Oracle® Warehouse Builder

User's Guide

11*g* Release 1 (11.1)

B31278-06

January 2009

ORACLE®

Oracle Warehouse Builder User's Guide, 11*g* Release 1 (11.1)

B31278-06

Copyright © 2000, 2009, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

This program contains Batik version 1.6.

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

For additional information about the terms and conditions, search for "Apache License" in Oracle Warehouse Builder online help.

Contents

6 Integrating with Applications

7 Retrieving Data From SAP Applications

8 Flat Files as Sources or Targets

9 Using Microsoft Products as Sources

10 Integrating Metadata Using the Transfer Wizard

11 Integrating with Business Intelligence Tools

12 Designing Target Schemas

13 Defining Oracle Data Objects

14 Defining Dimensional Objects

15 Data Transformation

16 Creating Mappings

17 Source and Target Operators

18 Data Flow Operators

19 Oracle Warehouse Builder Transformations

20 Designing Process Flows

21 Activities in Process Flows

22 Understanding Performance and Advanced ETL Concepts

Preface

This preface contains the following topics:

- Audience
- Documentation Accessibility
- Conventions
- Getting Help
- Related Publications

Audience

This book 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.

TTY Access to Oracle Support Services

To reach AT&T Customer Assistants, dial 711 or 1.800.855.2880. An AT&T Customer Assistant will relay information between the customer and Oracle Support Services at 1.800.223.1711. Complete instructions for using the AT&T relay services are available at `http://www.consumer.att.com/relay/tty/standard2.html`. After the AT&T Customer Assistant contacts Oracle Support Services, an Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process.

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 Administration Guide*
- *Oracle Warehouse Builder API and Scripting Reference*

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

What's New

This preface includes the following topics:

New in Oracle Warehouse Builder 11*g* Release 1 (11.1)

Changes in the Installation Requirements and Instructions

Due to significant changes in product architecture and security, this release introduces changes to the process for installing Warehouse Builder. For information, refer to the *Oracle Warehouse Builder Installation and Administration Guide*.

Beginning with Warehouse Builder11*g* Release 1 (11.1), the preferred method of implementing metadata security is through the user interface available in the Design Center and described in the *Oracle Warehouse Builder User's Guide*. If, in a previous release, you implemented security using a PL/SQL package, Warehouse Builder 11*g* Release 1 (11.1) does support that implementation.

Data Profiling and Data Correction for Oracle Master Data Management (MDM) Applications

Oracle Warehouse Builder 11*g* Release 1 (11.1) provides connectors that enable you to perform data profiling and data correction for Oracle Master Data Management (MDM) applications. This functionality is called Data Watch and Repair and can be used with the following MDM applications: Customer Data Hub (CDH), Product Information Management (PIM), and Universal Customer Master (UCM).

For more information about Data Watch and Repair, see "Setting Data Watch and Repair for Oracle Master Data Management (MDM)" on page 23-51.

Connectivity to Siebel

With Oracle Warehouse Builder 11*g* Release 1 (11.1), an application connector to Siebel is added. This connector allows you to connect to the Siebel metadata as could be done in previous versions with Oracle E-Business Suite, PeopleSoft, and SAP. For more information, see "Integrating with Siebel" on page 6-7.

Additions to Dimensional Objects

There are some modifications to the functionality for updating records in a Type 2 Slowly Changing Dimension (SCD). For more information, see "Updating Type 2 Slowly Changing Dimensions" on page 14-20.

You can now version hierarchies in a Type 2 SCD. For information about enabling hierarchy versioning, see "Hierarchy Versioning" on page 14-20.

Improvements to the Documentation Set

In this release, 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.

The *Oracle Warehouse Builder User's Guide* now includes enhanced introductory and conceptual information. Certain sections that were only included in the online help of the documentation set are now included in the *Oracle Warehouse Builder User's Guide*.

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

1

Introduction to Oracle Warehouse Builder

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

This section contains the following topics:

- Overview of Oracle Warehouse Builder
- Data Consolidation and Integration

Overview of Oracle Warehouse Builder

Oracle Warehouse Builder is a single, comprehensive tool for all aspects of data integration. Warehouse Builder leverages Oracle Database to transform data into high-quality information. It provides data quality, data auditing, fully integrated relational and dimensional modeling, and full lifecycle 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 a period of time, or it may be the result of business mergers. Whatever the cause of 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 Oracle Database to establish transparent connections to numerous third-party databases, applications, files, and data stores as listed in " Sources and Targets Supported in Warehouse Builder 11.1" on page 4-2.

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

  Warehouse Builder provides an extensive library of data transformations for data types such as text, numeric, date, and others. Use these transformations to reconcile the data from many different sources as described in "Data Transformation" on page 15-1.

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 23-1.

- **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. The process of designing your data store is described in "Designing Target Schemas" on page 12-1.

- **Audit trails**

  After consolidating data from a variety of sources into a single data store, you are likely to face the challenge of verifying the validity of the output information. For instance, can you track and verify how a particular number was derived? This is a question often posed by decision makers within your organization and by government regulators. The process of accessing deployment and auditing information is described in "Auditing Deployments and Executions" on page 27-1.

  See Also: *Oracle Database Licensing Information* for details about options and licensing for Oracle Warehouse Builder

2

Getting Started with Oracle Warehouse Builder

Oracle Warehouse Builder is a flexible tool that enables you to design and deploy various types of data integration strategies. Projects commonly implemented using Warehouse Builder involve mission critical operational systems, migration scenarios, integration of disparate operational systems, and traditional data warehousing. This chapter provides an introduction to using Warehouse Builder. It provides a starting point for using Warehouse Builder for the first time user and serves as a road map to the documentation.

If you have already read the *Oracle Database 2 Day + Data Warehousing Guide*, you may recognize some of the same content repeated here in an expanded format and with additional information for long-term planning and maintenance of not only data warehouses but data integration solutions in general.

This chapter includes the following topics:

- Understanding the Basic Concepts
- Implementing a Data Integration Solution

Understanding the Basic Concepts

Oracle Warehouse Builder is comprised of a set of graphical user interfaces to assist you in implementing solutions for integrating data. In the process of designing solutions, you create various objects that are stored as metadata in a centralized repository, known as a workspace.

The **workspace** is hosted on an Oracle Database. As a general user, you do not have full access to the workspace. Instead, you can access those workspaces to which you have been granted access.

You log in to a workspace by starting the **Design Center,** which is the primary graphical user interface. Use the Design Center to import source objects, design ETL processes such as mappings, and ultimately define the integration solution.

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. In a data warehousing project, for example, the integration solution is a target warehouse. In that case, the mappings you create in the Design Center ultimately define a target warehouse.

After you complete the design of a mapping 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

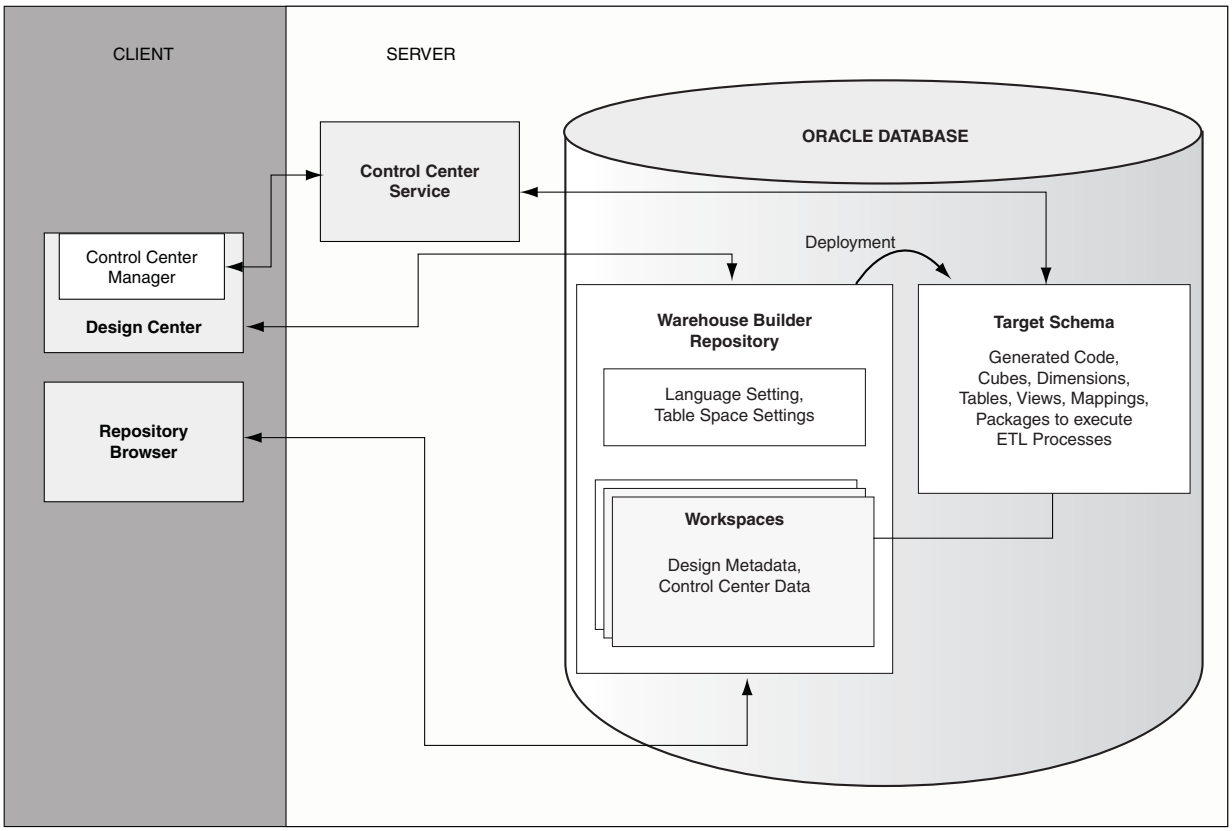
target schema. The **target schema** is generically defined as the Oracle Database which will execute the ETL logic you designed in the Design Center. Specifically, in a traditional data warehousing implementation, the data warehouse is the target schema and the two terms are interchangeable.

Figure 2–1 illustrates the Warehouse Builder components.

As previously noted, the Design Center is the primary user interface. It is also a centralized interface in that you can start 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.

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.

Figure 2–1 Warehouse Builder Components



This screenshot shows the Warehouse Builder components. This screenshot is divided into two halves. To the left half is the client consisting of the following, from top to bottom: Design Center which also contains the Control Center Manager, and the Repository Browser. To the right half is the server consisting of the following from left to right: the Control Center service, and the Oracle Database. The Oracle Database contains the following: to the left is the repository which contains the language settings and Workspaces that contain the design metadata and the control center data. To the right is the target schema which contains the generated code, cubes, dimensions, tables, views, mappings, and packages to execute the ETL process.

****

Implementing a Data Integration Solution

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

1. Before You Begin
2. Preparing the Warehouse Builder Design Center
3. Importing the Source Metadata
4. Profiling Data and Ensuring Data Quality
5. Designing the Target Schema
6. Designing ETL Logic
7. Deploying the Design and Executing the Data Integration Solution
8. Monitoring and Reporting on the Data Warehouse

Before You Begin

Before you can use any of the Warehouse Builder client components, first ensure you have access to a Warehouse Builder workspace.

To begin using Warehouse Builder, take the following steps:

1. Install the Warehouse Builder software and create the necessary workspaces as described in the *Oracle Warehouse Builder Installation and Administration Guide*.

   If an administrator has previously completed the installation, contact that person for the required connection information.

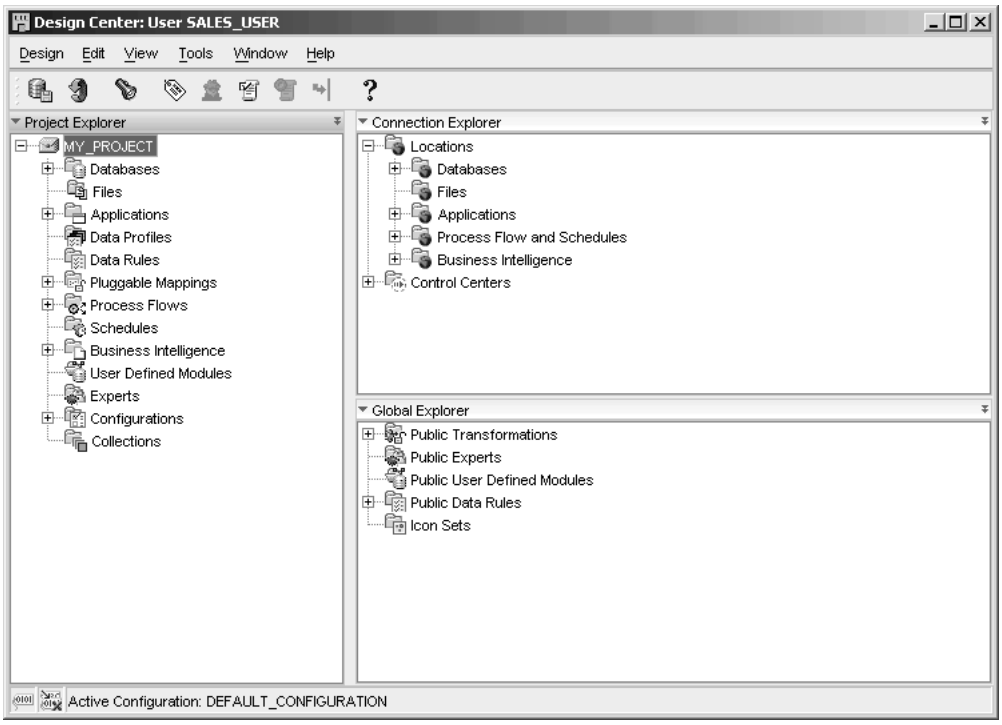
2. Start the Design Center.

   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, run `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 three explorers expanded.

Figure 2–2 The Design Center



This screenshot displays the Design Center. At the top of the window is the menu bar containing the following menu items: Design, Edit, View, Tools, Window, and Help. Below the menu bar is the toolbar consisting of he following icons: Save All, Refresh, Find, Properties, Configure, Validate, Generate, Deploy, and Table of Contents.

Below the toolbar are the following: To the left hand side is the Project Explorer. The right upper half is the Connection Explorer, and to the lower right half is the Global Explorer.

****

Use the Project Explorer to manage design objects for a given workspace. The design objects are organized into **projects** which provide a means for structuring the objects for security and reusability. Each project contains nodes for each type of design object that you can create or import.

Use the Connection Explorer to establish connections between the Warehouse Builder workspace and databases, data files, and applications.

Use the Global Explorer to manage objects that are common to all projects in a workspace and to administer security. Note that the Security node is visible to users who have an administrator role as discussed in the *Oracle Warehouse Builder Installation and Administration Guide*.

Preparing the Warehouse Builder Design Center

To prepare the Design Center, complete the following steps:

1. In the Project Explorer, identify the project to be used.

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

Alternatively, you can rename MY_PROJECT or define more projects. Each project you define is organized in the same fashion with nodes for databases, files, applications, and so on. For a different organization, consider creating optional collections as described in "Defining Collections" on page 3-9.

2. Connect to source and target data objects.

   In the Connection Explorer, establish these connections by defining **locations.** 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 4-4.

3. Identify the target schema.

   Although you can use a flat file as a target, the most common and recommended scenario is to use the Oracle Database as the target schema.

   To define the target schema, 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 Warehouse 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.

4. 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 proceed to the next step.

5. 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 workspace.

   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 25-1.

6. 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.

See Also: *Oracle Warehouse Builder Installation and Administration Guide* for more information about creating additional configurations.

7. 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 3-2.

Importing the Source Metadata

1. Create modules for each type of design object you intend to import metadata.

   In the Project Explorer, select a node such as Files. For that node, determine the locations from which you intend to ultimately extract data. Now create a module for each relevant location by right-clicking on the node and select **New.**

2. Import metadata from the various data sources.

   Right-click the module and select **Import** to extract metadata from the associated location. Warehouse Builder displays 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 4-1.

3. 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 23-1.

Profiling Data and Ensuring Data Quality

Data can only be transformed into actionable information when you are confident of its reliability. Before you load data into your target system, you must first understand the structure and the meaning of your data, and then assess the quality.

Consider using the data profiling option to better understand the quality of your source data. Next, correct the source data and establish a means to detect and correct errors that may arise in future loads. For more information, on data profiling and data quality, see "Understanding Data Quality Management" on page 23-1.

Designing the Target Schema

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 13–1 on page 13-2.

   To create data objects, you can either start 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 "Designing the Target Schema" on page 12-1.

2. As you design objects, 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, select **Validation Messages** from the View menu.

   For additional information, see "Validating Data Objects" on page 12-3.

3. Configure the data objects.

   Configuring data objects sets the physical properties of the object. You must not generate and deploy data objects without specifying the physical property values.

   When you create data objects, Warehouse Builder assigns default configuration property values based on the type of object. In most cases, these default values are appropriate. You can edit and modify the configuration property values of objects according to your requirement. For example, you configure a table to specify the name of the tablespace in which it is created.

   To configure a data object, select the data object in the Project Explorer and click the Configure icon. Or right-click the data object in the Project Explorer and select **Configure**.

4. 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. For more information about generation, see "Generating Data Objects" on page 12-5.

   In the Data Object 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, select **Generated Scripts** from the View menu.

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

Designing ETL Logic

1. 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. Operators represent both data objects and functions such as filtering, aggregating, and so on.

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

2. To manage dependencies between mappings, see "Designing Process Flows" on page 20-1.

Deploying the Design and Executing the Data Integration Solution

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, 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, from the Design Center menu select **Tools**, then **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" on page 25-1.

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 26-2. Or you can execute jobs manually as described in "Starting ETL Jobs" on page 25-8.

Monitoring and Reporting on the Data Warehouse

It is essential to ensure the quality of data entering your data warehouse over time. Data auditors enable you to monitor the quality of incoming data by validating incoming data against a set of data rules and determining if the data confirms to the business rules defined for your data warehouse. For more information about data auditors and data rules, see "Understanding Data Quality Management" on page 23-1.

Although the Control Center Manager displays histories for both deployment and execution, the Repository Browser is the preferred interface for monitoring and reporting on Warehouse Builder operations. For more information, see "Auditing Deployments and Executions" on page 27-1

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
- Setting Preferences
- Defining Collections
- Alternative Interfaces

Organizing Design Objects into Projects

Projects are the largest storage objects within a Warehouse Builder workspace. 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 workspace has a default project called `MY_PROJECT`. You can rename `MY_PROJECT`, or you can delete it after you create other projects. However, a workspace 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 workspace.

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 Warehouse Builder Warning dialog box provides the option of putting the project in the recycle bin.

4. Click **OK** to delete the project.

Deleting Objects in Warehouse Builder

To delete an object, right-click on the object in the Design Center and select **Delete**.

Warehouse Builder imposes few restrictions on deleting objects. For example, you cannot delete the last remaining project in a workspace. Similarly, you cannot delete a location with first unregistering it as described in "Deleting Locations" on page 4-6.

You can choose to move deleted objects to the recycle bin. This allows you to subsequently restore deleted objects back to the location from which they were deleted. To move deleted objects to the recycle bin, set the Recycle Deleted Objects environment preference as described in "Environment Preferences" on page 3-5.

It is possible, however, to delete objects upon which other objects are dependent. To protect important objects from being deleted, you can enable metadata security as described in *Oracle Warehouse Builder Installation and Administration Guide*.

Restoring Deleted Objects from the Recycle Bin

Warehouse Builder enables you to retrieve objects that were moved to a recycle bin. You can retrieve an object only if you did not empty the recycle bin after the object was moved into it. For information about emptying the recycle bin each time you exit the Design Center, see "Environment Preferences" on page 3-5.

To retrieve deleted objects, select **Recycle Bin** from the Tools menu. The Warehouse Builder Recycle Bin is displayed. Click **Restore** to restore the deleted object from the recycle bin and move it back to the location from which it was deleted.

Note that any references to the restored object are not restored. You must manually create the references again. For example, you create two table called EMP and DEPT. The table DEPT has a foreign key reference to the table EMP. You delete DEPT and subsequently restore it from the recycle bin. the foreign key reference to EMP is not restored. You must recreate this manually.

Setting Preferences

Warehouse Builder provides a set of user preferences that enable you to customize your user interface environment. To set user preferences, select **Tools** and then **Preferences** from the Design Center menu. The Preferences dialog box that is used to set preferences is displayed.

The Preferences dialog box contains two sections. The section on the left lists the categories of preferences. The section on the right displays the preferences and their corresponding values. Click a category on the left panel to display the preferences it contains and its value in the right panel.

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 the Locale list to set the language you want the client text to display. This list 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 window 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 window.

Note: Warehouse Builder displays the Job Details window only if you select the **Show Monitor** preference under the Process node of the Deployment preferences category.

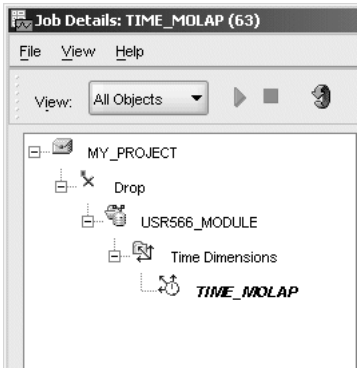
If this option is not selected, view the Job Details window by double-clicking the row representing the deployment or execution job in the Control Center Jobs panel of the Control Center.

You can set the following Control Center Monitor preferences:

- **Show Project:** Select this option to display the project name in the Job Details window 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 window. 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 window.
- **Show Action:** Select this option to display the action performed on the object in the object tree of the Job Details window. 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 window. 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 window in which the following Control Center Monitor preferences were selected: Show Project, Show Module, Show Action, and Show Type.

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



The screenshot shows the Job Details window. At the top is the menu bar containing the following menu options from left to right: File, View, and Help. Below the menu bar is a toolbar that contains, from left to right, the following: View field, Start icon, Stop icon, and Refresh icon. The current selection in the View field is All Objects.

Below the toolbar is the navigation tree that displays the following from top to bottom: MY_PROJECT, Drop, USR566_MODULE, Time Dimensions, and TIME_MOLAP.

****

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, from the Edit menu, select **Properties**. 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 box. 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 window. To display the Job Details window 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 must 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 box. 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 window when you deploy or execute an object. This dialog box 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 box when you deploy an object from the Design Center. The Control Center Jobs dialog box, 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 window 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 must be gathered for the repository schema. If statistics must be gathered, a warning dialog box 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 by selecting **Optimize Repository** from 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 Box:** Select this option to display a dialog box that asks for a confirmation before deleting an object. When this option is selected, if you delete an object, the Warehouse Builder Warning dialog box 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 window or Validation Results window. These dialog boxes 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 window or the Generation Results window.
- **Show Module:** Select this option to display the module node in Validation Results window or the Generation Results window.
- **Show Location:** Select this option to display the location node in Validation Results window or the Generation Results window.
- **Show Action:** Select this option to display the action node in Validation Results window or the Generation Results window.
- **Show Type:** Select this option to display the type node in Validation Results window or the Generation Results window.

