Starts the expert.
- `OMBDISCONNECT`: Closes the session.

Example 35–1 connects a user named `SCOTT` to the `GCCREP` repository on a host named `SCOTT-PC`. The expert is named `REFRESH_DATA_EXPERT`, and is located in `SALES_PROJECT` in `EXPERT_SALES_MODULE`.

For detailed information about these commands, refer to the *Oracle Warehouse Builder Scripting Reference*.

Example 35–1 Sample Tcl Script for Starting an Expert

```
OMBCONN scott/tiger@scott-pc:1521:orcl USE REPOS 'GCCREP'
OMBCC '/SALES_PROJECT/EXPERT_SALES_MODULE'
OMUSTART EXPERT 'REFRESH_DATA_EXPERT'
OMBDISCONNECT
```

To create a batch file:

Use a text editor to create a batch file that opens OMB Plus and passes it the Tcl file. Be sure to specify the full path names, as shown in the following example:

```
c:\oracle\product\BiToolsHome_1\owb\bin\win32\OMBPlus.bat c:\owb_scripts\Update_
Sales_Data.tcl
```

To create a shortcut:

Search Windows Help for instructions for adding a program to the Start menu. Create a shortcut to the batch file.

36

Keyboard Access

This chapter lists the keyboard options for accessing Warehouse Builder commands. For conventions that are supported by most applications designed for Windows NT/2000/XP, refer to the Microsoft Windows Keyboard Guide.

> **Note:** Oracle Warehouse Builder provides limited keyboard access to common mouse functions. If you are unable to use a mouse, you may need to use the scripting language to accomplish tasks. For information about the scripting language, see Oracle Warehouse Builder API and Scripting Reference.

This chapter includes the following topics:

General Windows Keys

The table lists the keyboard sequences and descriptions for general window keys.

Table 36–1 General Windows Keys

Keyboard Sequence(s)	Description
F1	Launches the Help facility.
F2	Renames selected icon.

Table 36–1 (Cont.) General Windows Keys

Keyboard Sequence(s)	Description
Ctrl + F4	Closes the current active window or dialog box.
Esc	Closes current menu and move Focus to parent (to whatever it was set before activating current menu). Currently Cancel command button enabled.
Delete	Deletes the selected items.
Home	Moves the cursor to the beginning of current line.
End	Moves the cursor to the end of the current line.
Shift + Home	Highlights from current position to beginning of line.
Shift + End	Highlights from current position to end of line.
Tab	Moves Focus to next control.
Shift + Tab	Moves Focus to previous control.
Shift + F10	Opens the shortcut menu for the active object (Focus is on object, item).
Spacebar	Activates current push button when Focus is on Button; toggles check box to yes/no state.
Enter	Activates current push button when Focus is on Button.

Tree View Control Keys

Table 36–2 lists the keyboard shortcuts and descriptions for tree view control keys.

Table 36–2 Tree View Control Keys

Keyboard Sequence	Description
Right arrow	Expands tree on window canvas, when Focus set on objects with hierarchical relationships.
Left arrow	Collapses tree on window canvas, when Focus is set on objects with hierarchical relationships.
Up/Down arrows	Selects the previous or next object.
Home	Moves Focus to the first control on window canvas.
End	Moves Focus to the last control on window canvas.

Accelerator Keys Set for Warehouse Builder Shortcut Keys

Accelerator keys are keyboard shortcuts for actions that are frequently performed. Accelerator keys enable you to bypass the menu by using a specific combination of keystrokes that perform the same function as a corresponding menu item. Table 36–3 lists the keyboard sequences and descriptions for accelerator keys.

Table 36–3 Accelerator Keys Set for Warehouse Builder

Keyboard Sequence	Description
Ctrl + N	Creates an object by invoking the New Wizards.
Ctrl + X	Cuts text.
Ctrl + O	Invokes Object Editor.