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.
- **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 Language 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 Language 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 Language 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 start 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 the workspace. You can start tools such as the Data Viewer and Debugger without entering passwords each time.

> **See Also:** *Oracle Warehouse Builder Installation and Administration Guide*, for more information about the encryption methodology.

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.

> **See Also:** *Oracle Warehouse Builder Installation and Administration Guide*, for more information about manually changing the security settings.

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. You can use a collection 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 Create 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
   - Contents Page
   - Summary Page

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 right arrow.

   The wizard displays the list of objects under the Selected section on the right panel. You can remove objects from the list by selecting objects and clicking the 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 box 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 box 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 tab to modify the contents of 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 and expand the collection node in the right panel.

   The list of objects that are referenced in the collection are displayed.

3. Add new objects to the collection by selecting the objects in the Available section and clicking the right arrow.

4. Remove objects referenced in the collection by selecting the objects in the Selected section and clicking the left arrow.

Alternative Interfaces

In addition to the Design Center, Warehouse Builder provides other interfaces to create and implement your data integration solution. One such interface is OMB Plus.

OMB Plus, an extension of the Tcl programming language, is the scripting language provided by Warehouse Builder. It is a flexible, high-level command line metadata

access tool for Warehouse Builder. With OMB Plus, you can write the syntactic constructs such as variable support, conditional and looping control structures, error handling, and standard library procedures.

Use OMB Plus to create, modify, delete, and retrieve object metadata in Warehouse Builder repository. You can use this scripting interface to:

- Perform complex actions directly in Warehouse Builder, without launching the client user interface.
- Define sets of routine operations to be executed in Warehouse Builder.
- Perform batch operations in Warehouse Builder.
- Automate a series of conditional operations in Warehouse Builder.

To access OMB Plus:

Select **Start**, **Programs**, **Oracle - *OWB_HOME***, **Warehouse Builder**, then **OMB Plus**.

or

From the Design Center, select **Window**, then **OMB*Plus**.

The Design Center displays the OMB*Plus panel.

> **See Also:** *Oracle Warehouse Builder API and Scripting Reference*, for information about OMB Plus and its commands.

Part I

Sources and Targets

This part contains the following chapters:

- Chapter 4, "Identifying Data Sources and Importing Metadata"
- Chapter 5, "Importing Design Definitions from Third Party Design Tools"
- Chapter 6, "Integrating with Applications"
- Chapter 8, "Flat Files as Sources or Targets"
- Chapter 9, "Using Microsoft Products as Sources"
- Chapter 10, "Integrating Metadata Using the Transfer Wizard"
- Chapter 11, "Integrating with Business Intelligence Tools"

4

Identifying Data Sources and Importing Metadata

In Oracle Warehouse Builder you can access data from a variety of sources. You can interpret and extract metadata from custom as well as packaged applications and databases. As a precursor to extracting any data set, you first import its metadata.

This chapter includes the following topics:

- About Source Data and Metadata
- Supported Sources and Targets
- General Steps for Importing Metadata from Sources
- Using the Import Metadata Wizard
- Reimporting Definitions from an Oracle Database

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. You can overcome these problems by first importing 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.

Before you import source metadata into Warehouse Builder, first create a module that will contain these metadata definitions. The type of module you create depends on the source from which you are importing metadata. For example, to import metadata definitions from an Oracle database, create an Oracle module. To import metadata definitions from flat files, create a flat file module.

Supported Sources and Targets

Table 4–1 lists the data storage systems and applications that Warehouse Builder 11.1 can access. The table lists the supported sources and targets for each **Location** node as displayed in the Connection Explorer.

Table 4–1 Sources and Targets Supported in Warehouse Builder 11.1

Location Node in the Connection Explorer	Supported Sources	Supported Targets
Databases/Oracle	Oracle DB 8.1, 9.0, 9.2, 10.1, 10.2, 11.1	Oracle DB 9.2, 10.1, 10.2, 11.1
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. See "Using Microsoft Products as Sources" on page 9-1.	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 "Importing Definitions from Flat Files" on page 8-9.	Comma-delimited and XML format flat files. See "Flat Files as Sources or Targets" on page 8-1.
Applications	SAP R/3: 3.*x*, 4.0*x*, 4.6*x*, 4.7, 5.0; mySAP ERP 2004; mySAP ERP 2005 (with SAP NetWeaver 2004, SAP BASIS 700 Components). See "Retrieving Data From SAP Applications" on page 7-1. Oracle E-Business Suite, see "Integrating with E-Business Suite" on page 6-1 PeopleSoft 8, 9, see "Integrating with PeopleSoft" on page 6-4 Siebel, see "Integrating with Siebel" on page 6-7	None
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	In general, you can deploy a schedule in any Oracle database location, version 10*g* or later. To deploy a schedule in Concurrent Manager, version 11*i* or 12*i* is required. However, for both versions, you must select 11*i* as the version when you create a location in Warehouse Builder.

Table 4–1 (Cont.) Sources and Targets Supported in Warehouse Builder 11.1

Location Node in the Connection Explorer	Supported Sources	Supported Targets
Business Intelligence/Discoverer	None	Discoverer 10.1
Databases/Transportable Module Source	See "Moving Large Volumes of Data" in the *Oracle Warehouse Builder Installation and Administration Guide*.	N/A
Databases/Transportable Module Target	N/A	See "Moving Large Volumes of Data" in the *Oracle Warehouse Builder Installation and Administration Guide*.

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 computer. 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.

General Steps for Importing Metadata from Sources

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

1. Review the list of supported sources and targets in Table 4–1 to determine if the source from which you want to extract data is supported in Warehouse Builder.
2. Create a location as described in "Creating Locations" on page 4-4.
3. Create a module for the source metadata as described in "Creating Modules" on page 4-7.
4. Right-click the module and select **Import.**
5. Follow the prompts in the Import Metadata Wizard.

   The wizard prompts you for information based on the type of source you selected. For more information, see "Using the Import Metadata Wizard" on page 4-9.
6. (Optional) For Oracle data objects, view the data stored in the data object using the Data Viewer. Right-click the object and select **Data**.

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 a period of time, the source metadata may change. If this occurs, you can use Warehouse Builder to identify the ETL logic that would be impacted and potentially made invalid due to a change in metadata.

See Also:

- "Managing Metadata Dependencies" in the *Oracle Warehouse Builder Installation and Administration Guide*
- "Example: Updating a Target Schema" on page 25-13

To introduce the changed metadata into Warehouse Builder, right-click the desired module and select **Import.** As described in "Reimporting Definitions from an Oracle Database" on page 4-11, Warehouse Builder recognizes when you are reimporting metadata.

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 4-2.

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 re-use 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.

Automatically Created Locations

During installation, Warehouse Builder creates an Oracle location named `OWB_REPOSITORY_LOCATION`. This location provides the connection details to the Warehouse Builder workspace. You cannot rename or delete the workspace 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 workspace location.

Types of Locations

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.
- **Applications:** Targets for SAP systems.
- **Process Flows and Schedules**: Targets for managing ETL.
- **Business Intelligence**: Targets for metadata derived from Databases or Oracle modules.

Creating Locations

In addition to the automatically created locations, you can create your own locations that correspond to target schemas that you want to use as sources or targets.

To create a location:

1. In the Connection Explorer, expand the Locations node and then the node that represents the type of location you want to create.

   For example, to create an Oracle database location, expand the Locations node, the Databases node, and then the Oracle node.

2. Right-click the type of location and select **New**.

   The Create <location_type> Location dialog box is displayed.

3. Complete the dialog box. Click the **Help** button for additional details.

Using SQL*Net to Create Locations

When you create Oracle locations of type SQL*Net, you must set up a TNS name entry for these locations. The TNS name must be accessible from the Oracle Database home. To do this, run the Net Configuration Assistant from the Oracle Database home.

While setting up a TNS name for use during deployment and execution of maps and process flows, the TNS name must be accessible from the Warehouse Builder home used to run the control center service. To make the TNS name accessible, run the Net Configuration Assistant from the Warehouse Builder home. Next, restart the control center service so that it can pick up the changes.

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.

> **See Also:** *Oracle Warehouse Builder Installation and Administration Guide* for more information about managing passwords

Granting Privileges to a Target Location

Some deployments require the owner of the target location to have more powerful privileges than are granted 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 and Unregistering 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 workspace 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, select **Register.**

   The Location dialog box is displayed.
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, select **Unregister**.
3. Click **OK** to confirm the action.

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 Manager, and then you can delete the location from the Connection Explorer.

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 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 Language Reference* for more information about the `CREATE DATABASE LINK` command
- *Oracle Database SQL Language Reference* for more information about the `CREATE DIRECTORY` command

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 select **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 select **New**.

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

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 reimport 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.

Creating Modules

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. 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 select **New**.

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

3. On the Name and Description page, provide a name and an optional description for the module.
4. Click **Next**.

   The Connection Information page is displayed.
5. Provide details about the location that is associated with this module.

   The contents of the Connection Information page depend on the type of module you create. For more information about providing information on this page, click **Help**.
6. Click **Next** to display the Summary page.

   Review the information you provided and click **Back** to modify entered values.
7. Click **Finish**.

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.

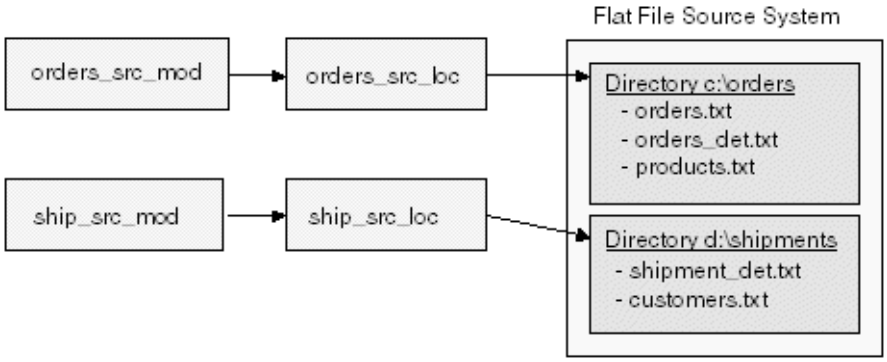
   The Configuration Properties dialog box is displayed.
3. In the Identification folder, select a new value for the Locations property.

Example: Importing Metadata from Flat Files

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

Figure 4–1 provides a diagrammatic representation of accessing flat file data stored in different drives or directories on your source system. Each location maps to a particular directory on your source system.

Figure 4–1 Importing Data From Flat File Sources



This image describes how to access data from flat file sources. On the left are two rectangles labeled orders_src_mod and ship_src_mod that represent the flat file source modules. In the center are two rectangles labeled orders_src_loc and ship_src_loc that represent the locations corresponding to these source modules. There is an arrow from orders_src_mod to orders_src_loc and another arrow from ship_src_mod and ship_src_loc. On the right is a rectangle that represents the file system. This rectangle contains the following from top to bottom: a rectangle labeled Directory c:\orders, that represents the directory c:\orders on the file system, and a rectangle labeled Directory d:\shipments, that represents the directory d:\shipments. There is an arrow from the rectangle labeled orders_src_loc to the rectangle labeled Directory c:\orders. There is also an arrow from the rectangle labeled ship_src_loc to the rectangle labeled Directory d:\shipments.

****

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 modules.

The Import Metadata Wizard supports importing of tables, views, materialized views, dimensions, cubes, external tables, sequences, 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.

You can import metadata definitions either from 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 a warning dialog box. This dialog box informs you that you must first specify the location details. Click **OK**. The Edit Oracle Database Location dialog box for the Oracle module is displayed. Use this dialog box to specify the location information. Clicking **OK** on this dialog box displays the Welcome page of Import Metadata Wizard.

2. Complete the following pages:
   - Filter Information Page
   - Object Selection Page
   - Summary and Import Page

- Import Results Page

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 Types The Object Type section displays the types of database objects that you can import. This include tables, dimensions, external tables, sequences, materialized views, cubes, views, PL/SQL transformations, and user-defined types. Select the types of objects you want to import. For example, to import three tables and one view, select **Tables** and **Views**.

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 right 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 **Move All**.

Importing Dependent Objects The Import Metadata wizard enables you to import the dependent objects of the object being imported. If you are reimporting definitions, previously imported objects appear in bold.

Select one of the following options to specify if dependent objects should be included in the import:

- **None**: Moves only the selected object to the Selected list. No dependencies are imported when you select this option.
- **One Level**: Moves the selected object and the objects it references to the Selected list. This is the default selection.
- **All Levels**: Moves the selected object and all its references, direct or indirect, to the Selected list.

Click **Next** to display the Summary and Import page.

Importing Dimensions When you import a dimension that uses a 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 must bind the dimension to its implementation table. For more information on how to perform binding, see "Binding" on page 14-3.

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, if required, 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 box is displayed. For more information on the contents of this dialog box, see "Advanced Import Options" on page 4-12.

Click **Finish** to import the selected objects. The Importing Progress dialog box 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.

Reimporting Definitions from an Oracle Database

Reimporting 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 workspace. 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 reimport 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 reimported.

4. The Summary and Import page displays. For objects that already exist in the workspace or ones that you are reimporting, the Reimport action is displayed in the Action column.

   If the source contains new objects related to the object you are reimporting, the wizard requires that you import the new objects at the same time. For these objects, the Create action displays in the Action column.

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 box displays.

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 reimport.

7. Click **OK** to proceed.

   Click **Undo** to undo all changes to your workspace.

Advanced Import Options

The Advanced Import Options dialog box displays the options that you can configure while importing objects. This dialog box enables you to preserve any edits and additions made to the object definitions in the Warehouse Builder workspace.

By default, all options on this dialog box are checked. Clear boxes to have these 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 box 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 SQL Collections

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 are not preserved.
- **Preserve repository added columns:** The columns you added to the object in the workspace 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 to the table in the workspace.
- **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 consists of the following options:
  - **Preserve repository added indexes:** Select this option to retain any indexes added to the workspace 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 workspace 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 workspace 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 workspace 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 workspace table must be preserved.
- **Import descriptions:** Select this option to import the descriptions of the table.

Advanced Import Options for Object Types

Select these options for reconciling 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 workspace.

Advanced Import Options for SQL Collections

SQL collection includes nested tables and Varrays.

Import descriptions: Select this option to import the descriptions of nested tables and Varrays.

Updating Oracle Database Source Definitions

The Edit Module dialog box 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 box 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 Locations 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 workspace 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 list.

3. To edit the data location, click the Data Locations 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 dialog box displays. Specify the details of the data location here.

5

Importing Design Definitions from Third Party Design Tools

This chapter shows you how to integrate design definitions from third-party design tools such as Oracle Designer.

This chapter includes the following:

- Using Design Definitions from Oracle Designer 6i/9i
- Example: Importing from CA Erwin

Using Design Definitions from Oracle Designer 6*i*/9*i*

You can create a source module that connects to 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 Elements contain definitions for namespace and ownership of objects, and enable you to view objects even if they are owned by a different user. Because Designer 6*i*/9*i* container elements are controlled by workareas, they are version controlled. 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. To select Designer 6*i*/9*i* objects as a source:

- Specify a workarea, and
- Specify the container element in the workarea

The Module Editor detects the Designer version to which you are connected. If it finds Designer 6*i*/9*i*, the Metadata Location tab shows two 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 4-9.

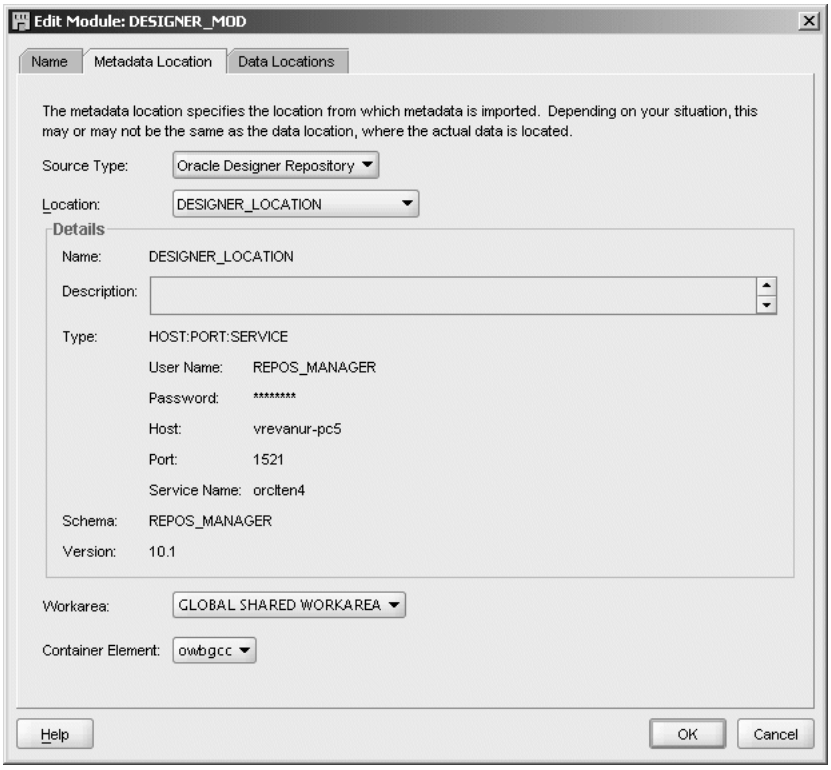
Using Designer 6*i*/9*i* as a Metadata Source

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 lists, **Workarea** and **Container Element**, are visible in the Metadata Location tab.
4. Select the Designer *6i/9i* object from the workarea and select the specific container element.

Figure 5–1 The Metadata Location Tab



This image displays the Edit Module window. At the top are three tabs ordered from left to right: Name, Metadata Location, and Data Locations. The Metadata Location tab is currently displayed. You can specify the workarea and container element which contains the Oracle Designer *6i/9i* object. At the top are the following list ordered from top to bottom: Source Type list (currently displaying as Oracle Designer Repository) and the Location list (currently displaying as (DESIGNER_LOCATION). Below the list is the Details box. The Details box contains the following ordered from top to bottom:

- Name label (currently displaying as DESIGNER_LOCATION)
- Description field
- Type label containing the following information:
  - HOST.PORT.SERVICE
  - User Name
  - Password
  - Host
  - Port
  - Service Name
- Schema label (currently displaying as REPOS_MANAGER
- VERSION label (currently displaying as 10.1)

Below the Details box are the following lists ordered from top to bottom:

- Workarea list (currently displaying as GLOBAL SHARED WORKAREA)
- Container Element list (currently displaying as owbgcc)

At the lower left-hand corner of the window is the Help button. At the lower right-hand corner of the window, the following buttons, ordered from left to right, OK and Cancel.

****

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
- Reimporting Definitions from an Oracle Database
- Updating Oracle Database Source Definitions

Example: Importing from CA Erwin

Scenario

A movie rental company uses tools from different vendors for data modelling, extraction, transformation and loading (ETL), and reporting purposes. Using a variety of tools has led to several metadata integration issues for this company. Often, the design work done using one tool cannot be completely integrated or reused in another. This company wants to find a method to streamline and integrate all its metadata designs and ETL processes using a single tool.

Solution

Warehouse Builder enables the company to import and integrate metadata designs from different tools and use them for data modelling and ETL purposes using only one tool. Warehouse Builder uses the seamlessly integrated technology from Meta

Integration Technology Inc. (MITI) to import the metadata and reuse the data models designed by other third-party tools.

This case study shows you how to easily import design files developed using CA ERwin into Warehouse Builder. You can then reuse the metadata for ETL design and reporting using a single tool. You can follow this model to import files from other tools such as Sybase PowerDesigner and Business Objects Designer.

Case Study

This case study shows you how the movie rental company can migrate their ERwin data model designs into Warehouse Builder. They can also use this model to import designs from other third party tools and consolidate their design metadata in a central workspace. Follow these steps:

1. Download Metadata from CA ERwin
2. Install the Meta Integration Model Bridge
3. Create an MDL File from the CA ERwin Data
4. Import the MDL file into Warehouse Builder

Use Warehouse Builder Transfer Wizard to import the ERwin metadata into Warehouse Builder.

Download Metadata from CA ERwin

Download the design metadata from CA ERwin to your local system.

Install the Meta Integration Model Bridge

Warehouse Builder enables you to integrate with Meta Integration Model Bridges (MIMB). These bridges translate metadata from a proprietary metadata file or repository to the standard CWM format that can be imported into Warehouse Builder using the Warehouse Builder Transfer Wizard. To import files from different design tools into Warehouse Builder, you must first obtain an MIMB license and install the bridges on your system. Follow these steps to complete the installation.

To download MIMB:

1. Download the Model Bridge product from the following Web site:

   `http://www.metaintegration.net`

2. Install the MIMB by running the setup on your system.
3. During installation, select **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 computer, you must provide the JNI library directory path name. Your path environment variable must contain the `metaintegration` directory. If not, you must add it to the path:

   ```
   c:\program files\metaintegration\win32
   ```

Create an MDL File from the CA ERwin Data

Create an MDL file from CA ERwin using Warehouse Builder.

After you install the MIMB product, follow these steps to create an MDL file from ERwin and other third party design tools:

1. From the Project Explorer, select and expand the Project node to which you want to import the metadata. In this example, the ERwin files are imported into `MY_PROJECT`.
2. From the **Design** menu, select **Import, Bridges** to start the Warehouse Builder Transfer Wizard.

   This wizard seamlessly integrates with the MITI technology to translate the third-party metadata into a standard CWM format that is imported into Warehouse Builder. Follow the wizard to complete the import.
3. In the Metadata Source and Target Identification page, select **CA ERwin 4.0 SP1 to 4.1** in the From field.
4. In the Transfer Parameter Identification page, provide the path where the ERwin files are located in the Erwin4 Input File field. In this example, the company wants to import the `Emovies.xml` file from ERwin.
5. Accept the default options for all other fields.

   In the OWB Project field, enter the Warehouse Builder project where you want to import the ERwin file. In the Warehouse Builder MDL field, enter a name and select the location to store the .mdl file that will be generated.
6. Complete the remaining wizard steps and finish the import process.

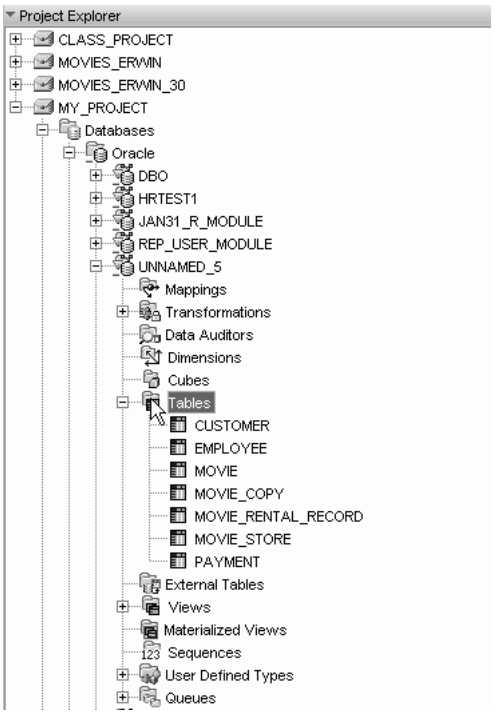
Import the MDL file into Warehouse Builder

Import the MDL file to import metadata from the CA ERwin file into Warehouse Builder. To import the MDL file:

1. Select MY_PROJECT and from the **Design** menu, select **Import**, **Warehouse Builder Metadata** to open the Metadata Import dialog box.
2. In the File Name field, specify the name of the `mdl` file you generated in "Create an MDL File from the CA ERwin Data" on page 5-4.
3. Click **Import** to import the metadata into Warehouse Builder.