Table 36–3 (Cont.) Accelerator Keys Set for Warehouse Builder

Keyboard Sequence	Description
Ctrl + C	Copies text.
Ctrl + V	Pastes text.
Ctrl + F	Opens Object Find dialog.
Shift + Right Arrow	Highlights text.
Ctrl + S	Opens Commit Confirmation dialog.
Ctrl + P	Opens Print dialog.
Ctrl +Tab	Moves from grid to the next available control, chooses the shortcut page tab where available.
Ctrl + F8	Shows the Clipboard contents.
Alt +F6	Switches between two or more Warehouse windows.

Menu Commands and Access Keys

Menu titles and menu items have underlined access keys. Press **Alt** with the access key to activate the control or menu anywhere within the active window. If an item does not have an underlined character, use up or down arrows to move the focus to the menu item and press **Enter**. Access keys can sometimes be used without the **Alt** key for choosing controls or menu items. Use access keys without **Alt** to select items from an open menu. Table 36–4 lists the keyboard sequences and descriptions for menu commands and access keys.

Table 36–4 Menu Commands and Access Keys

Keyboard Sequence	Description
Alt	Activates the menu bar of the active window. The first menu name on the left is selected.
Alt + any printing character	Activates the menu with the underlined character (access key) on the main menu bar.
Any printing character	Activates the menu with the underlined character (access key) on an open menu.
Left/Right arrows	Moves the Focus between menus on the menu bar in the direction of the arrow. If the original menu was open, the target menu will be opened and the focus on the first item.
Up/Down arrows	Opens the selected menu. Select previous and next command on the open menu.
Enter	Opens the selected menu when focus is on the menu title, activates a menu item when focus is on a menu item.
Alt + F4	Closes active window, returns Focus to the setting before activating current window or menu.
Shift + F10	Opens the shortcut menu for the active object (Focus is on object, item).

Mapping Editor Keyboard Operations

You can navigate the Mapping Editor canvas using the Tab key and the keyboard arrow keys.

The Tab key enables you to navigate to the next object on the canvas and select it. When no objects are selected, the system selects the node that is closest to the upper-left corner of the Mapping Editor canvas. The order of navigation is determined by the position in which the objects appear on the canvas. The system follows a Z navigation path.

Within a selected attribute group, you can navigate between attributes using the up and down keys.

When positioned on an input attribute, the left arrow key enables you to navigate to the incoming mapping line. There is only one incoming mapping line for each attribute. The right arrow key is not active.

When you select an object and then press the **Delete** key, the selected object is deleted from the canvas. You can delete the following objects:

- Operators. Warehouse Builder prompts you to confirm the delete before the delete occurs.
- Mapping lines from or to an attribute. You cannot delete mapping lines that start or end at an attribute group or header.

Connect Operators Mnemonic Key Assignments

To provide easy access to the elements of the Connect Operators Dialog, the following mnemonic keys have been assigned. Table 36–5 lists the keyboard sequences and descriptions for mnemonic keys.

Table 36–5 Connect Operators Mnemonic Key Assignments

Keyboard Sequence	Description
Alt + o	OK.
Alt + c	Cancel.
Alt + y	Copies source attributes to target group and match.
Alt + p	Matches by position of source and target attributes.
Alt + n	Matches by name.
Alt + i	Ignores case differences.
Alt + s	Ignores special characters.
Alt + e	Ignores source prefix.
Alt + u	Ignores source suffix.
Alt + t	Ignores target prefix.
Alt + a	Ignores target suffix.
Alt + g	Go.
Alt + d	Displays mappings.
Alt + h	Help.

Cube Shortcut Keys

Table 36–6 lists the keyboard shortcuts and descriptions for the cube

Table 36–6

Keyboard Sequence	Description
F8	Adds a new or existing dimension.
F9	Adds a new or existing cube.

Tables, Views, Materialized Views, and User Defined Types (Object Types, Varrays, Nested Tables) Shortcut Keys

Table 36–7 lists the keyboard shortcuts and descriptions for Tables, Views, Materialized Views, and User Defined Types.

Table 36–7

Keyboard Sequence	Description
F2	Adds a new or existing table.
F3	Adds a a new or existing View.
F4	Adds a new or existing Materialized View.
F5	Adds a new or existing Object Type.
F6	Adds a new or existing Varray.
F7	Adds a new or existing Nested Table.

Dimension Shortcut Keys

Table 36–8 lists the keyboard shortcuts and descriptions for Dimensions.