   If the metadata file version and the workspace version are not compatible, then the Metadata Upgrade window pops up. Click **Upgrade** to upgrade the `.mdl` file.
4. After you finish importing the ERwin files into Warehouse Builder, expand the MY_PROJECT folder, then the Databases node, and then the Oracle node. You can see the imported source metadata objects, as shown in Figure 5–2.

Figure 5–2 Metadata Objects Imported from CA Erwin

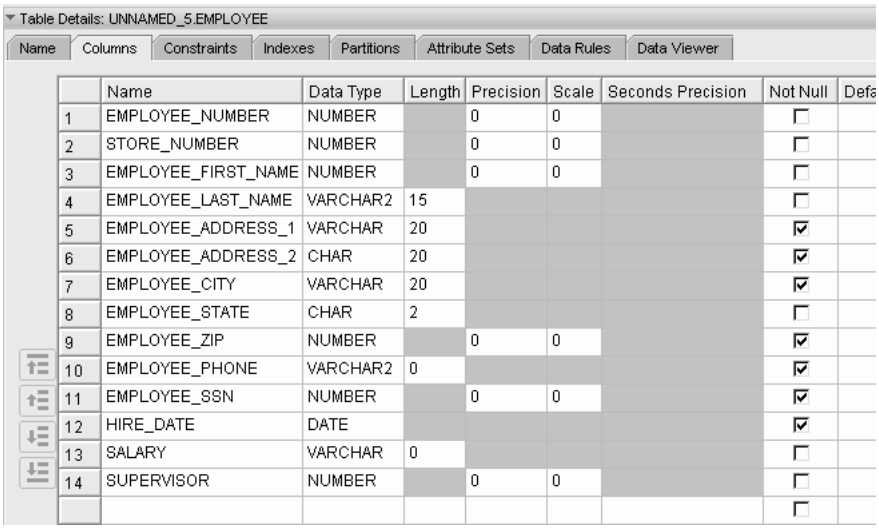


This screenshot shows the various objects in the Project Explorer. The project MY_PROJECT is currently expanded. The Tables node within the Oracle module called UNNAMED_5 is expanded to display the tables imported from the CA ERwin file.

****

5. Double-click the table names to see the properties for each of these tables. Warehouse Builder imports all the metadata including descriptions and detailed information on table columns and constraints, as shown in Figure 5–3.

Figure 5–3 Table Properties Imported from CA Erwin



	Name	Data Type	Length	Precision	Scale	Seconds Precision	Not Null	Defa
1	EMPLOYEE_NUMBER	NUMBER		0	0		☐	
2	STORE_NUMBER	NUMBER		0	0		☐	
3	EMPLOYEE_FIRST_NAME	NUMBER		0	0		☐	
4	EMPLOYEE_LAST_NAME	VARCHAR2	15				☐	
5	EMPLOYEE_ADDRESS_1	VARCHAR	20				☑	
6	EMPLOYEE_ADDRESS_2	CHAR	20				☑	
7	EMPLOYEE_CITY	VARCHAR	20				☑	
8	EMPLOYEE_STATE	CHAR	2				☐	
9	EMPLOYEE_ZIP	NUMBER		0	0		☑	
10	EMPLOYEE_PHONE	VARCHAR2	0				☑	
11	EMPLOYEE_SSN	NUMBER		0	0		☑	
12	HIRE_DATE	DATE					☑	
13	SALARY	VARCHAR	0				☐	
14	SUPERVISOR	NUMBER		0	0		☐	
							☐	

This screenshot displays the Table Details tab of the Data Object Editor. This window has Nine tabs, ordered from left to right: Name, Columns, Constraints, Indexes,

Partitions, Attribute Sets, Data Rules, and Data Viewer. The Columns tab is currently displayed. This tab contains the column descriptions of the selected table. It shows the following details for each column: Name, Data Type, Length, Precision, Scale, Seconds Precision, Not Null.

****

6. The designers at the movie rental company can use these sources tables to model ETL designs in Warehouse Builder, generate ETL code, and run reports on them. Furthermore, Warehouse Builder enables them to easily import all the scattered third-party design metadata and consolidate all their design and development efforts.

6

Integrating with Applications

Warehouse Builder enables you to use integrate with applications such as SAP, Siebel, Peoplesoft, and Oracle E-Business Suite. This chapter describes how to import metadata from these applications.

This chapter includes the following topics:

- Integrating with E-Business Suite
- Integrating with PeopleSoft
- Integrating with Siebel

Integrating with E-Business Suite

Warehouse Builder enables you to import metadata stored in an E-Business Suite database using the Import Metadata Wizard.

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 want 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 to access the metadata and data respectively.

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.

To import E-Business Suite metadata:

1. From the Project Explorer, expand the Applications node.
2. If you have not already done so, create an E-Business Suite module and that will contain the imported metadata.

   To create an E-Business Suite module, right-click ORACLE_EBUSINESS_SUITE under the Applications node and select **New**. The Create Module Wizard is

displayed. Follow the prompts in the wizard. Click **Help** on a wizard page for more information about that page.

Ensure that the location associated with the E-Business Suite module contains information needed to connect to the E-Business Suite source. If you created a location earlier, associate that location with the module being created by selecting that location on the Connection Information page. Or create a new location by clicking Edit on the Connection Information page of the Create Module Wizard. For more information about the details to be entered on this page, click **Help**.

3. Right-click the E-Business Suite source module into which you want to import metadata and select **Import**.

   Warehouse Builder displays the Welcome page for the Import Metadata Wizard.

4. Click **Next.**

5. 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" on page 6-2.

- 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" on page 6-3.

Select 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 box.

2. The Business Component Hierarchy dialog box lists the available E-Business Suite business domains.

> **Note:** It may take between 2 and 10 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 box 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 box displays a list of objects available in the selected business domain.

4. Review this dialog box to ensure that you are selecting the required objects and click **OK** to go back to the Business Component Hierarchy dialog box.

   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 box is displayed.

   Review this dialog box to ensure that you are selecting the required 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 workspace.

The time it takes to import the E-Business Suite metadata to the workspace depends on the size and number of tables and the connection between the E-Business Suite application server and the workspace. Importing 500 or more objects could take one to three hours or more, especially if you are connecting servers in separate Local Area Networks (LANs).

When the import completes, the Import Results dialog box displays. Click **OK** to finish importing.

Integrating with PeopleSoft

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. You can use the Import Metadata Wizard to import metadata from Peoplesoft applications into Warehouse Builder.

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.

To import PeopleSoft metadata:

1. From the Project Explorer, expand the **Applications** node.
2. If you have not already done so, create a Peoplesoft module that will contain the imported metadata.

   Right-click PEOPLESOFT8_9 and select **New**. The Create Module wizard is displayed. Click **Help** on a wizard page for more information about the page.

Ensure that the location associated with the PeopleSoft module contains information needed to connect to the PeopleSoft source. If you created a location earlier, associate that location with the module being created by selecting the location on the Connection Information page. Or create a new location by clicking Edit on the Connection Information page of the Create Module Wizard. For more information about the details to be entered on this page, click **Help**.

3. Right-click the PeopleSoft source module into which you want to import metadata and select **Import**.

   Warehouse Builder displays the Welcome page for the Import Metadata Wizard.

4. Click **Next.**

5. 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" on page 6-5.

- 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" on page 6-6.

Select 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 box.

   The Import Metadata Wizard displays Loading Progress dialog box while it is retrieving the business domains.

2. The Business Component Hierarchy dialog box lists the available PeopleSoft business domains.

   Note: It may take between 2 and 10 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 box 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 box.

4. Review this dialog box to ensure that you are selecting the required 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. In the Object Type section, select the types of 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 box is displayed.

Review this dialog box to ensure that you are selecting an appropriate number of tables.

3. Click **OK.**

   The selected objects appear in the Selected 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 Connector reads the table definitions from the PeopleSoft application server and creates the metadata objects in the workspace.

The time taken to import PeopleSoft metadata to the workspace depends on the size and number of tables and the connection between the PeopleSoft application server and the workspace. Importing 500 or more objects could take one to three hours or more, especially if you are connecting to servers in separate Local Area Networks (LANs).

When the import completes, the Import Results dialog box displays. Click **OK** to finish importing metadata.

Integrating with Siebel

Siebel applications provide Customer Relationship Management (CRM) solutions. Warehouse Builder provides a Connector for Siebel systems that enables you to extract both metadata and data from your Siebel systems.

The Siebel Connector enables you to connect to any Siebel application, read its metadata, import the metadata into Warehouse Builder, and extract data from the system.

Importing Siebel Metadata Definitions

Before you import metadata definitions from Siebel, you must create a Siebel module. You can then import metadata definitions from Siebel using the Import Metadata Wizard. This wizard enables you to filter the Siebel objects you want to import and verify those objects. You can import metadata for tables, views, and sequences.

To import metadata definitions from Siebel:

1. Create a Siebel source module, as described in "Creating a Siebel Source Module" on page 6-8.
2. Import metadata from Siebel, as described in "Importing Siebel Metadata" on page 6-8.

Creating a Siebel Source Module

1. From the Project Explorer, expand the Applications node.
2. Right-click Siebel and select **New**.

   The Create Module wizard is displayed.
3. Click **Next** to display the Name and Description page.
4. Specify a name and an optional description for the Siebel source module and click **Next**.

   The Connection Information page is displayed.
5. Specify the connection information for the Siebel source module and click **Next**.

   Ensure that the location associated with the Siebel module contains information needed to connect to the Siebel source. If you created a location earlier, associate that location with the module being created by selecting the location on the Connection Information page. Or create a new location by clicking **Edit** on the Connection Information page of the Create Module Wizard.

   For more information about the details to be entered on this page, click **Help**.
6. On the Summary page, review the options entered on the previous wizard pages. Click **Back** to modify any selections. Click **Finish** to create the Siebel source module.

Importing Siebel Metadata

1. Right-click the Siebel source module into which you want to import metadata and select **Import**.

   Warehouse Builder displays the Welcome page for the Import Metadata Wizard.
2. Click **Next.**

   The Filter Information page is displayed.
3. Select the objects to be imported and click **Next**.

   Warehouse Builder enables you to select objects using Text String matching. You can search for 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 Siebel application database. For more information, see "Filtering Siebel Metadata by Text String" on page 6-9
4. On the Objects Selection page, select the objects to be imported into the Siebel module and click **Next**.

   You can choose whether you want to import tables with foreign key relationships for each object that you choose to import using the following options on this page:

   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.

5. Review the summary information and click **Finish** to complete the import. To modify any selections, click **Back**.

After you import metadata for tables, views, or sequences from Siebel applications, you can use these objects in mappings.

Filtering Siebel Metadata by Text String

1. Select **Text String, where object**.
2. In the Object Type section, 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 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_`.

7

Retrieving Data From SAP Applications

Many companies implement SAP ERP system and Warehouse Builder enables easy access to the data in these SAP systems.

This chapter describes how you can retrieve 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 retrieve data from the SAP system and load this data to a target table on the Warehouse Builder system.

It consists of 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 on the SAP R/3 versions supported by Oracle Warehouse Builder 11*g*, 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

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 an SAP system.
2. Importing metadata from SAP.
3. Creating an extraction mapping in Warehouse Builder that defines:
   - The SAP source tables from which data is to be retrieved

- The transformation operators that operate on the source tables to retrieve data based on certain criteria
- The target table in Warehouse Builder to store the data retrieved 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 OWB server using FTP.
   - Loading of the target table with the data from the data file using SQL*Loader.

   To execute ABAP code on an SAP system you require SAP Function Modules.

SAP Function Modules

To access SAP data from non-SAP systems, you typically use a function module to execute an ABAP program that retrieves 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 Retrieval

Data retrieval from the SAP system can be Completely Managed By Warehouse Builder, Managed By Warehouse Builder With SAP Verification, or Manual depending on whether you have access rights to the predefined function module or the customized function module.

Completely Managed By Warehouse Builder

In this system, Warehouse Builder has access to upload and execute the generated ABAP using the default function module, and to use FTP to retrieve 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 7-16 for more details on implementing this data retrieval mechanism.

Managed By Warehouse Builder With SAP Verification

In this system, as a Warehouse Builder user, you do not have access rights to the default function module that executes the ABAP code in the SAP system. Instead the

SAP administrator first verifies the ABAP code generated by 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 function module.

See "Semi Automated System" on page 7-18 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 retrieve the generated data file 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 7-19 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 retrieve 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 have to interact with the SAP administrator to retrieve 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 these 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

Usually, you can access the SAP development environment using the predefined function module RFC_ABAP_INSTALL_AND_RUN. As a result, you can implement a completely Automated System for data retrieval.

Test and Production Environment

In the test and production environments, you are not usually given access rights to use the predefined function module. 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 retrieve the resultant data. You can therefore implement either a Semi Automated System or a Manual System for data retrieval.

A typical data retrieval system consists 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.

The following sections provide details of the tasks involved in retrieving data from an SAP system:

1. "Connecting to an SAP System" on page 7-5
2. "Importing Metadata from SAP Tables" on page 7-9
3. "Creating SAP Extraction Mappings" on page 7-13
4. "Retrieving Data From the SAP System" on page 7-16.

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 an 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 7–1 for the list of files required in Windows.

Table 7–1 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.
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`	

Make sure that you 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 make sure to add *`OWB_ORACLE_HOME`*`\owb\bin\admin` to LD_LIBRARY_PATH.

See Table 7–2 for the list of files required in Unix.

Table 7–2 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`	

Make sure that you restart the client after copying the 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 7–3:

Table 7–3 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.
Missing saprfc32.dll	Missing saprfc32.dll file or the 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. You can choose either SAP R/3 version 3.x or SAP R/3 version 4.x system type as your source. After you select the application version, you need to set the connection information between Warehouse Builder and the SAP application server. You can set the connection either by selecting from existing SAP locations or by creating a new SAP location as defined in "Connecting to an SAP System" on page 7-5.

Note: Before you begin the task of creating a SAP location, ensure that you have all the necessary information to create the location. 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: The server name, the user name, password, system number, and client number. Obtain these details from your system administrator.

When you set the connection information, you can choose 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 7-7.

2. Type the connection information in the appropriate fields. The fields displayed on this page depend on the connection type you choose.

   Note: Make sure that you have copied the DLL files to the right location. For more information, see "Required Files For SAP Connector" on page 7-5.

   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 where the data file retrieved from the SAP system is stored. 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 which 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 business domains depending on the network location of the SAP application server, the type of LAN used, or 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 on 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

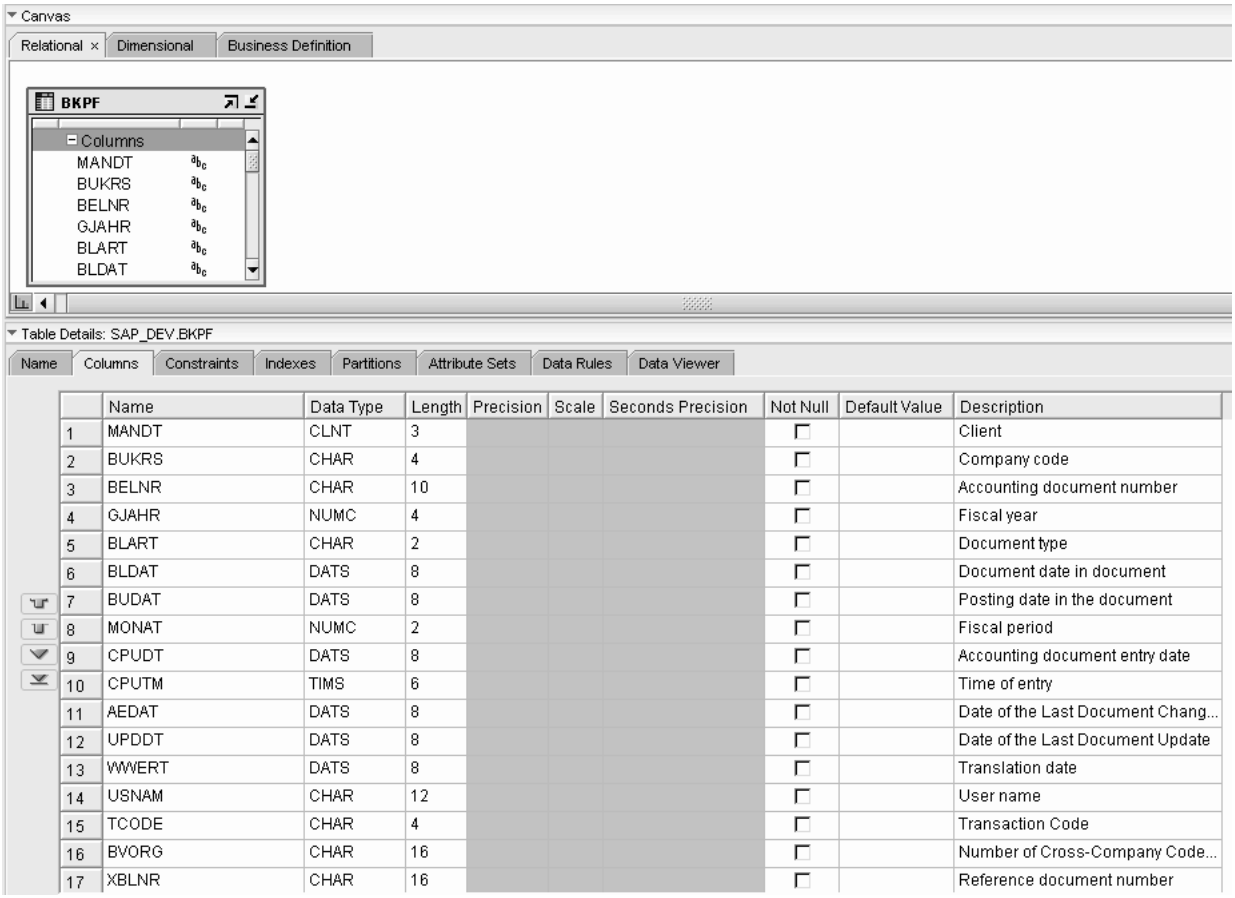
With SAP tables, you cannot view the data after you import the metadata from these tables. However, you can get a good insight about the data that is stored in the tables by viewing the Column Descriptions and the Constraints Details.

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. The description for the columns of the table are visible as shown in Figure 7–1.

Figure 7–1 The Columns Editor with the Description for the Columns of SAP Table



	Name	Data Type	Length	Precision	Scale	Seconds Precision	Not Null	Default Value	Description
1	MANDT	CLNT	3				☐		Client
2	BUKRS	CHAR	4				☐		Company code
3	BELNR	CHAR	10				☐		Accounting document number
4	GJAHR	NUMC	4				☐		Fiscal year
5	BLART	CHAR	2				☐		Document type
6	BLDAT	DATS	8				☐		Document date in document
7	BUDAT	DATS	8				☐		Posting date in the document
8	MONAT	NUMC	2				☐		Fiscal period
9	CPUDT	DATS	8				☐		Accounting document entry date
10	CPUTM	TIMS	6				☐		Time of entry
11	AEDAT	DATS	8				☐		Date of the Last Document Chang...
12	UPDDT	DATS	8				☐		Date of the Last Document Update
13	WWERT	DATS	8				☐		Translation date
14	USNAM	CHAR	12				☐		User name
15	TCODE	CHAR	4				☐		Transaction Code
16	BVORG	CHAR	16				☐		Number of Cross-Company Code...
17	XBLNR	CHAR	16				☐		Reference document number

The figure shows the Columns editor for the SAP table BKPF. The left most column contains the name of each column in the table, while the right-most column provides the description of each column of the table. The details that are visible from left to right include Name, Data Type, Length, Precision, Scale, Not Null, Default Value, and Description.

****

Constraints Details

The other benefit of data object editor is that you can get information on 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 Project Explorer. 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**, **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 retrieve data from the SAP system.

> **Note:** For details of mappings in Warehouse Builder, see Chapter 16, "Creating Mappings".

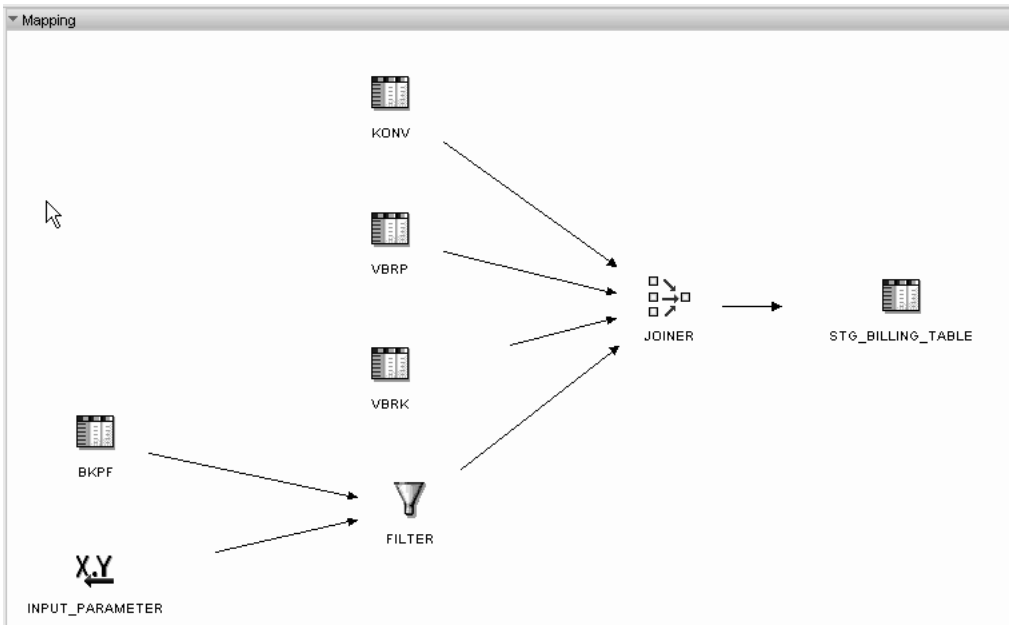
Defining an SAP Extraction Mapping

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 retrieved data, as shown in Figure 7–2.

> **Note:** You cannot have both SAP and non-SAP (Oracle) source tables in a mapping. The staging table though is an Oracle table.

Figure 7–2 SAP Extraction Mapping



The figure shows a mapping to retrieve data from aSAP system. It consists of the following objects from left to right: Table BKPF and an Input Parameter are connected to a Filter operator. Tables KONV, VBRP, VBRK, and the output from the Filter operator are connected to a Joiner. The output data from Joiner is connected to a staging table STG_BILLING_TABLE.