Table 36–8

Keyboard Sequence	Description
F8	Adds a new or existing Dimension.
F9	Adds a new or existing Cube.

Business Definitions Shortcut Keys

Table 36–9 lists the keyboard shortcuts and descriptions for Business Definitions.

Table 36–9

Keyboard Sequences	Description
F11	Adds a new or existing Item Folder.
F12	Adds a new or existing Business Area.

Mappings Shortcut Keys

Table 36–10 lists the keyboard shortcuts and descriptions for Mappings

Table 36–10

Keyboard Sequence	Description
F2	Edits Mapping.
F4	Validates the mapping.
F5	Generate.

Pluggable Mappings Editor Shortcut Key

Table 36–11 lists the keyboard shortcut and descriptions for Pluggable Mappings editor.

Table 36–11

Keyboard Sequence	Description
F2	Edits a Mapping.

37

Reserved Words

This chapter lists the reserved words that should not be used to name objects in Warehouse Builder.

Reserved Words

Table 37–1 provides the list of reserved words. Do not use these words as physical object names within a Warehouse Builder Project. In addition to these reserved words, do not use words containing the prefix OWB$ as object names.

Table 37–1 List of Reserved Words

Reserved Words	Reserved Words (Cont.)	Reserved Words (Cont.)	Reserved Words (Cont.)
■ ACCESS	■ ADD	■ ALL	■ ALTER
■ AND	■ ANY	■ AS	■ ASC
■ AUDIT	■ BETWEEN	■ BY	■ CHAR
■ CHECK	■ CLUSTER	■ COLUMN	■ COMMENT
■ COMPRESS	■ CONNECT	■ CREATE	■ CURRENT
■ DATE	■ DECIMAL	■ DEFAULT	■ DELETE
■ DESC	■ DISTINCT	■ DROP	■ ELSE
■ EXCLUSIVE	■ EXISTS	■ FILE	■ FLOAT
■ FOR	■ FROM	■ GRANT	■ GROUP
■ HAVING	■ IDENTIFIED	■ IMMEDIATE	■ IN
■ INCREMENT	■ INDEX	■ INITIAL	■ INSERT
■ INTEGER	■ INTERSECT	■ INTO	■ IS
■ LEVEL	■ LIKE	■ LOCK	■ LONG
■ MATERIALIZED_ VIEW	■ MAXEXTENTS	■ MINUS	■ MLSLABEL
■ MODE	■ MODIFY	■ NAME_ ALREADY_ COMMITTED	■ NAME_ ALREADY_ RESERVED_BY_ YOU
■ NAME_ RESERVED_ BY_OTHER_ SESSION	■ NOAUDIT	■ NOCOMPRESS	■ NOT

Table 37–1 (Cont.) List of Reserved Words

Reserved Words	Reserved Words (Cont.)	Reserved Words (Cont.)	Reserved Words (Cont.)
■ NOWAIT	■ NULL	■ NUMBER	■ OWB_NS_FATAL_ ERROR
■ OWBNOOP	■ OF	■ OFFLINE	■ ON
■ ONLINE	■ OPTION	■ OR	■ ORDER
■ PCTFREE	■ PEER_ RESERVATION	■ PRIOR	■ PRIVILEGES
■ PUBLIC	■ RAW	■ RENAME	■ RESOURCE
■ REVOKE	■ ROW	■ ROWID	■ ROWNUM
■ ROWS	■ SELECT	■ SESSION	■ SET
■ SHARE	■ SIZE	■ SMALLINT	■ START
■ STDDEF	■ SUCCESSFUL	■ SYNONYM	■ SYSDATE
■ TABLE	■ TABLEDEF	■ TABLEPAT	■ THEN
■ TO	■ TRIGGER	■ UID	■ UNION
■ UNIQUE	■ UPDATE	■ USER	■ USERDEF
■ VALIDATE	■ VALUES	■ VARCHAR	■ VARCHAR2
■ VIEW	■ WB_CUSTOM_ TRANS	■ WB_ PREDEFINED_ TRANS	■ WHENEVER
■ WHERE	■ WITH	■ No Value	■ No Value

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X