****

In this example, the Input Parameter holds a Date value, and the data from table BKPF is filtered based on this date. The Joiner operator enables you to join data from multiple tables, and the combined data set is stored in a staging 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 7–4.

Table 7–4 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 automatically sets the language parameter 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 code file 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. For MySAP ERP and all other versions, select SAP R/3 4.7. Note that different ABAP code is required 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.

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.

For detailed information about deployment, see "Deploying to Target Schemas and Executing ETL Logic" in *Oracle Warehouse Builder Transformation Guide*.

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.

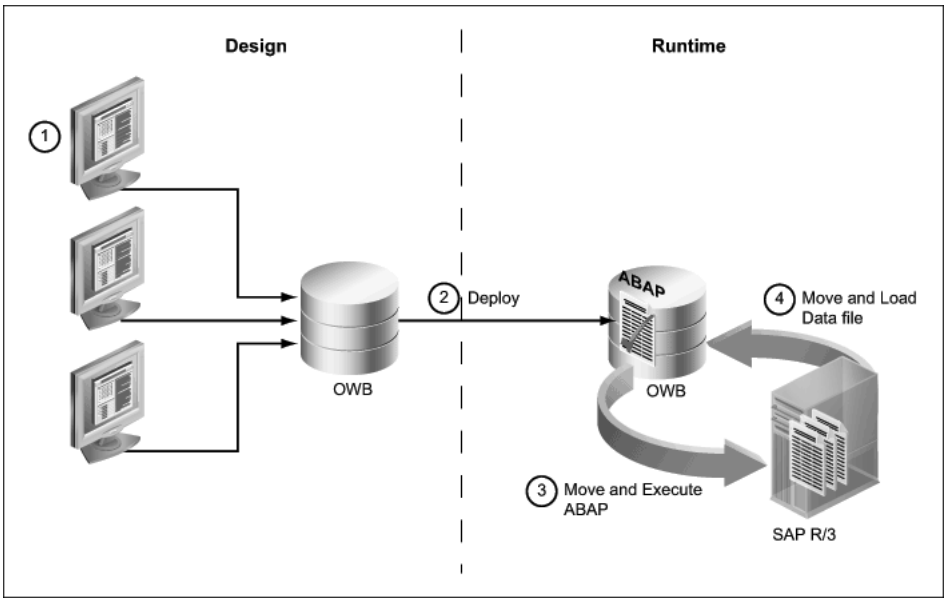
Depending on whether data retrieval from the SAP system is fully automated, semi-automated, or manual, you need to carry out the subsequent tasks. This section consists of the following topics:

- "Automated System" on page 7-16
- "Semi Automated System" on page 7-18
- "Manual System" on page 7-19

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 retrieve data directly from the SAP system without being dependent on the SAP administrator, as shown in Figure 7–3.

Figure 7–3 Automated Data Retrieval



The figure shows the automated data retrieval mechanism. You design and run the mapping in Warehouse Builder. The ABAP code gets executed in the SAP system, and the data is retrieved and loaded to the target table.

****

Because there is no dependence, you can automate the process of sending the ABAP code to the SAP system and retrieving 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 retrieved 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 retrieved.
   - **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 sent using FTP to this directory.
   - 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 code retrieves 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.

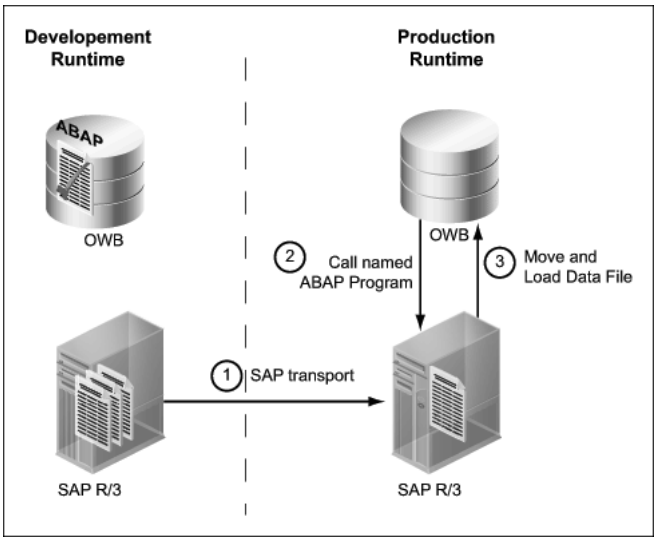
   The file is stored in the location specified in the FTP Directory field.
6. Using SQL*Loader, Warehouse Builder loads the target table in the mapping with the data from the data file.

The advantage of this system is that you can create a fully automated end-to-end solution to retrieve 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, retrieves the resultant data file, and loads the target table with the retrieved data.

Semi Automated System

In a semi automated system, as a Warehouse Builder user, you do not have access to the predefined function module, and therefore cannot use this function module to execute ABAP code. You 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 7–4.

Figure 7–4 Semi Automated Implementation



The figure shows the SAP development environment and the SAP production Environment. ABAP code is designed in the Development environment and transported to the production environment. The SAP administrator creates a new ABAP report name, and from the Warehouse Builder, you call this ABAP program and retrieve data.

****

A semi automated system works as follows:

1. You design the extraction mapping and generate the ABAP code for this mapping.
2. You then transport the ABAP code to the test system to test the code.
3. You then send the ABAP code to the SAP administrator, who loads it to the SAP repository.
4. The SAP administrator creates a new ABAP Report Name.
5. You can then call this ABAP Report Name to execute the ABAP code in the production environment.
6. Before you run the mapping in the SAP system, ensure that you have set the following configuration properties for the mapping:
   - **ABAP Report Name**: The SAP administrator will provide the report name after verifying the ABAP code. You will then execute this ABAP file.
   - **SAP Location**: The location on the SAP system from where data is retrieved.
   - **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 Warehouse Builder system. The data file generated by the execution of the ABAP code is sent using FTP to this directory.
   - Also provide a username who has Write permissions on the FTP directory.
7. In the production environment, when you run the mapping, Warehouse Builder generates the ABAP code and sends it to the SAP system using a Remote Function Call (RFC).
8. In the SAP system, the ABAP code gets executed using the customized function module and a data file is generated.

   This data file is stored in the location specified by Staging File Directory.
9. 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.
10. Warehouse Builder uses SQL*Loader to load the target table with data from the data file.

Manual System

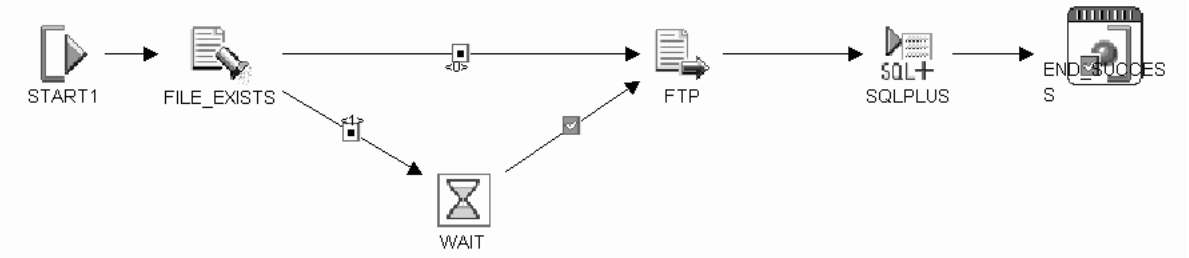
In a manual system, your role as a Warehouse Builder user is restricted to generating the ABAP code for the mapping, and sending the ABAP code to the SAP administrator. The tasks involved in this system are:

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.

On the Warehouse Builder end, you can create a Process Flow to retrieve the data file. The Process Flow must contain the following activities as shown in Figure 7–5:

Figure 7–5 Process Flow to Retrieve SAP Data



The figure shows the activities of a Process Flow. It consists of the following activities from left to right: START, FILE EXISTS, WAIT, FTP, SQL PLUS, END SUCCESS.

****

1. A File Exists activity to check for the presence 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 certain applications, 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.

8

Flat Files as Sources or Targets

You can use data from ASCII and binary flat files as sources and targets. 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
- Using the Create Flat File Wizard
- Importing Definitions from Flat Files
- Using the Flat File Sample Wizard
- Updating a File Definition
- Using External Tables

About Flat Files

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 workspace. 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 Workspace" on page 8-1. For binary format, see "Adding Existing Binary Files to the Workspace" on page 8-2.

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 Workspace

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 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 client and data files reside on Windows operating systems, store the data files on any drive the client computer can access.

2. Creating Flat File Modules on page 8-3

Create a module for each folder in your file system from which you want to sample existing files.

3. Importing Definitions from Flat Files on page 8-9

   Use the Import Metadata Wizard to select the flat files you wish to import. On the Connection Information page, select the option to start the Flat File Sample Wizard.

4. Using the Flat File Sample Wizard on page 8-10

   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 Workspace" on page 8-2.

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, see "External Table Operators versus Flat File Operators" on page 8-3 and"Creating a New External Table Definition" on page 8-26.

6. Updating a File Definition on page 8-23

   If changes are made to the structure of the flat file, update the file definition.

Adding Existing Binary Files to the Workspace

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 8-5.

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, column properties are based on the SQL properties you defined when importing the flat file. For more information about SQL properties for flat files, see "SQL Properties" on page 8-8.

When you use an external table as a source in a mapping, you can use it as 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 for 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 8-26.

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, SQL*Loader code is generated. 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.

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 about differences between external tables and SQL*Loader (flat file operators), see *Oracle Database Utilities*.

Flat Files as Targets

You define a new flat file 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

You can either import or define flat files. Create a flat file modules to store definitions of flat files.

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, the Import Metadata Wizard described in "Importing Definitions from Flat Files" on page 8-9 is displayed.

Or, to define a new flat file, see "Using the Create Flat File Wizard" on page 8-4.

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.

If you intend 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.

Edit File System Location Dialog Box

On the Edit File System Location dialog box, enter the fully qualified directory, including the drive letter, if appropriate.

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 16-3.

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 workspace. 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 reserved words. For a list of reserved words, see "Reserved Words" in the *Warehouse Builder Online Help*.
- **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 computer hosting Warehouse Builder. For complete information about NLS character sets, see *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 list. The 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 list. The 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 Record.**

If the file should contain more than one record type, select **Multi Record.** Use multiple record types to create a flat file with more than one record type. 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.

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 via 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, SQL*Loader and the properties you set here are used. 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 about SQL*Loader field and data types, see *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 8-25.
- **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 17-34.

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, the type is updated if you later

change the SQL*Loader properties. However, if you override the SQL type by selecting a new SQL type from the 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.

Importing Definitions from Flat Files

If you have existing flat files to use as sources, you can import and then sample the metadata from these flat files. Use the Import Metadata Wizard to import metadata from flat files. This metadata must be imported into a file module.

To import flat file definitions:

1. Establish network connectivity to the files you wish to import.

   If you are accessing the data files directly, and if the 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 client and data files reside on Windows operating systems, store the data files on any drive the client computer can access.

2. Create a flat file module that will contain the imported flat file definitions.

   Create a module for each folder in your file system from which you want to import files.

   When you create a flat file module, the location corresponding to this module is a folder in the file system from which metadata is being imported. Use the Connection Information Page of the Create Module Wizard to specify the folder that contains the source metadata.

   Note that a flat file location does not include subfolders of the specified folder.

3. Right-click the flat file module and select **Import**.

   The Import Metadata Wizard is displayed.

4. On the Filter Information page, filter file names by selecting one of the following options:

   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.

5. On the Object Selection page, move the names of the files to be imported from Available Objects on the left to the Selected Objects section on the right.

   Because inbound synchronization for flat files is not permitted, 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.

6. On the Summary and Import page, ensure that metadata for the selected flat files is available in the workspace. You cannot complete the import if the metadata is not present.

   If the Status field contains a red x, metadata is not available in the workspace. For all such files, either select a file with a matching format in the workspace or sample the file.

   Use the **Same As** field to select a file with a matching format.

   To sample a file, select the file and click **Sample**. The Flat File Sample Wizard is launches. The Flat File Sample Wizard enables you to view a sample of the flat file and define record organization and file properties. You can 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 Workspace" on page 8-2.

7. Once you provide metadata information for all files you want to import, click **Finish**.

   The wizard creates definitions for the files, stores the definitions in the flat file module, and inserts the file names under the flat file module in the 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 start 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 multi-byte 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 8-4.

After you complete the Flat File Sample Wizard, it stores the metadata in the workspace 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 16-3.

Describing the Flat File

Use the Name page to describe the flat file you are sampling.

- **Name:** This name uniquely identifies the file in the workspace. 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 workspace 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 reserved words. For a list of reserved words, see "Reserved Words" in the *Warehouse Builder Online Help*.

- **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. The default Globalization Support or National Language Support (NLS) character set matches the character set defined for the computer 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 list. For complete information about NLS character sets, see *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 8-16.

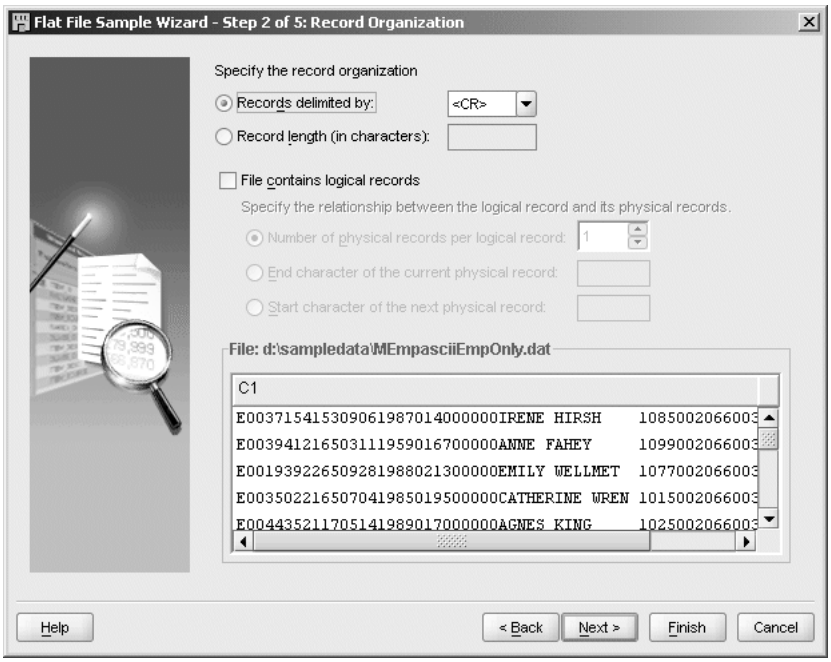
After you complete the file set up information, click **Next** to continue with the wizard.

- 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 8–1 Record Organization Page for the Flat File Sample Wizard



This image displays the Record Organization page for the Flat File Sample wizard. At the top of the page is the Specify the record organization options, ordered from top to bottom, Records delimited by (currently selected) followed by a list box to the right and a Record length (in characters). Below is a File contains logical records check box. The options below the check box are currently disabled because the check box is not selected. Below is the File box displaying the content of the .dat file. At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page, the following buttons, ordered from left to right: Back, Next, Finish, and Cancel.

****

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
```

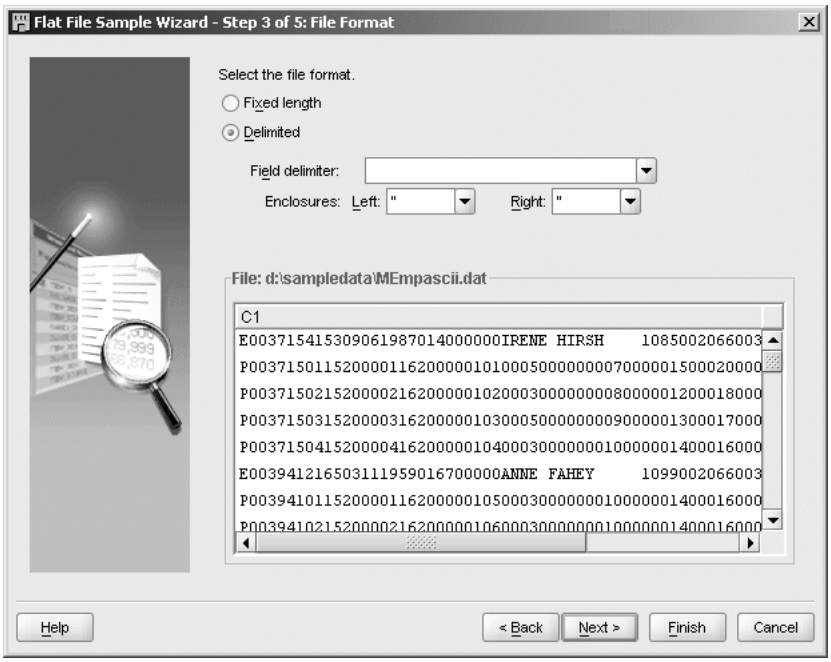
  More than two records can be joined with this technique. The following example shows four physical records for each logical record using continuation at beginning.

```
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 multi-byte character file with a fixed record format. Instead, consider "Using the Create Flat File Wizard" on page 8-4.

Figure 8–2 File Format Page for the Flat File Sample Wizard



This image displays the File Format page for the Flat File Sample wizard. At the top is the Select the file format label with the following options ordered from top to bottom, Fixed length and Delimited (currently selected). Below the options is the Field delimiter list. Below the Field delimiter, is the Enclosure label, followed by a Left list followed by Right list. At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page, the following buttons, ordered from left to right: Back, Next, Finish, and Cancel.

****

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.

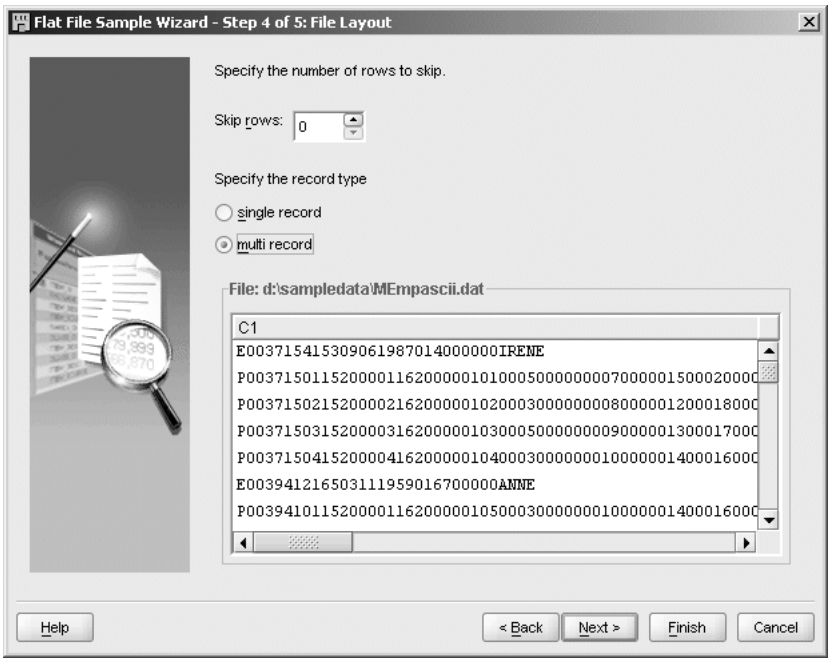
When your file is delimited, specify the following properties:

- **Field delimiter:** Field delimiters designate where one field ends and another begins. You can type in a field delimiter or select one from the list. The 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 list. The 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 to specify the number of rows to skip and to select between a single record type versus multiple record types.

Figure 8–3 File Layout Page for the Flat File Sample Wizard



This image displays the File Layout page for the Flat File Sample wizard. At the top is the Skip rows list. Below the list is the Specify the record type label with the following options, ordered from top to bottom, Single record and multi record (currently selected). Below the options is the File box, showing the contents of the .dat file. At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page, the following buttons, ordered from left to right: Back, Next, Finish, and Cancel.

****

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 about multiple record types, see "Selecting Record Types (Multiple Record Type Files Only)" on page 8-15.

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 8-19. If the data file has a single record type and delimited file format, proceed to "Specifying Field Properties" on page 8-20.

Example: Flat File with Multiple Record Types

In files with multiple record types, one of the fields distinguishes one record type from the next. 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.

Figure 8–4 shows an example of a comma delimited file with two record types, "E" and "P". In this case, instruct the wizard to scan the first field. The wizard returns "E" and "P" as the type values.

Figure 8–4 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
```

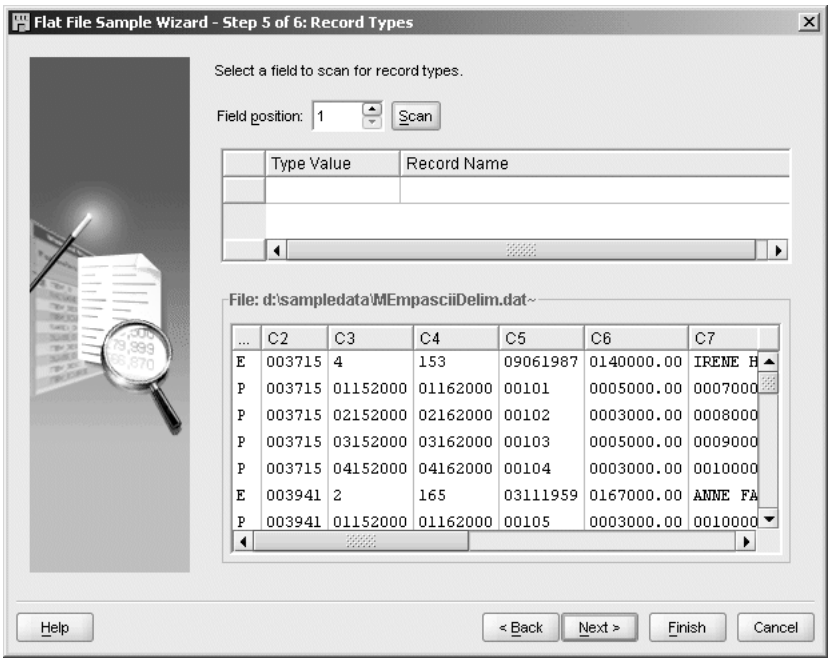
When you use the wizard to sample flat files with multiple record types, ensure that the sample size you specified on the Name page is large enough to include each record type at least once. The default is 10,0000 characters.

Because sampling cannot be canceled 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 8–5 Record Types Page for Delimited Files



This image displays the Record Types page for the delimited files in the Flat File Sample wizard. At the top is the Field position field (currently displaying as 1). To the right of the field is the Scan button. Below the scan button is a table with two columns, ordered from left to right, Type Value and Record Name. Below this table is the File box. This box displays the information about the .dat file in a tabular format. At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page, the following buttons, ordered from left to right: Back, Next, Finish, and Cancel.

****

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.

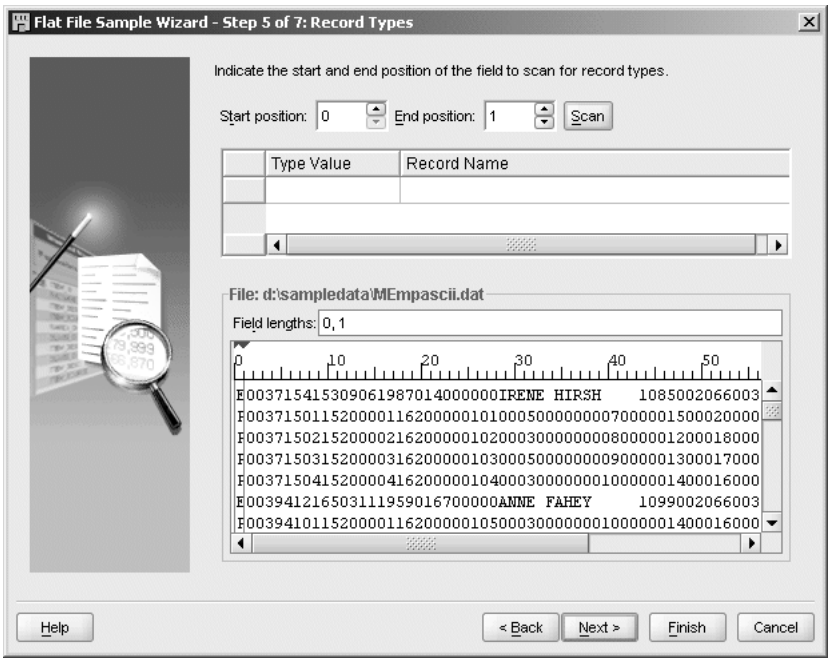
   Click a record name to rename it or select a different record name from the list. You can associate a record name with multiple record type values. You can also add or delete type values using the **New** and **Delete** buttons.

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 8–6 Record Types Page for Fixed Length Files



This image displays the record type page for fixed length files in the Flat File Sample wizard. To the top left is the Start position field (currently displaying as 0) followed to the right by End position field (currently displaying 1). To the right of these fields is the Scan button. Below is a table with two columns ordered, from left to right: Type Value and Record Name. Below this table is a File box. In this box is the Field lengths field (currently displaying as 0, 1). This box also shows the record of the .dat file. At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page, the following buttons, ordered from left to right: Back, Next, Finish, and Cancel.

****

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 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.

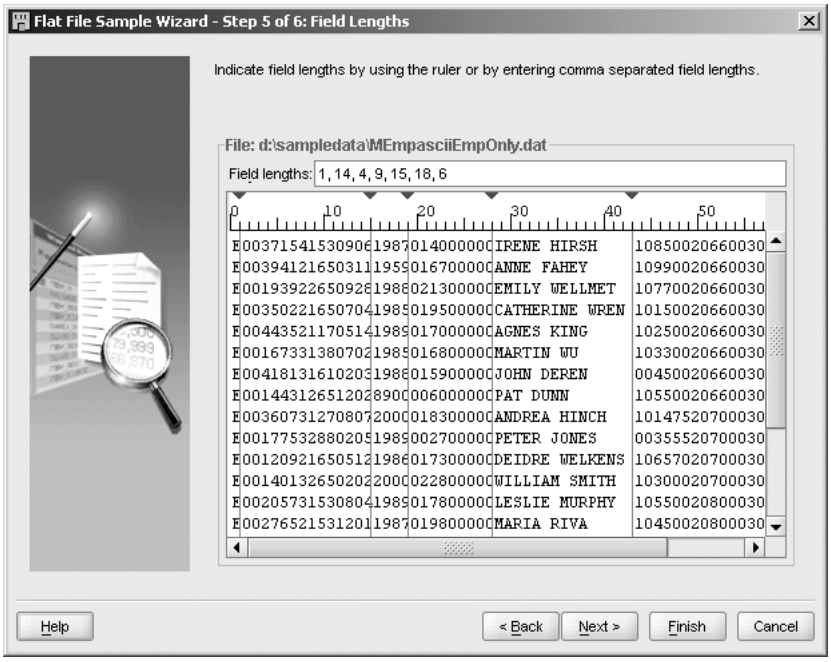
Specifying Field Lengths (Fixed-Length Files Only)

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 8-20.

You can define field lengths by typing in the field lengths or by using the ruler.

Figure 8–7 Field Lengths Page for the Flat File Sample Wizard



This image displays the field lengths page for the Flat File Sample wizard. In the center of the page is a File box. At the top is a Field lengths field (currently displaying as 1, 14, 4, 9, 15, 18, 6). This box also shows the record of the .dat file. At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page, the following buttons, ordered from left to right: Back, Next, Finish, and Cancel.

****

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.

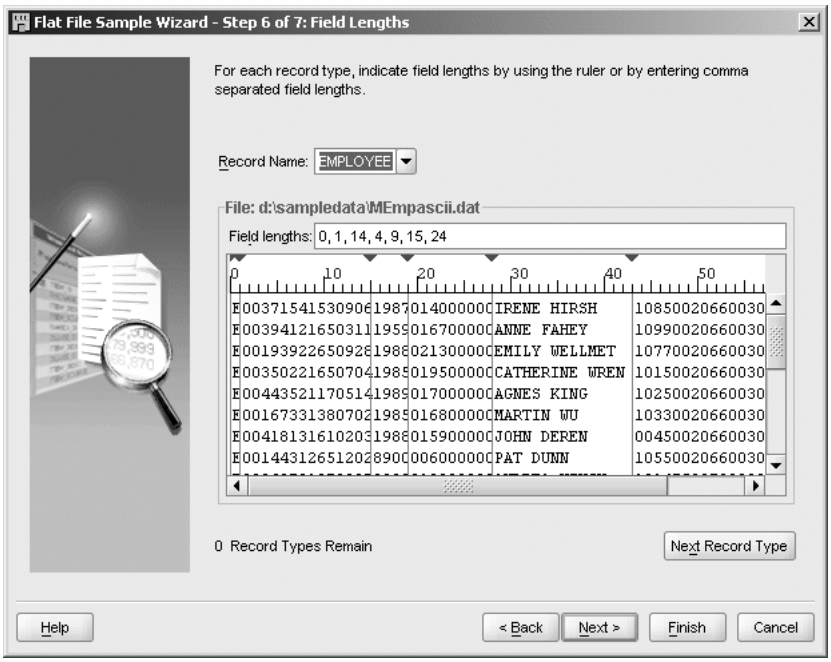
Note that when you specify a field length using the ruler, your tick markers indicate the starting and ending borders for each field. From this information, the positions occupied by each field is determined. For example, a three-character field occupying positions 6, 7, and 8 is internally identified 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 8–8 Field Lengths for Multiple Record Files page



This image displays the field lengths for multiple record files page in the Flat File Sample wizard. In the center of the page is a File box. At the top is a Field lengths field (currently displaying as 0, 1, 14, 4, 9, 15, 24). This box also shows the record of the .dat file. Below the File box, to the right-hand corner is the Next Record Type button. At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page, the following buttons, ordered from left to right: Back, Next, Finish, and Cancel.

****

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.

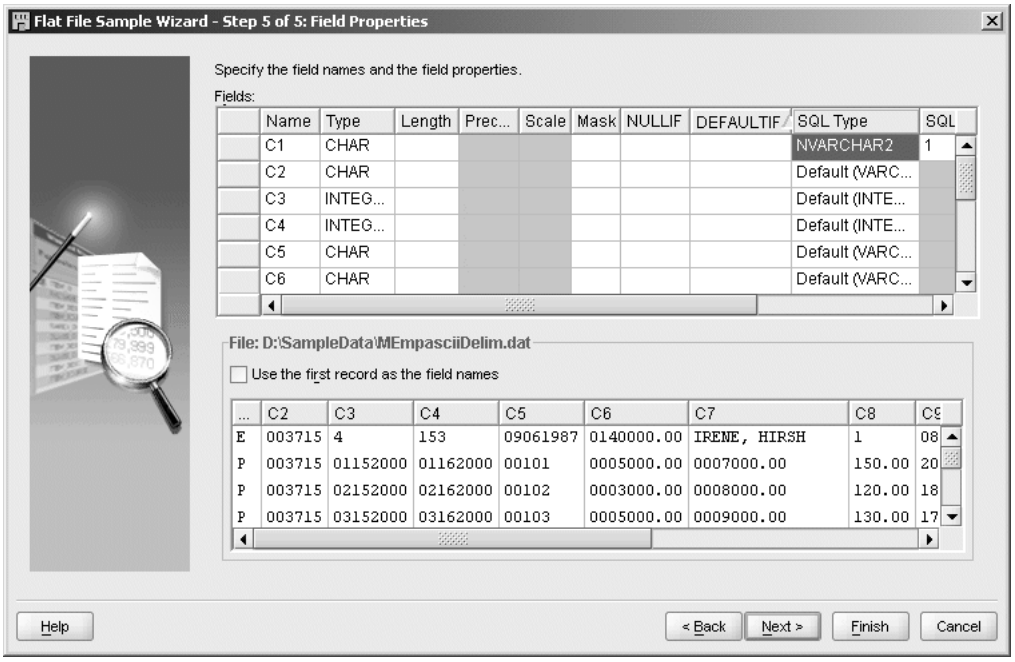
For single record file types, you can instruct the wizard to use the first record in the file to name the fields. Indicate this by checking the box entitled **Use the first record as the field names.** If you select this option, all the field data type attributes default to `CHAR`.

Since you can use a flat file in a mapping either directly as a flat file source or indirectly via 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 "Importing Definitions from Flat Files" on page 8-9. You can select more files to sample or select **Finish** to begin the import.

Figure 8–9 Field Properties Page for the Flat File Sample Wizard



This image displays the field properties page for the Flat File Sample wizard. At the top is a field names and the field properties table. This table contains the following columns, ordered from left to right: Name, Type, Length, Precision, Scale, Mask, NULLIF, DEFAULTIF, and SQL Type. Other columns are currently not displayed in this image. Below the table is a File box. At the top is the "Use the first record as the field names" check box. Below the check box is a table showing the information available in the .dat file. At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page, the following buttons, ordered from left to right: Back, Next, Finish, and Cancel.

****

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, SQL*Loader and the properties you set here are used. 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 about SQL*Loader field and data types, see *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 8-25.
- **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 17-34.

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, the default is updated 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 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 8–1 shows an example of what the record type values for the two sample records earlier might be:

Table 8–1 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 act like regular source tables. When you create and define an external table, the metadata for the external table is saved in the workspace. 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 workspace by one of two methods described in "Importing ASCII Files into the Workspace" on page 8-1 and "Adding Existing Binary Files to the Workspace" on page 8-2.

To create a new external table definition:

1. From the 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. The wizard lists all the flat files available in the workspace. 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" on page 8-1.

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 list 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 box.

2. Use the Select the Object to synchronize list to specify the flat file with which the external table is to be synchronized.

   By default, the flat file that was used to create the external table is displayed in this 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 box.

4. Use the **Match Strategy** list to specify how the search is performed for matches and the external table with the information from the flat file is updated. 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** list to indicate how differences in metadata between the existing external table definition and the record you specified are handled.

   Merge: The metadata from the existing external table definition and the record you specified is combined.

   Replace: The metadata is deleted 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 box is closed and you return to the Reconcile dialog box.

7. Click OK to complete synchronizing the external table definition.

Editing External Table Definitions

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 box is displayed. The tabs and properties that you can edit depend on how you defined the external table in the workspace.

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 13-6.

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 13-27.

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 8-27.

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** 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 23-42.

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 workspace. In this case, you can view and edit the access parameters.
- You created an external table in an Oracle Database and imported its definition. 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 the external table is generated and deployed, it is not recommended. Warehouse Builder does not validate the changes. For more information about 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 workspace 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 toolbar.

   The Configuration Property window is displayed.
3. To configure a property, click the white space and make a selection from the list.

Access Specification

If you imported the external table into the workspace 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, the Oracle Database is directed 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, the Oracle Database is directed 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, the Oracle Database is directed 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 workspace 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 it is assumed that 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, that endian is used.
- **String Sizes in:** This property indicates how data with multibyte character sets, such as UTF16, is handled. By default, the lengths for character strings in the data file are assumed to be 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 workspace 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 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 13-43 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 data file such as *DATAFILE1.* Your entry displays as a new node in the right panel of the Configuration Properties dialog box.

2. Type in the following values for each data file 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*.

9

Using Microsoft Products as Sources

Warehouse Builder enables you to source data stored using Microsoft products. This chapter discusses using Microsoft Excel and Microsoft SQL Server as sources.

Using Excel Spreadsheets as Sources

Scenario

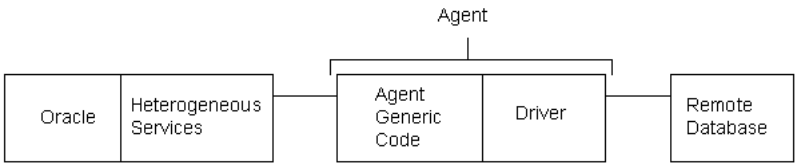
A company stores its employee data in an Excel file called `employees.xls`. This file contains two worksheets: `employee_details` and `job_history`. You need to load the data from the `employee_details` worksheet into a target table in Warehouse Builder.

Solution

To load data stored in an Excel file into a target table, you must first use the Excel file as a source. Warehouse Builder enables you to source data stored in a non-Oracle source, such as Microsoft Excel, using the Heterogeneous Services component of the Oracle Database.

Figure 9–1 describes how the Oracle Database uses Heterogeneous services to access a remote non-Oracle source.

Figure 9–1 Heterogeneous Services Architecture



This screenshot shows the Heterogeneous Services architecture. This screenshot is divided into three main boxes. The first box is divided into two parts: Oracle and Heterogeneous Services. The second box is divided into two parts: Agent Generic Code and Driver. The third box contains the Remote Database. All the three boxes are connected to each other.

****

The Heterogeneous Services component in the database communicates with the Heterogeneous Services agent process. The agent process, in turn, communicates with the remote database.

The agent process consists of agent-generic code and a system-specific driver. All agents contain the same agent-generic code. But each agent has a different driver depending on the type of data being sourced.

Case Study

This case study shows you how to use an Excel file called `employees.xls` as a source in Warehouse Builder.

Step 1: Install ODBC Driver for Excel

To read data from Microsoft Excel, you must have the ODBC driver for Excel installed.

Step 2: Delimit the Data in the Excel File (Optional)

To source data from an Excel file, define a name for the range of data being sourced:

1. In the `employee_details` worksheet, highlight the range that you want to query from Oracle.

   The range should include the column names and the data. Ensure that the column names confirm to the rules for naming columns in the Oracle Database.

2. From the Insert menu, select **Name** and then **Define**. The Define Name dialog box is displayed. Specify a name for the range.

Step 3: Create a System DSN

Set up a System Data Source Name (DSN) using the Microsoft ODBC Administrator.

1. Select **Start**, **Settings, Control Panel, Administrative Tools, Data Sources (ODBC).**

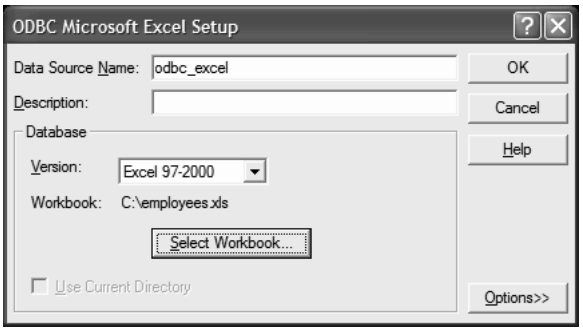
   This opens the ODBC Data Source Administrator dialog box.

2. Navigate to the System DSN tab and click **Add** to open the Create New Data Source dialog box.

3. Select **Microsoft Excel Driver** as the driver for which you want to set up the data source.

   Click **Finish** to open the ODBC Microsoft Excel Setup dialog box as shown in Figure 9–2.

Figure 9–2 ODBC Microsoft Excel Setup Dialog Box



This screenshot displays the ODBC Microsoft Excel Setup dialog box. To the left is the Data Source Name field. To the right of the Data Source name is the OK button.

On the left, below the Data Source Name field, is the Description field. To the right of the Description field is the Cancel button. Below the Cancel button is the Help button.

Below the Description field is the Database box. This box contains the Version list, in which Excel 97-2000 is currently displayed. Below the Version list is the read-only field Workbook. To the right of this is the location of the Excel file. Below the location is the Select Workbook button. Below the Select Workbook button is the check box Use Current Directory.

At the lower right corner of the window is the Options button.

****

4. Specify a name for the data source. For example, `odbc_excel`.
5. Click **Select Workbook** to select the Excel file from which you want to import data.
6. Verify that the Version field lists the version of the source Excel file accurately.

Step 4: Create the Heterogeneous Services Initialization File

To configure the agent, you must set the initialization parameters in the heterogeneous services initialization file. Each agent has its own heterogeneous services initialization file. The name of the Heterogeneous Services initialization file is `init`*`SID`*`.ora`, where *`SID`* is the Oracle system identifier used for the agent. This file is located in the *`ORACLE_HOME`*`/hs/admin` directory.

Create the `initexcel.ora` file in the *`ORACLE_HOME`*`/hs/admin` directory as follows:

```
HS_FDS_CONNECT_INFO = odbc_excel
HS_AUTOREGISTER = TRUE
HS_DB_NAME = hsodbc
```

Here, `odbc_excel` is the name of the system DSN you created in Step 3. `excel` is the name of the Oracle system identifier used for the agent.

Step 5: Modify the listener.ora file

Set up the listener on the agent to listen for incoming requests from the Oracle Database. When a request is received, the agent spawns a Heterogeneous Services agent. To set up the listener, modify the entries in the `listener.ora` file located in the *`DATABASE_ORACLE_HOME`*`/network/admin` directory as follows:

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = excel)
      (ORACLE_HOME = C:\oracle11g\product\11.1.0\db_1)
      (PROGRAM = hsodbc)
    )
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = C:\oracle11g\product\11.1.0\db_1)
      (PROGRAM = extproc)
      )
  )
```

1. For the `SID_NAME` parameter, use the `SID` that you specified when creating the initialization parameter file for the Heterogeneous Services, which, in this case, is `excelsid`.
2. Ensure that the `ORACLE_HOME` parameter value is the path to your Oracle Database home directory.

3. The value associated with the PROGRAM keyword defines the name of the agent executable.

Remember to restart the listener after making these modifications.

> **Note:** Ensure that the initialization parameter GLOBAL_NAMES is set to FALSE in the database's initialization parameter file. FALSE is the default setting for this parameter.

Step 6: Create an ODBC Source Module

Use the following steps to create an ODBC source module:

1. From the Project Explorer, create an ODBC source module.

   On the navigation tree, ODBC is listed within the Non-Oracle node under the Databases node.

2. You can provide the connection information for the source location either at the time of creating the module, or while importing data into this module.

3. To provide connection information while creating the module, on the Connection Information page, click **Edit** and provide the following details:

   Ensure that the service name you provide is the same as the SID_NAME you specified in the listener.ora file.

   Provide the host name and the port number using the Host Name and Port number fields respectively.

   Because you are not connecting to an Oracle database, you can provide dummy values for user name and password. The fields cannot be empty.

   The Schema field can be left empty because you will not be importing data from a schema.

Step 7: Import Metadata from Excel Using the Metadata Import Wizard

Use the Metadata Import Wizard to import metadata from the Excel file into Warehouse Builder. Select **Tables** as the Filter condition. The wizard displays all the worksheets in the source Excel file under the Tables node in the list of available objects.

1. Select **employee_details** and use the right arrow to move it to the list of selected objects.

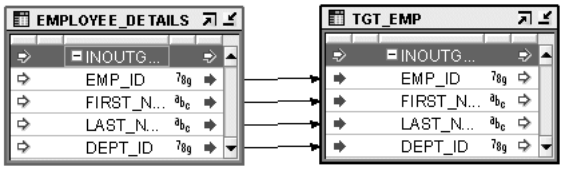
2. Click **Finish** to import the data.

   The data from the employee_details worksheet is now stored in a table called employee_details in the ODBC source module.

Step 8: Create a Mapping to Load Data Into the Target Table

In the Warehouse Builder console, expand the module that contains the target table. Use the table called employee_details in the ODBC source module as a source to load data into the target table. Figure 9–3 displays the mapping used to load data into the target table.

Figure 9–3 Mapping to Load Data Into the Target Table



This screenshot displays the mapping that loads data into the target table. The the left is the EMPLOYEE_DETAILS table which contains a collapsible INOUTGRP1 with the following rows: EMP_ID, FIRST_NAME, LAST_NAME, and DEPT_ID. To the right is the TGT_EMP table which contains a collapsible INOUTGRP1 with the following rows: EMP_ID, FIRST_NAME, LAST_NAME, and DEPT_ID. All the rows in the EMPLOYEE table are connected to the respective rows in the TGT_EMP table

****

Step 9: Deploy the Mapping

Use the Control Center Manager or Design Center to deploy the mapping you created in step 8. Ensure that you first deploy the source module before you deploy the mapping.

Troubleshooting

This section lists some of the errors that you may encounter while providing the connection information.

Error

```
ORA-28546: connection initialization failed, porbable Net8 admin error

ORA-28511: lost RPC connection to heterogeneous remote agent using
SID=(DESCRIPTION=(ADDRESS_
LIST=(ADDRESS=(PROTOCOL=TCP)(Host=localhost)(PORT=1521)))(CONNECT_
DATA=(SID=oracledb)))

ORA-02063: preceeding 2 lines from OWB###
```

Probable Cause

Providing the same SID name as that of your database.

Action

Provide an SID name different from the SID name of your database.

Error

```
ORA-28500: connection from ORACLE to a non-Oracle system returned this message:
[Generic Connectivity Using ODBC][H006] The init parameter <HS_FDS_CONNECT_INFO>
is not set. Please set it in init<orasid>.ora file.
```

Probable Cause

Name mismatch between SID name provided in the `listener.ora` file and the name of the `init`*`SID`*`.ora` file in *`ORACLE_HOME`*`/hs/admin`.

Action

Ensure that the name of the `initSID.ora` file and the value provided for the `SID_NAME` parameter in `listener.ora` file is the same.

> **Tip:** Ensure that you restart the listener service whenever you make changes to the `listener.ora` file.

Using SQL Server as a Source

Scenario

Your company has data that is stored in SQL Server and you would like to import this into Warehouse Builder. Once you import the data, you can perform data profiling to correct anomalies, and then transform the data according to your requirements by using mappings.

Solution

In Warehouse Builder, you can connect to non-Oracle data sources. Once connected, you can import metadata just as from any Oracle data source.

Case Study

To connect to SQL Server and import metadata, refer to the following sections:

1. "Creating an ODBC Data Source" on page 9-6
2. "Configuring the Oracle Database Server" on page 9-7
3. "Adding the SQL Server as a Source in Warehouse Builder" on page 9-8
4. What's Next on page 9-8

If you encounter problems implementing this solution, see "Troubleshooting" on page 9-8.

Creating an ODBC Data Source

You must create an ODBC data source to connect to the SQL Server. To do this, you must set up a System Data Source Name (DSN):

1. Select **Start**, **Control Panel, Administrative Tools, Data Sources (ODBC).**

   This opens the ODBC Data Source Administrator dialog box.
2. Navigate to the System DSN tab and click Add to open the Create New Data Source dialog box.
3. Select **SQL Server** as the driver for which you want to set up the data source.
4. Click **Finish** to open the Create A New Data Source to SQL Server Wizard.
5. In the **Name** field, specify a name for the data source. For example, `sqlsource`.
6. In the **Server** field, select the server to which you want to connect and click **Next**.
7. Specify whether the authentication should be done at the Operating System level or at the server level. Click **Next**.
8. Select the database file and click **Next**.
9. Accept the default values in the next screen and click **Finish**.
10. Test the data source to verify the connection.

Configuring the Oracle Database Server

Next, you must configure Oracle Database to connect to SQL Server. Warehouse Builder can then use this configuration to extract metadata from the SQL Server. There are two steps involved in this:

- Creating a Heterogeneous Service Configuration File
- Editing the listener.ora file

Creating a Heterogeneous Service Configuration File

You must create the heterogeneous file in the *ORACLE_HOME*\hs\admin directory. The naming convention for this file should be as follows:

- Must begin with init
- Must end with the extension .ora
- Must not contain space or special characters

For example, you can name the file initsqlserver.ora.

Enter the following in the file:

```
HS_FDS_CONNECT_INFO = sqlsource
HS_FDS_TRACE_LEVEL = 0
```

Here, sqlsource is the name of the data source that you specified while creating the ODBC data source.

Editing the listener.ora file

You must add a new SID description in the listener.ora file. This file is stored in the *ORACLE_HOME*/network/admin directory.

Modify the file as shown:

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = sqlserver)
      (ORACLE_HOME = c:\oracle10g\oracle_home)
      (PROGRAM = hsodbc)
    )
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = c:\oracle10g\oracle_home)
      (PROGRAM = extproc)
      )
  )
```

The SID_NAME parameter must contain the name of the configuration file you created in the previous step. However, it must not contain the init prefix. For example, if the configuration file you created in the previous step was initsqlserver.ora, then the value of the SID_NAME parameter should be sqlserver.

ORACLE_HOME must point to the Oracle home location of your database installation.

The value associated with the PROGRAM keyword defines the name of the executable agent, which, in this case, is hsodbc.

Restart the listener service after making these modifications.

Adding the SQL Server as a Source in Warehouse Builder

The final step involves adding an ODBC module in Warehouse Builder, and importing the data from the SQL server into this module.

To add an ODBC source module in Warehouse Builder:

1. Within a project in the Project Explorer, navigate to the Databases, Non-Oracle node.
2. Right-click **ODBC** and select **New**.
3. Create a new ODBC module using the Create Module Wizard.
4. You can provide the connection information for the source location either at the time of creating the module, or while importing data into this module.
5. In the Edit Location dialog box, make sure that you enter User Name and Password within double quotation marks ("). For example, if the user name is mark, enter "mark".
6. For Service Name, enter the SID name you provided in the `listener.ora` file. Also select the schema from which you wish to import the metadata.

To import metadata into the ODBC module:

1. Right-click the module and select **Import**.
2. Import the metadata using the Import Metadata Wizard.

   The tables and views available for import depend on the schema you selected when providing the connection information.

What's Next

After you successfully import metadata into Warehouse Builder, you can use the data profiling functionality to check the quality of the data. Or you can skip data profiling and proceed with designing a mapping to extract, transform, and load the data.

Troubleshooting

Some of the errors that you may encounter while providing the connection information are listed here:

Error

```
ORA-28500: connection from ORACLE to a non-Oracle system returned this message:
[Generic Connectivity Using ODBC][Microsoft][ODBC Driver Manager] Data source name
not found and no default driver specified (SQL State: IM002; SQL Code: 0)

ORA-02063: preceding 2 lines from OWB_###
```

Probable Cause

Creating the DSN from the User DSN tab.

Action

Create the DSN from the System DSN tab.

Error

```
ORA-28500: connection from ORACLE to a non-Oracle system returned this message:
[Generic Connectivity Using ODBC][Microsoft][ODBC SQL Server Driver][SQL
```

```
Server]Login failed for user 'SA'. (SQL State: 28000; SQL Code: 18456)

ORA-02063: preceding 2 lines from OWB_###
```

Probable Cause

The user name and password in the Edit Location dialog box are not enclosed within double quotation marks.

Action

Enter the user name and password within double quotation marks.

> **Tip:** Make sure that you restart the listener service whenever you make changes to the `listener.ora` file.

10

Integrating Metadata Using the Transfer Wizard

There are several utilities for sharing and/or integrating metadata with other tools and systems. This section discusses 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 Transfer Wizard.

This section contains these topics:

- Using the Transfer Wizard
- Integrating with the Meta Integration Model Bridges (MIMB)
- Importing Metadata
- Transfer Considerations
- Importing Metadata from an OMG CWM-Standard System

Using the Transfer Wizard

The Transfer Wizard enables you to synchronize, integrate, and use metadata stored in OMG CWM formats. Use the Transfer Wizard to import selected metadata from OMG CWM compliant applications version 1.0 into the workspace. You can also import metadata that is stored in a proprietary format by integrating with MIMB (Meta Integration Model Bridges). For more information about the formats that MIMB can translate, see "Integrating with the Meta Integration Model Bridges (MIMB)" on page 10-1.

The Transfer Wizard can be used only in single-user mode. This means that no other users can be connected to the workspace into which you are importing OMG CWM metadata. Before you use the Transfer Wizard to import OMG CWM metadata into a workspace, ensure that you are the only user connected to that workspace.

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 workspace to the standard CWM format. You can then import this metadata 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 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 DB2
- IBM Rational Rose
- Informatica PowerCenter
- Merant AppMaster Designer 4.0
- Microsoft SQL Server
- Microsoft Visio
- Microsoft Visual Studio
- MicroStrategy
- NCR Teradata
- Oracle Designer
- Popkin System Architect
- ProActivity 3.x & 4.0
- SAS ETL Studio
- Select SE 7.0
- Silverrun RDM
- 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, see the MIMB documentation at *http://www.metaintegration.net*.

Follow these steps to integrate with MIMB.

Download the Meta Integration Model Bridge

After you download the MIMB on your system, the installation is automatically recognized and the new import options are displayed on the Metadata Source and Target Identification Page of the Transfer Wizard.

To download MIMB:

1. Download the Model Bridge (personal) product from the following Web site: *http://www.metaintegration.net/*.
2. Install the MIMB by running the setup on your system.
3. During installation, select **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 computer, 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 by starting the Transfer Wizard from the Design Center. For more information about importing metadata using this wizard, see "Importing Metadata" on page 10-3.

Importing Metadata

The Transfer Wizard enables you to import OMG CWM metadata. Perform the following steps to import metadata:

1. Run the Transfer Wizard.

   The Transfer Wizard converts the OMG CWM compliant metadata and stores it in an MDL file. For more information about using the Transfer Wizard, see "Running the Transfer Wizard" on page 10-3.
2. Import the MDL file created in Step 1 using the Metadata Loader Import Utility.

   For more information importing the MDL file, see "Importing the MDL File" on page 10-5.

Running the Transfer Wizard

From the **Design** menu of the Design Center, select **Import**, and then **Bridges**. Warehouse Builder displays the Welcome page of the Transfer Wizard. This page lists the steps you perform while using the 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 box. Click **Next** to proceed.

Provide details on the following pages of the 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 10-1.

The **To** field contains the value OWB mdl and you cannot edit 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 10–1 describes the transfer parameters used by the Transfer Wizard.

Table 10–1 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 project into which the OMG CWM model will be imported.
Warehouse Builder MDL	Specify the name of the .mdl file that will store the converted source metadata.
Default Module Type	Select the type of module into which you want to import the metadata. The options are **Source warehouse module** and **Target warehouse module**. The Transfer Wizard will create the correct module and import the metadata into it. When you select the Source warehouse module option, the names are case-sensitive. When you select 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 select 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.
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 select in the 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 box is displayed. For details about this dialog box, see "Oracle WB Transfer Dialog Box" on page 10-5.

Oracle WB Transfer Dialog Box This dialog box 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 box 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 Transfer Wizard displays a reminder informing you that you must import the .mdl file into the workspace.
- 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 box. 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 box, click **Return to Wizard** and update the incorrect information about 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 box, 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 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 Transfer Wizard is 10.1.0.3.

Importing the MDL File

After the Transfer Wizard converts the source metadata and stores it in an .mdl file, you must import this .mdl file into the workspace. The Metadata Import Utility enables you to import an MDL file.

> **See Also:** ""Importing and Exporting Using the Metadata Loader (MDL)" in the *Oracle Warehouse Builder Installation and Administration Guide*.

Use the following steps:

1. From the **Design** menu, select **Import**, and then **Warehouse Builder Metadata**.

   The Metadata Import dialog box 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 box is displayed. The File Name field contains the name of the .mdl file that stores the upgraded metadata.

4. Click **Upgrade**.

   The Metadata Import Progress dialog box is displayed. The text above the progress bar displays the status of the import.

   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 workspace, 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 objects.

Importing Metadata from an OMG CWM-Standard System

OMG stands for Object Management Group. An OMG CWM-standard system may have more than one implementation of a cube. In such cases, the Transfer Wizard imports the first implementation of the cube. The log file of the import contains the details of the version of the cube implementation that is imported.

Table 10–2 lists the object conversion from an OMG CWM file system into the workspace.

Table 10–2 Object Conversion from OMG CWM to OWB Workspace

Object in OMG CWM	Object Imported to Warehouse Builder
Package	Project
Schema	Module
Dimension	Dimension
Level	Level
Attribute	Level Attributes
Hierarchy	Hierarchy
Cube	Cube
Measure	Cube Measure
Table	Table
Column	Column

Table 10–2 (Cont.) Object Conversion from OMG CWM to OWB Workspace

Object in OMG CWM	Object Imported to Warehouse Builder
Foreign Key	Foreign Key
Unique Constraint/Primary Key	Unique Key
View	View
Column	Column

Table 10–3 lists the data type conversion from an OMG CWM system to a workspace.

Table 10–3 Data Type Conversion from OMG CWM to OWB Workspace

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`

11

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.

This chapter includes the following topics:

- Integrating with Business Intelligence Tools
- Using Business Definitions
- Using the Data Object Editor with Business Intelligence Objects
- Configuring Business Intelligence Objects
- Accessing Business Intelligence Objects Using Oracle BI Discoverer

Integrating with Business Intelligence Tools

Warehouse Builder introduces Business Intelligence (BI) objects that enable you to integrate with Oracle Business Intelligence tools such as Discoverer. You can define BI objects in Warehouse Builder that enable you to store definitions of business views. You can then deploy these definitions to the Oracle Business Intelligence tools and extend the life-cycle of your data warehouse. The method you use to deploy business definitions depends on the version of Discoverer to which you want to deploy and the Warehouse Builder licensing option you purchased. For more information, see "Deploying Business Definitions to Oracle Discoverer" on page 25-5.

This section contains the following topics:

- Introduction to Business Intelligence Objects in Warehouse Builder
- Introduction to Business Definitions
- About Business Definitions in Warehouse Builder

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. 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. These definitions are stored under the Business Intelligence node on the Warehouse Builder Project Explorer.

The Business Intelligence node contains an additional node called Business Definitions. You start by first creating a Business Definition module to store the definitions to be deployed to Discoverer. For details, see "About Business Definitions in Warehouse Builder" on page 11-2.

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
- 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 workspace
- Ability to utilize Warehouse Builder metadata management features such as snapshots, multilanguage support, and command-line interaction

About Business Definitions in Warehouse Builder

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.

See Also:

- "Deriving Business Intelligence Objects" on page 11-22
- "Deploying Business Definitions to Oracle Discoverer" on page 25-5

Using Business Definitions

Business definitions are the equivalent of Discoverer End User Layer (EUL) objects and enable you to integrate with Oracle Discoverer. Business definitions facilitate data analysis of data stored in your data warehouse. You can define and then deploy business objects to Oracle Discoverer.

Note: The method of deploying business definitions depends on the Warehouse Builder licensing option you purchased and the version of Oracle Discoverer to which you want to deploy business definitions. For more details, see "Deploying Business Definitions to Oracle Discoverer" on page 25-5.

You can create business definitions or derive them from existing schemas. For information about creating business definitions, see "Creating Business Definitions" on page 11-4. For information about deriving business definitions, see "Deriving Business Intelligence Objects" on page 11-22.

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:

1. From the 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 13-6.

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** 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.** The Edit Discoverer Location dialog box is displayed. Provide the required information to connect with your target system and click **OK.** For more information about the Edit Discoverer Location dialog box, see "Defining Discoverer Locations" on page 11-4.

> **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 box:

- **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 select SQL*Net Connection, specify the net service name in the **Net Service Name** field. When you select 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 to Oracle Discoverer" on page 25-5.

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 box.

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 the Business Definition module is created, 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:

- 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: Simple and Complex. Simple Item Folders contain items from exactly one table in your workspace. 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 workspace, as described in "Deriving Business Intelligence Objects" on page 11-22. 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:

- Editing an Item Folder
- Creating an Item Folder
- Creating a Business Area

Editing an Item Folder

After you derive your design object definitions, an Item Folder is created 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 guidelines below.

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 left arrow to return the items from the Selected column to the Available column. You then use the right arrow 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 left arrow 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 box. This dialog box contains a subset of the options in the Expression Builder. Use the Formula dialog box to create your calculation. This field is populated after you close the Formula dialog box. For more information about the Expression Builder, see "The Expression Builder User Interface" on page 18-4.
- **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** list contains Lists of Values, Alternative Sort Orders, and Drills to Detail. You can also remove a reference to an Item Class.

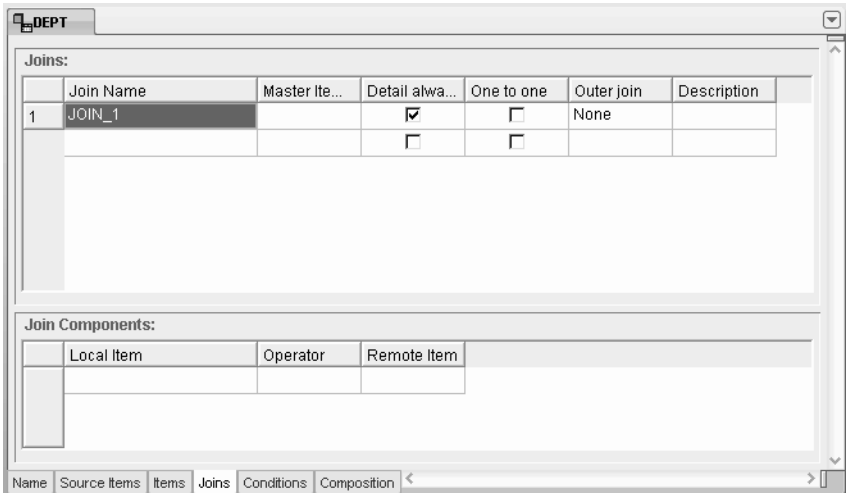
- **Default Position:** Select the position of this Item on a Discoverer report.
- **Default Aggregate:** Indicate if the Item will default to an aggregate in the Discoverer report.
- **Heading:** The title for the Item in a Discoverer report.
- **Format Mask:** The format mask for this Item when it 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 file names, 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 data types.

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 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 11–1 Creating and Editing Joins



This image displays the Item Folder Details. At the top are four tabs ordered from left to right: Name, Source Items, Items, Joins, Conditions. The Joins tab is currently displayed. Join tab is used to define Joins. Below the tab is the JOINS table, containing the following columns, ordered from left to right: Join Name, Master Item Folder, Detail always has master, One to one, Outer join, Description. One row is displayed. Below the Join table is the Join Components table. This table contains the following columns, ordered from left to right: Local Item, Operator, and Remote Item. The Join Components table defines the join condition.

****

On the Joins page, click a row in the Specify the Joins field. Provide the following information:

- **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 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:** This list is populated 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.

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 about the Expression Builder, see "The Expression Builder User Interface" on page 18-4.
- **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, Item Folders are created as part of the derived business definitions. However, if you want to define a customized Item Folder, you can 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 11-6.

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 about how you specify the details on each tab, refer to the description of these tabs in the "Editing an Item Folder" section on page 11-6.

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 box is displayed. Follow the steps listed in "Steps to Create an Item Folder" on page 11-11.

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 box is displayed. Follow the steps listed in "Steps to Create an Item Folder" on page 11-11.

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 list, select the name of the Business Definition module to which the Item Folder should belong.
4. In the **Item Folder Type** list, 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 about the contents of these tabs, see "Editing an Item Folder" on page 11-6.

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 workspace 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 workspace, 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 using the right arrow 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 using the Data Object Editor for an Item Folder. For more information about creating joins, see "Joins Tab" on page 11-8.

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, the Join Selection page is displayed. The 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 11-6.

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 allows 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 about the contents of these tabs, refer to the description of these tabs in the "Editing a Business Area" section on page 11-13.

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 box 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 box 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, all the Item Folders available within the Business Definition module are displayed. You can multi-select the Item Folders by pressing the Ctrl key and using the right arrow 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 the Business Area is created, 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, use the Edit Business Area dialog box. For details, see "Editing a Business Area" on page 11-13.

Editing a Business Area

Warehouse Builder enables you to edit the definitions for a Business Area using the Edit Business Area dialog box.

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 box containing two tabs: Name and Item Folders. Edit these tabs as follows:

Editing the Business Area Name

The Name tab allows 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 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, etc. 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

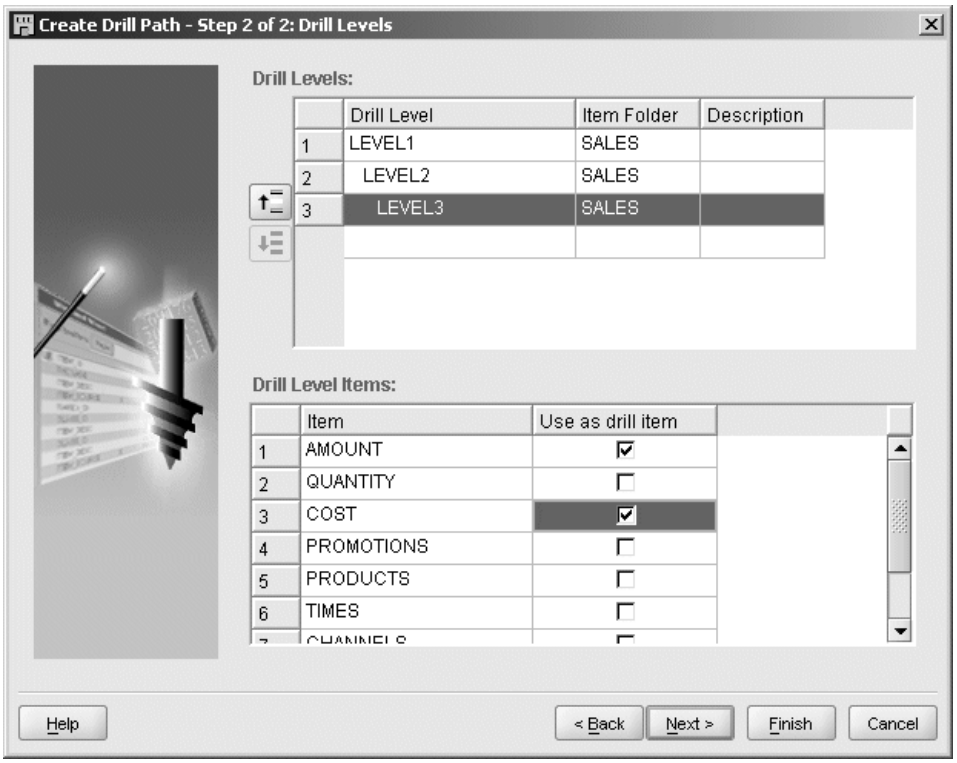
Use the Drill Levels page to define a drill level and specify the Item Folder it references. Optionally, you can provide a description for the Drill Levels. To define drill levels, click a row and provide the following information:

- **Drill Level:** Type a name for the drill level.
- **Item Folder:** From the 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.

Figure 11–2 Creating Drill Levels



This image displays the Drill Levels page of the Create Drill Path Wizard. The Drill Levels section displays the drill levels in a table. The table contains the following columns, ordered from left to right: Drill Level, Item Folder, and Description. Below the table is the Drill Level Items section. This section contains a Drill Level Items table with two columns, ordered from left to right, Item and Use as drill item. At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page, the following buttons, ordered from left to right: Back, Next, Finish, and Cancel.

****

Specifying the Join

If there are more than one join paths between the Item Folders referenced by the drill levels, the Join Selection page is displayed. The list displays 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 allows you to edit a drill path using the Edit Drill Path dialog box.

Editing a Drill Path

Warehouse Builder enables you to edit drill paths using the Drill Path using the Edit Drill Path dialog box.

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 box 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 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 list to view a report on the quantities of a product sold in four specific countries.

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, an Item Class that you can use both as a List of Values and as a Drill to Detail is created.

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 workspace. 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 workspace. Expand the nodes to select the items that will reference your list of values. Use the right arrow 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 allows you to edit the name, description, and defining and referencing items associated with an LOV using the Edit List of Values dialog box.

Editing Lists of Values

Warehouse Builder enables you to edit a list of values using the Edit List of Values dialog box.

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 box 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 right arrow 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 left arrow 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 box 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, an Item Class that can be used both as an Alternative Sort Order and as a Drill to Detail is created.

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 workspace. Expand the nodes to select the items that will reference your Alternative Sort Order. Use the right arrow 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 allows you to edit the name, description, and defining and referencing items associated with an alternative sort order using the Edit dialog box.

Editing Alternative Sort Orders

The Edit Alternative Sort Order dialog box 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 box 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 right arrow 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 left arrow.

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 box 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, 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 using the Create Drill to Detail dialog box.

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 box.

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 box lists the Item Folders in the Business Definition Module. Select a referencing item from this set and use the right arrow to move it to the **Selected** column.

Editing Drills to Detail

Use the Edit Drill to Detail dialog box 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 box is displayed. The contents of this dialog box are the same as the Create Drill to Detail dialog box. 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 box, see "Create Drill to Detail" on page 11-20.

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 object 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 guidelines below.

Naming the Registered Function

In the Name and Description page, type a name and optional description for the alternative sort order.

From the **Select the return type of the function** list, select a return type for the function. Select **Available to User** 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 **Type** list, select the data type for the parameter. Use the **Description** field to 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 allows you to edit the name, description, and parameters of the function using the Edit dialog box.

Editing Registered Functions

Use the Edit Registered Function dialog box 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 box 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 data type. Delete a

parameter by right-clicking the gray 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:

1. From the 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 guidelines below.

You can also start 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 guidelines below.

Selecting Source Objects

The Source Objects page allows you to select additional objects for derivation. The **Available** column displays all the objects available in your workspace 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 right arrow 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, shortcuts are created 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** list. 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 **Show advanced parameters** 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 list. 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 to validate the selected objects before deriving them.
- **Abort on error:** Check the box if you want 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.

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 11-25.
- Add Item Folders to a Business Area, see Create "Adding Item Folders to a Business Area" on page 11-25.
- Create Item Folders, see "Creating Item Folder Using the Data Object Editor" on page 11-26.
- Add Items to an Item Folder, see "Adding Items to An Item Folder" on page 11-26.

- Synchronize Item Folders with the underlying database table, see "Synchronizing Item Folders" on page 11-27.
- Create joins, see "Creating Joins Using the Data Object Editor" on page 11-28.

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 box.
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** 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 11-25.

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 box is displayed.
4. Choose **Select an existing Item Folder**.
5. From the selection tree, select 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 box 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** 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** 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 box is displayed.
8. Use this dialog box 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

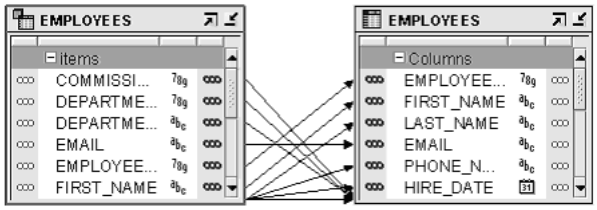
You can use the Data Object Editor to add items to an Item Folder. Follow these steps:

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. 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 11–3 Item Folder and its Source Object



This image displays the Detail view of an Item Folder. The Item Folder on the left contains items that are stored in the object on the right.

****

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 Builder displays the Add a New or Existing <Object> dialog box.

4. Select the Select an existing <Object> option.

5. From the selection tree, select 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 box is displayed.

4. Review the details displayed on this dialog box and click **OK**.

   Warehouse Builder synchronizes the item folder with the data object on which the item is based.

Synchronize Item Folder Dialog Box

The Synchronize Item Folder dialog box 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 box displays the details of the changes to be made to the Item Folder.

The Synchronize Item Folder dialog box 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 to synchronize the Item Folder. The available actions are Update and None. If you select 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 box 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 box and update the Item Folder definition. If you do not wish to perform the actions listed on this dialog box, 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 box.

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 box.

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.

Configuring Business Intelligence Objects

During the design phase, you create definitions for the business intelligence 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 box is displayed. Click the object name on the left side of this dialog box 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 object matching during deployment to Discoverer should be performed. When you deploy business definitions, an .eex file is first created and then this file is imported 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.

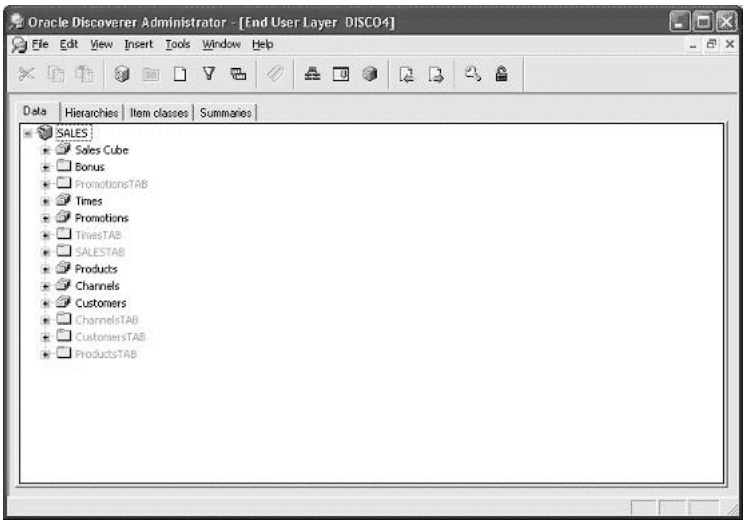
Accessing Business Intelligence Objects Using Oracle BI Discoverer

Once you successfully deploy the business intelligence objects that you create, these objects are available in Oracle BI Discoverer. 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, 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.

Figure 11–4 Discoverer Administrator Showing Business Intelligence Objects



This image displays the Discoverer Administrator interface that shows the Business Intelligence Objects that you deployed. At the top of the window is the menu bar. Below the menu bar is the toolbar. Below the toolbar are four tabs ordered from left to right: Data, Hierarchies, Item classes, and Summaries. The Data tab is currently displayed. The Data tab contains information regarding SALES in a tree format.

****

You can now use Oracle BI Discoverer to create reports based on the business intelligence objects that you display.

Part II

ETL and Data Quality

This part contains the following chapters:

12

Designing Target Schemas

Warehouse Builder is also a design tool that enables you to design your data warehouse. Target schemas contain all the necessary data objects in your data warehouse 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. You can design target schemas, both relational and dimensional, using the Data Object Editor.

This chapter includes the following topics:

- Designing the Target Schema
- Configuring Data Objects
- Validating Data Objects
- Generating Data Objects

Designing the Target Schema

To create a target schema, you create any of the dimensional or relational objects listed in Table 13–1 on page 13-2. You can design a relational target schema or a dimensional target schema. In this section, the term *dimensions* refers to both regular dimensions and Slowly Changing Dimensions (SCDs).

Designing a Relational Target Schema

A relational target schema is one that contains relational data objects such as tables, views, materialized views, and sequences. All the warehouse data is stored in these objects.

To design a relational target schema:

1. If you have not already done so, create an Oracle module that will contain the objects for your target schema. Ensure that the location associated with this module refers to the target schema.
2. Create the relational data objects.

   Relational objects that you can create include tables, views, materialized views, and sequences. You can create additional structures pertaining to relational objects such as constraints, indexes, and partitions. You may have already imported some existing target objects. To create additional data objects, refer to "Defining Oracle Data Objects" on page 13-1.

   Note that this step only creates the definitions of the objects in the workspace. To create the objects in the target schema, you must deploy these objects.

3. Configure the data objects.

   In this step, you set the physical properties of the data objects. For example, you specify the name of the tablespace in which a table should be created. Each data object has a set of default configuration properties. You can choose to modify these default values.

   See "Configuring Data Objects" on page 12-3.

4. Validate the data objects.

   Validation verifies the metadata definitions and configuration properties of data objects. Correct any errors that are encountered during the validation.

   See "Validating Data Objects" on page 12-3.

5. Generate code that will create these data objects in the target schema.

   Generation produces code that is required to create the data objects created in step 2 in the target schema.

   See "Generating Data Objects" on page 12-5.

Designing a Dimensional Target Schema

A dimensional target schema uses dimensional objects to store the data warehouse data. Dimensional objects include dimensions and cubes. Dimensional objects transform the visualization of the target schema from a table-oriented environment to a more business-focussed environment. This helps you obtain answers to complex analytical queries quickly and more efficiently.

To design a dimensional target schema:

1. If you have not already done so, create the Oracle module that will contain your dimensional objects. Ensure that the location associated with this module refers to the target schema.
2. Create the dimensions required in your target schema.

   See "Creating Dimensions" on page 14-32.

3. Create time dimensions.

   Data warehouses use time dimensions extensively to store temporal data. See "Creating Time Dimensions" on page 14-64.

4. Create the cubes required for the target schema.

   See "Creating Cubes" on page 14-49.

5. Configure the dimensions and cubes.

   Configure the dimensional objects you created in steps 2, 3, and 4 to set physical properties for these objects. You can accept the default properties or modify them.

   See "Configuring Data Objects" on page 12-3.

6. Validate the dimensions and cubes.

   In this step, you verify the metadata definitions and configuration properties of the dimensional objects created in steps 2, 3, and 4. Correct any errors resulting from the validation.

   See "Validating Data Objects" on page 12-3.

7. Generate code that will create these dimensions and cubes in the target schema.

See "Generating Data Objects" on page 12-5.

Note: Defining dimensional objects only creates the definitions of these objects in the workspace. To create these objects in the target schema, you must deploy these objects

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. Or you can specify the type of implementation for dimensional objects. 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.

See Also: "Creating Additional Configurations" in the *Oracle Warehouse Builder Installation and Administration Guide*.

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

You can configure objects using the Data Object Editor or the Project Explorer. To configure an object using the Data Object Editor, use the Configuration panel of the editor. This panel displays the configuration details for the object currently selected on the canvas. You can even drill down to, say and index in a table in the Selected Objects tab of the Explorer panel to see those configuration details.

To configure an object using the Project Explorer:

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

   or

   Right-click the object and select **Configure**.

   The Configuration Properties dialog box is displayed.
2. Select a parameter to display its description at the bottom of the right panel.
3. Enter your changes and click **OK**.

Validating Data Objects

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.

You can validate a single object or multiple objects at a time. You can also validate objects that contain objects, such as modules and projects. In this case, all data objects contained by that object are validated. Use the Project Explorer or the Data Object Editor to validate data objects.

When you validate objects, Warehouse Builder displays the Validation Results window that contains the results of the validation. For more information about this dialog box, click **Help** and then **Topic**.

Validating Data Objects Using the Project Explorer

In the Project Explorer, select the data object and click the Validate icon. You can select multiple objects by holding down the **Ctrl** key while selecting objects.

or

In the Project Explorer, select the data object or data objects. To select multiple objects, hold down the **Ctrl** key while selecting objects. Right-click the data object and select **Validate**. If you selected multiple objects, ensure that the **Ctrl** key is pressed when you right-click.

Validating Data Objects Using the Data Object Editor

Right-click the icon representing the data object on the Data Object Editor canvas and select **Validate**.

or

Select the object on the canvas and either click the Validate icon or select **Validate** from the Object menu.

Editing Invalid Objects

The results of validating data objects are displayed in the Validation Results window. From this window, you can access the editors for objects and rectify errors in their definition, if any.

To edit invalid definitions:

1. In the Validation Results window, double-click an invalid object from the tree or from the validation messages grid.

   An editor for the selected object is displayed.

2. Edit the object to correct problems.
3. Close the editor when you are finished and re-validate.

Generating Data Objects

When you generate data objects, Warehouse Builder produces the code required to create the data objects in the target schema. 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.

You can view the generated scripts and also store them to a file system.

When you generate code for a data object, Warehouse Builder first validates the object and then generates code.You may skip the validation step and directly generate code for your data objects. However, it is recommended that you validate objects before you generate them. This enables you to discover and correct any errors in data object definitions before the code is generated.

Use the Project Explorer or the Data Object Editor to generate code for data objects. When you generate objects, Warehouse Builder displays the Generation Results window that contains the results of the generation. For more information about this window, click **Help** and then **Topic**.

Generating Data Objects Using the Project Explorer

To generate a single data object, select the data object and click the Generate icon. Or right-click the data object and select **Generate**.

To generate code for multiple objects, select the objects by holding down the Ctrl key and click the Generate icon. Or select the data objects and, while continuing to hold down the Ctrl key, right-click and select **Generate**.

Generating Objects Using the Data Object Editor

Open the Data Object Editor for the data object by right-clicking the object and selecting **Open Editor**. The canvas displays a node that represents the data object.

Right-click the data object node on the canvas and select **Generate**.

or

Select the data object node on the canvas. Click the Generate icon or select **Generate** from the Object menu.

Viewing Generated Scripts

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 window.
2. Select the Scripts tab on the right of this window.

   The Scripts tab contains a list of the generated scripts for the object you selected.
3. Select a specific script and click **View Code**.

   The selected script displays in a code viewer, which is read-only.

Saving Generated Scripts to a File

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 **Save As**.

   The Save dialog box opens and you can select a location where you want to save the script file.

13

Defining Oracle Data Objects

After you finish designing your data warehouse or data mart, you are ready to design your target system. 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:

- About Data Objects
- About the Data Object Editor
- Using the Data Object Editor to Edit Oracle Data Objects
- Using Constraints
- Using Indexes
- Using Partitions
- Using Tables
- Using Views
- Using Materialized Views
- Using Attribute Sets
- Using Sequences
- Using User-Defined Types
- Configuring Data Objects

About Data Objects

The Oracle module contains nodes for each type of data object that 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.

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. You can also create optional structures associated with relational objects such as constraints, indexes, partitions, and attribute sets.

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 Project Explorer. Intelligence objects enable you to store definitions of business views. You can deploy these definitions to analytical tools such as Oracle Discoverer and perform ad hoc queries on the warehouse. For more information about intelligence objects, see "Integrating with Business Intelligence Tools" on page 11-1.

Table 13–1 describes the types of data objects you can use in Warehouse Builder.

Table 13–1 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 13-25 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 8-25 for more information.
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 13-28 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 13-31 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 13-36 for more information.
Dimensions	Dimensional	A general term for any characteristic that is used to specify the members of a data set. The three most common dimensions in sales-oriented data warehouses are time, geography, and product. Most dimensions have hierarchies. See "About Dimensions" on page 14-8 and "About Time Dimensions" on page 14-23 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 14-28 for more information.

Table 13–1 (Cont.) Data Objects in Warehouse Builder

Data Object	Type	Description
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 an .mdl file using a previous version of the product.
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 "Creating Object Types" on page 13-38 for more information.
Varrays	Relational	A varray is an ordered collection of elements. See "Creating Varrays" on page 13-40 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 "Creating Nested Tables" on page 13-41 for more information.

Supported Data Types

Table 13–2 displays the data types you can use to create and edit columns.

Table 13–2 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.

Table 13–2 (Cont.) Data Types

Data Type	Description
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.
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 data type is like the VARCHAR2 data type, 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 type MDSYS.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.

Table 13–2 (Cont.) Data Types

Data Type	Description
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.
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.

Table 13–2 (Cont.) Data Types

Data Type	Description
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.

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 box. Warehouse Builder maintains a business and a physical name for each object stored in a workspace. 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 3-7 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 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.

About the Data Object Editor

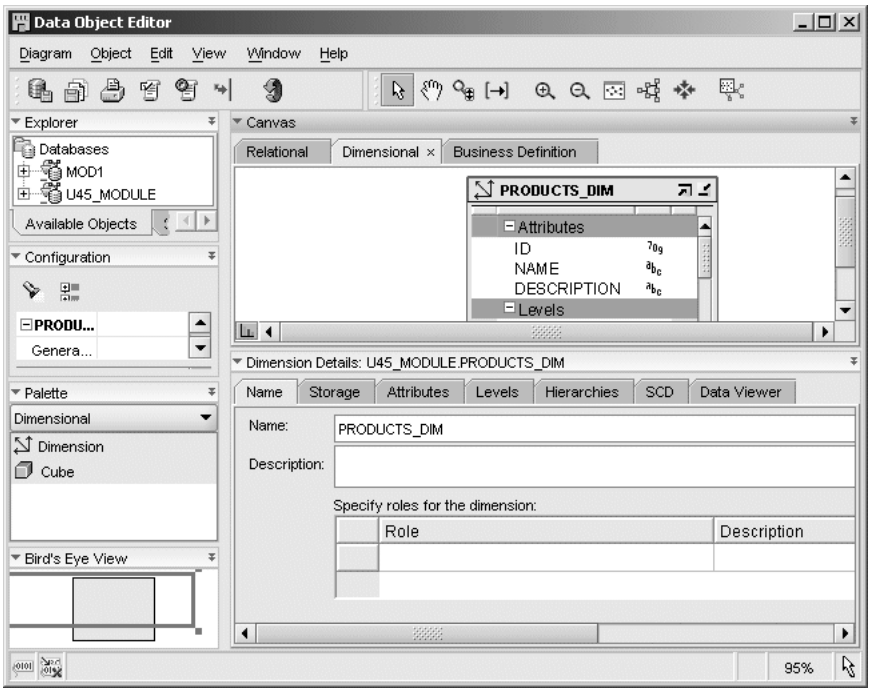
The Data Object Editor provides a centralized interface to create, edit, configure, validate, and deploy Oracle data objects. You can use the Data Object Editor with relational, dimensional, and business intelligence objects. You can also view the data stored in these objects.

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. 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. For more information about the Data Object Editor components, refer to the online help.

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.

Figure 13–1 displays the Data Object Editor.

Figure 13–1 Data Object Editor Window



This screenshot shows the Data Object Editor window. At the top of the window is the Title Bar. Below the Title Bar is the Menu Bar containing menu items. Below the Menu Bar is the Toolbar that contains icons. Below this are the remaining Data Object Editor components.

The left half of the window is divided into four parts with the following panes, ordered from top to bottom: Explorer, Configuration, Palette, and Bird's Eye View. The right half of the window is divided into two sections. The upper half consist of the Canvas which has three tabs: Relational, Dimensional, and Business Definition. The Dimensional tab is currently displayed. The lower half consists of the Dimension Details which contains the following tabs: Name, Storage, Attributes, Levels, Hierarchies, SCD, and Data Viewer. The Name tab is currently displayed.

At the bottom of the window is the Indicator Bar.

****

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.

Starting the Data Object Editor

Use one of the following methods to start the Data Object Editor:

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

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 for the data object, navigate to the Data Viewer tab of the Details panel. Click **Execute Query**.

The Data Viewer tab contains the following buttons: Execute Query, Get More, Where Clause, and More. The More button is displayed at the bottom of the tab.

Click **Execute Query** to execute a query on the data object and fetch its data.

By default, the Data Viewer displays the first hundred rows of data. To retrieve the next set of rows, click **Get More**. Alternatively, you can click **More** to perform the same action.

Click **Where Clause** to specify a condition that is used to restrict the data displayed by the Data Viewer. Clicking this button displays the Where Clause dialog box. Use this dialog box to specify the condition used to filter data. You can use this option for tables and views only.

The columns and column names displayed in the Data Object Editor are taken directly from the location in which the actual table is deployed. If the table definition in the Data Viewer does not match with what you see in the Data Object Editor, it is because the changes you made in the editor have not yet been deployed.

Using the Data Object Editor to Create Data Objects

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.

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

- Menu Bar

  See Creating Data Objects Using the Menu Bar on page 13-8.

- Canvas

  See Creating a Data Object Using the Canvas on page 13-9.

- Data Object Editor Palette

  See Creating a Data Object Using the Data Object Editor Palette on page 13-10.

Creating Data Objects Using the Menu Bar

To create a data object using the menu bar:

1. If it is not already open, open the Data Object Editor.

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. To create dimensions and cube, select the Dimensional 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 box. For more information about this dialog box, click **Help**.

   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. If it is not already open, open the Data Object Editor.

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 containing the types of data objects you can create.

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 box. For more information about this dialog box, click **Help**.

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. If it is not already open, open the Data Object Editor.
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 the Add a New or Existing <Object> dialog box. For more information about this dialog box, click **Help**.
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.

Using the Data Object Editor to Edit Oracle Data Objects

You edit a relational, dimensional, or business intelligence objects (item folders and business areas only) 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 SALES 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, see:

- Creating Dimensions on page 14-32
- Creating Cubes on page 14-49
- Editing a Business Area on page 11-13
- Editing an Item Folder on page 11-6

- Creating Table Definitions on page 13-25
- Creating View Definitions on page 13-29
- Creating Materialized View Definitions on page 13-31

Using Constraints

You can optionally create constraints on relational data objects such as tables, views, and materialized views.

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.

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 constraint.

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 13-12
   - Creating Foreign Key Constraints on page 13-12
   - Creating Unique Key Constraints on page 13-13
   - Creating Check Constraints on page 13-13

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 list in the Local Columns column.

5. In the **Local Columns** 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 box is displayed. Use this dialog box to select the table and the column that the current column references.

5. In the Key Selector dialog box, 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 box.

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 list in the Local Columns column.

5. In the **Local Columns** 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 can 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 Indexes

Use indexes to enhance query performance of your data warehouse. In Warehouse Builder, you can create indexes for tables and materialized views. 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 *Oracle Database Data Warehousing Guide*.

Creating Indexes

In general, you define indexes by using the Indexes tab in the Data Object Editor. To start 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 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".

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.

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.

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 start 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 panel.

You can create the following types of partitions:

- **Range Partitioning**: 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 22-24.

- **Hash Partitioning**: 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**: 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. The partitions are created and named automatically. You can then configure the partitions to share the same tablespace list.
- **List Partitioning**: 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**: 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**: 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.

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.

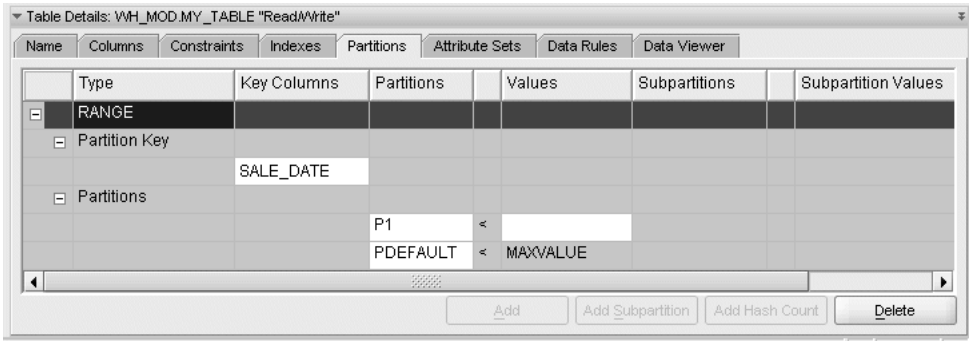
To use range partitioning, go to the Partitions tab in the Data Object Editor to specify a partition key and assign a name and value range for each partition you want to create. Figure 13–3 shows an example of a table partitioned into four range partitions based on the instructions below:

To partition data by ranges:

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.

Figure 13–2 Partition Tab with Range Partition Selected



This image displays the Partitions tab that defines a range partition. The Partition key is the SALE_DATE column.

****

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.

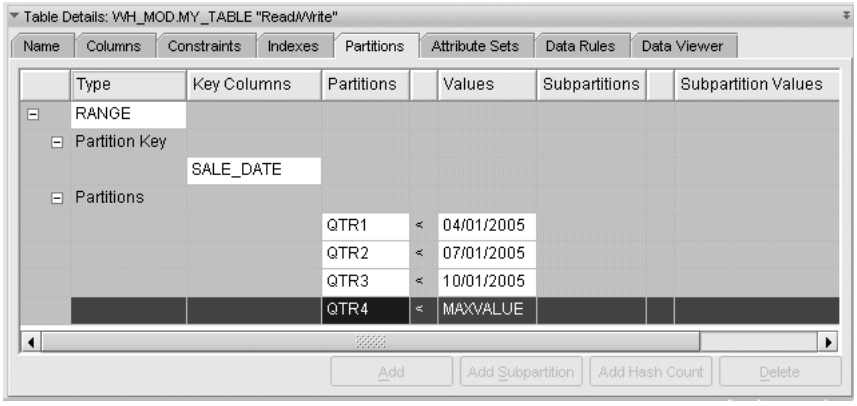
   To add more partitions between the first and last partitions, click the Partitions node and select **Add.**

   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 13–3 shows how to define a partition for each quarter of a calendar year.

Figure 13–3 Example Table with Range Partitioning



This image displays the Partitions tab that contains an example of defining a range partition.

****

You can 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.

Hash Partitioning

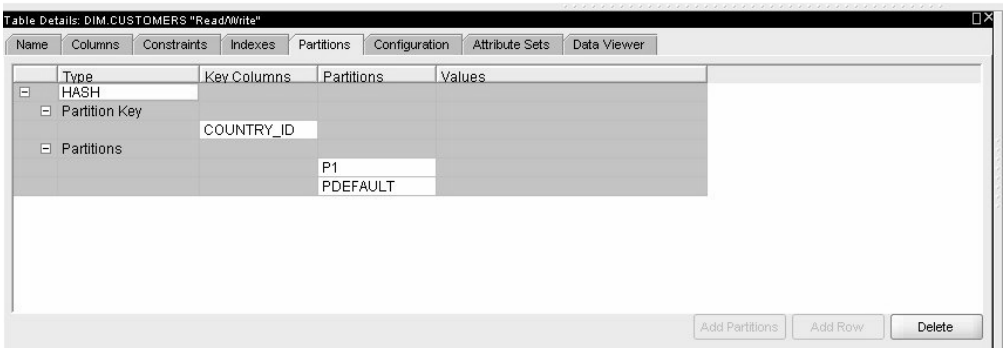
Hash partitioning assigns data to partitions based on a hashing algorithm that Oracle Database applies to a partitioning key you identify. The hashing algorithm evenly distributes rows among partitions, giving partitions approximately the same size. Hash partitioning is a good and easy-to-use alternative 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:

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.

Figure 13–4 Hash Partitioning



This image displays the Partitions tab that contains an example of defining a hash partition.

****

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

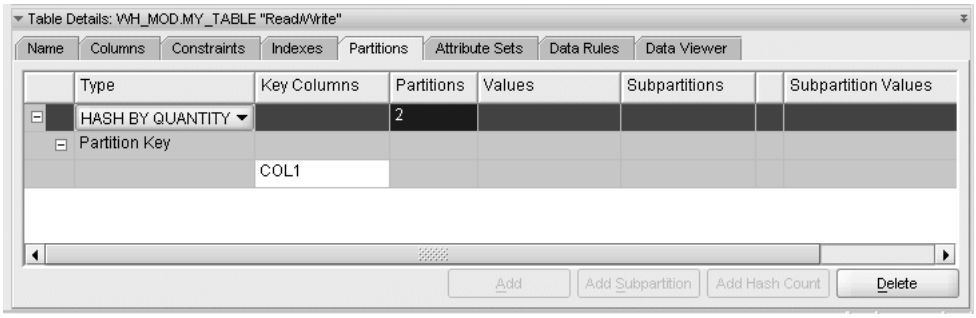
Use hash by quantity to quickly define hash partitioning. You define a partition key and the number of partitions and the partitions are automatically created and named. You can then configure the partitions to share the same tablespace list.

To partition data based on the hash algorithm:

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.

Figure 13–5 Hash by Quantity Partitioning



This image displays the Partitions tab that defines a hash by quantity partition.

****

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

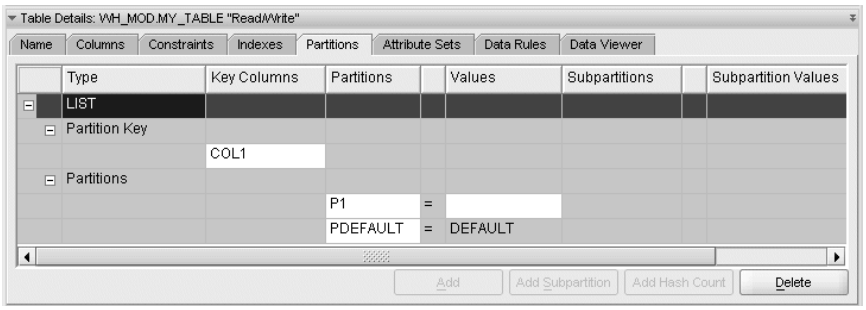
List partitioning enables you to explicitly assign rows to partitions. You can achieve this by specifying a list of discrete values for each partition. The advantage of list partitioning is that you can group and organize unordered and unrelated sets of data in a natural way.

To partition data based on a list of values:

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.

Figure 13–6 Partition Tab with List Partition Selected



This image displays the Partitions tab that defines a list partition.

****

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.

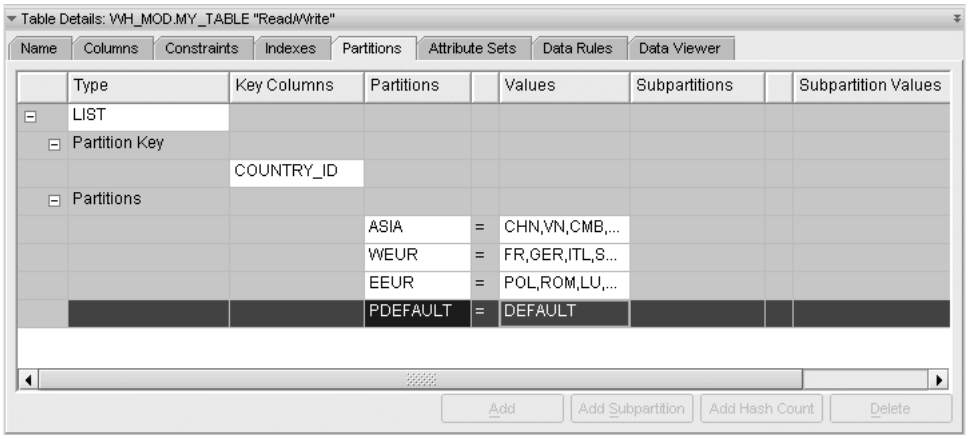
   To add more partitions between the first and last partitions, click the Partitions node and select **Add.**

   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 13–7.

Example of List Partitioning

Figure 13–7 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 13–7 List Partitioning Based on a Single Key Column

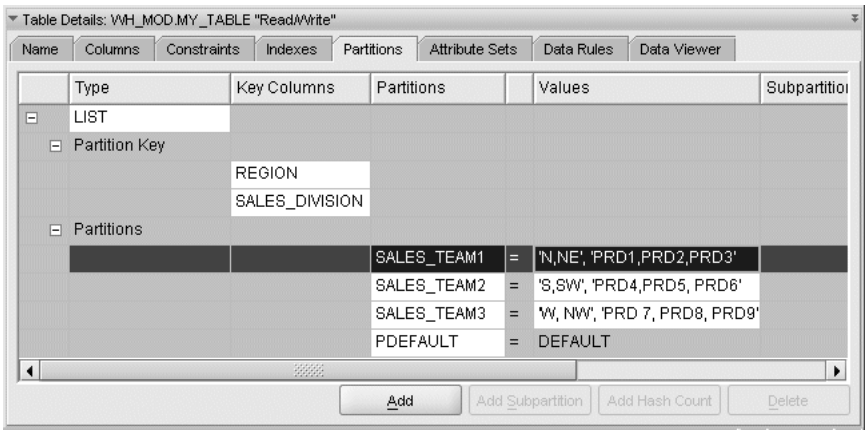


This image displays the Partitions tab that defines a list partition based on a single key column.

****

Figure 13–8 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 correspond to SALES_DIVISION.

Figure 13–8 List Partitioning Based on Multiple Key Columns



This image displays the Partitions tab that defines a list partition 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.

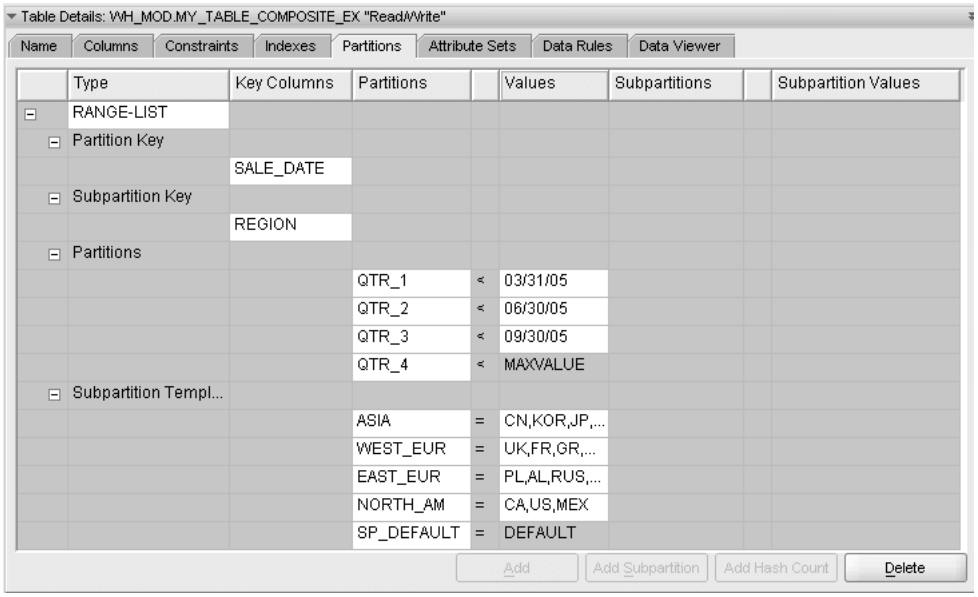
To range partition data and then subpartition based on list, hash, or hash by quantity:

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 13-16.

Figure 13–9 Range-List Partitioning with List Defined under Subpartition Template



This image displays the Partitions tab that defines a range-list partition.

****

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 13-22.
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 13-23.

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.

Creating Custom Subpartitions

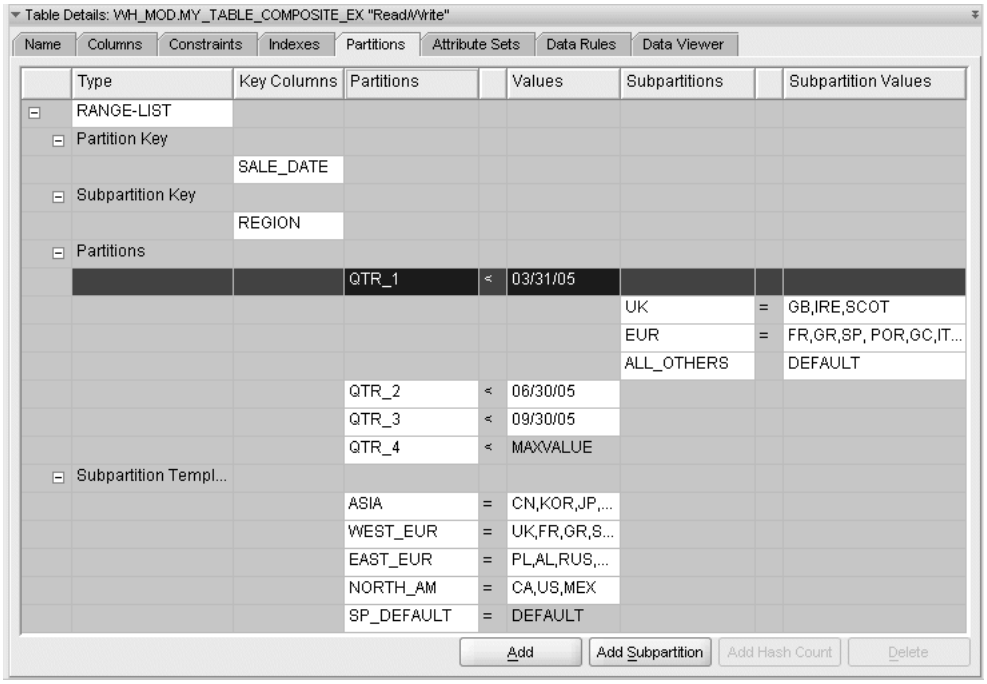
Using the subpartition template is the most convenient and likely the most common way to define subpartitions. Entries you specify under the subpartition template apply uniformly to all the partitions under the partition node. However, in some cases, you may want to override the subpartition template.

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 13–10 Subpartitions Overriding Subpartition Template



This image displays the Partitions tab that defines a range-list partition with subpartitions overriding the Subpartition template.

****

Index Partitioning

For all types of indexes except bitmap indexes, you can determine whether the index inherits the partitioning method of the underlying table. An index that inherits the partitioning method of the underlying table 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.

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 releases before Oracle 10*g*, 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 with 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 you want to specify an index partition key other than any of the table partition keys. In such cases, 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 panel and expand the Partition Parameters node.

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, the default tablespaces associated with the table are used 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, the default tablespaces associated with the table are used and the performance advantage for defining partitions is not realized when the limits for the partition tablespaces are exceeded.

Using Tables

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:

- Creating Table Definitions on page 13-25
- Editing Table Definitions on page 13-27

Creating Table Definitions

The table you create 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 workspace. You can later use these definitions to generate `.ddl` scripts that can be deployed to create physical tables in your target database. These tables can then be loaded with data from chosen source tables.

Use the Data Object Editor to create a table. Follow the steps listed below:

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 13-26
   - Columns Tab on page 13-26
   - Constraints Tab on page 13-26
   - Indexes Tab on page 13-26
   - Partitions Tab on page 13-27
   - Attribute Sets Tab on page 13-27
   - Data Rules Tab on page 13-27

   The Data Viewer tab enables you to view the data stored in the workspace table. For more information about the Data Viewer tab, see "Data Viewer" on page 13-8.

   After you finish defining the table using these tabs, the table definitions are created and stored in the workspace. The new table name in also added to 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 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 13-28.

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** 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 13-3.
- **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 about creating constraints, see "Creating Constraints" on page 13-11.

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 about creating these indexes, see "Creating Indexes" on page 13-14.

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 about these partitions and how to create each type of partition, see "Using Partitions" on page 13-15.

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 "Using Attribute Sets" on page 13-34.

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 23-13.

Before you apply a data rule to a table, the data rule should have been defined in the workspace. For information about creating data rules, see "Creating Data Rules" on page 23-43.

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 to Objects" on page 23-44.

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

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 13-26.

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 13-26.

Removing a column: Navigate to the Columns tab. Right-click the gray 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 13-11 and "Editing Constraints" on page 13-14 respectively.

To delete a constraint, select the row that represents the constraint by clicking the gray 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 13-34. See "Editing Attribute Sets" on page 13-35 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 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 workspace you must deploy the changed table definition.

Using Views

You can define views and materialized views. This section describes views. For information about materialized views, see "Using Materialized Views" on page 13-31.

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.

A view can be defined to model a query on your target data. This query information is stored in the workspace. You can later use these definitions to generate `.ddl` scripts that can be deployed to create views in your target system.

For information about using views, refer to:

- Creating View Definitions on page 13-29
- Editing View Definitions on page 13-30

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 workspace. 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 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 about the following Data Object Editor tabs:
   - Name Tab on page 13-29
   - Columns Tab on page 13-30
   - Query Tab on page 13-30
   - Constraints Tab on page 13-30
   - Attribute Sets Tab on page 13-30
   - Data Rules Tab on page 13-30
   - Data Viewer Tab on page 13-30
5. Close the Data Object Editor.

   Warehouse Builder creates a definition for the view, stores this definition in the workspace, and inserts its name in the Project Explorer.

Name Tab

Follow the rules in Naming Conventions for Data Objects 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 13-26.

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 these constraints are not used 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 13-11.

Attribute Sets Tab

Use the Attribute Sets tab to define attribute sets for the view. For more information about attribute sets and how to create them, see "Using Attribute Sets" on page 13-34.

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 13-27.

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 13-8.

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 about these details, see "Columns Tab" on page 13-30.

Removing a column: Navigate to the Columns tab. Right-click the gray 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 13-11 and "Editing Constraints" on page 13-14 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 13-34. See "Editing Attribute Sets" on page 13-35 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

You can define views and materialized views. This section discusses materialized views. For information about conventional views, see "Using Views" on page 13-28.

The following sections provide information about using materialized views:

About Materialized Views

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, you can configure it to take advantage of the query rewrite and fast refresh features available in Oracle Database. For information about query rewrite and fast refresh, "Fast Refresh for Materialized Views" on page 13-48.

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.

The Data Object Editor enables you to create materialized views. To create a materialized view:

1. From the 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 about the following tabs of the Data Object Editor to create the materialized view definition:
   - Name Tab on page 13-32
   - Columns Tab on page 13-32
   - Query Tab on page 13-32
   - Constraints Tab on page 13-33
   - Indexes Tab on page 13-33
   - Partitions Tab on page 13-33
   - Attribute Sets Tab on page 13-33
   - Data Rules Tab on page 13-33
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 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 13-26.

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 are not used when enumerating DDL for the view. For information about creating constraints, see "Creating Constraints" on page 13-11.

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 about creating indexes, see "Creating Indexes" on page 13-14.

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 about how to define attribute sets, see "Creating Attribute Sets" on page 13-34.

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 about these details, see "Columns Tab" on page 13-30.

Removing a column: Navigate to the Columns tab. Right-click the gray 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 13-11 and "Editing Constraints" on page 13-14 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 13-34. See "Editing Attribute Sets" on page 13-35 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, a predefined attribute set is generated containing all the columns in that table. In addition, predefined attribute sets are generated for each defined constraint. Predefined attribute sets cannot be modified or deleted.

Creating Attribute Sets

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** 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, select **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.

   Select **Select All** to select all the displayed columns in the attribute set. Select **Deselect All** to exclude all the columns from the attribute set. To remove a column from the attribute set, deselect **Include**.
7. If you selected BRIDGE-TYPE, click **Advanced.**

   Warehouse Builder displays the Advanced Attribute Set Properties dialog box.
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: Select **Hidden** 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 box.

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** 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 gray 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 gray 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 gray cell to the left of an attribute and use the buttons to move the attribute up or down in the order.

- Delete the attribute set

  Right-click the gray 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, you are creating sequence definitions that are saved in the workspace. 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 on page 13-36
- Creating a Sequence Definition on page 13-36
- Editing Sequence Definitions on page 13-37

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, these attributes are created.

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 Project Explorer, expand the warehouse module node.
2. Right-click Sequences and select **New** from the 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 13-6, 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

Use the Edit Sequence dialog box 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 box is displayed. This dialog box 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 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 box 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 data types use Oracle built-in data types and other user-defined data types as the building blocks of object types that model the structure and behavior of data in applications. The built-in data types are mostly scalars and do not provide the same flexibility that modelling an application specific data structure does.

Consider 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 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 13-3 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 13-38 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;
```

A Varray is an ordered set of elements. Therefore the first address in the list is considered as the primary address, and the remaining addresses are considered as 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 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 13-40 to edit the definition of the varray.

About Nested Tables

A 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. An example of this would be 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 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 13-41 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 Design Objects

In this phase, you assign physical deployment properties to the object definitions 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 box. The following sections show you how to assign physical properties to your logical design model.

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 Project Explorer, expand **Databases,** expand **Oracle,** and right-click a target module name and select **Configure.**

   Warehouse Builder displays the Configuration Properties dialog box.

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 box.

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, the tablespace value configured at the target module level is used during code generation. If you configure a tablespace for each index at the object level the tablespace value configured at the target module level is overwritten.

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, the value configured at the target module level is used during code generation. If you configure a tablespace for each individual object, the tablespace value configured at the target module level is overwritten.

Generation Preferences

End of Line: Defines the end of line markers for flat files. This depends 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 Oracle 8*i*, row-based code is generated.

Each release 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 Oracle 8*i* as the PL/SQL Generation Mode, you cannot access some Oracle 9*i* Warehouse Builder components such as Table Functions and External Tables. For a list of components not compatible with prior releases of the Oracle Database, see *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 6, "Integrating with 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.** The Configuration Properties dialog box displays. Set the configuration parameters listed in the following sections.

Identification

- **Deployable:** Select this option to indicate that you want to deploy this table. Scripts are generated only for table constraints marked deployable.
- **Error Table Only:** Select this option to perform generation or deployment actions only on the error table associated with a table. Use this option when you want to add an error table to an existing database table. This setting only controls the actions of the Deployable parameter, but does not override it.

  Unselect this option to deploy the error table along with the primary table.

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, the table is generated based on the tablespace value set in the target module configuration properties. If you configure the tablespace for individual objects, the tablespace value configured for the target module is overwritten.

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-separated 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.

Error Table

- **Error Table Name:** Indicates the name of the error 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 error 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 13-45.
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 13-48.

  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. Select 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 13-48.

  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:** Select PRIMARY_KEY to create a primary key materialized view. Select ROWID to create a ROWID materialized view. The default setting is PRIMARY_KEY.
- **FOR UPDATE:** Select 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-separated 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`.

Error Table

- **Error Table Name:** Indicates the name of the error 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 error 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.
- **Error Table Only:** Select this option to perform generation or deployment actions only on the error table associated with the materialized view. Use this option when you want to add an error table to an existing materialized view. This setting only controls the actions of the Deployable parameter, but does not override it.

  Unselect this option to deploy the error table along with the materialized view.

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 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, 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 about additional restrictions, see *Oracle Database 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 about views, refer to:

- "About Views" on page 13-28
- "Creating View Definitions" on page 13-29

Configuring Sequences

A script is generated 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 box 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.

14

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:

- About Dimensional Objects
- About Dimensions
- About Slowly Changing Dimensions
- About Time Dimensions
- About Cubes
- Creating Dimensions
- Creating Slowly Changing Dimensions Using the Data Object Editor
- Configuring Dimensions
- Creating Cubes
- Configuring Cubes
- Creating Time Dimensions

About Dimensional Objects

This section describes the basic concepts related to dimensional objects. If you are familiar with dimensional objects concepts and the types of implementations for dimensional objects in Warehouse Builder, skip the next few sections and continue with "Creating Dimensions" on page 14-32.

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. However, you can edit a dimension that you created using the Create Dimension Wizard and modify it to use a snowflake schema implementation.

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

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

You set the Deployment Option configuration property to specify the type of implementation for a dimensional object. For more information on setting this property, see "Configuring Dimensions" on page 14-48 and "Configuring Cubes" on page 14-64.

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.

For a relational implementation, you cannot use the Data Viewer to view the data stored in the dimensional object. You can however view the data stored in the implementation tables of the dimensional object using the Data Viewer.

Binding Binding is the process of connecting the attributes 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.

For dimensions, you connect the level attributes and level relationships to the columns in the implementation objects. For cubes, you connect the measures and dimension references to implementation table columns.

Warehouse Builder provides two methods of binding:

- Auto binding
- Manual binding

Auto Binding In auto binding, Warehouse Builder creates the implementation tables, if they do not already exist. The attributes and relationships of the dimensional object are then bound to the columns that store their data. You can perform auto binding using both the wizards and the editors.

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 14-14.

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

To perform auto binding:

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

   The Data Object Editor for this dimensional object is displayed.

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

   Alternatively, select the dimensional object node on the canvas and from the **Object** menu select **Bind**.

   If the Bind option is not enabled, verify if the dimensional object uses a relational or ROLAP implementation. In the case of dimensions, ensure that the Manual option is not set in the Implementation section of the Storage tab.

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.

To perform manual binding for a dimensional object:

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

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

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

   The Data Object Editor for the dimensional object is displayed. On the canvas, the Dimensional tab is active.

3. Right-click the dimensional object and select **Detail View**.

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

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 box. 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 box 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. For dimensions, if more than one data object is used to store the dimension data, perform steps 4 to 6 for each data implementation object.
8. For dimensions, map the attributes in each level of the dimension to the columns that store their data. Also map the level relationships to the database column that store their data.

   For cubes, map the measures and dimension references to the columns that store the cube data.

   To map to the implementation object columns, hold down your mouse on the dimension or cube attribute, drag, and then drop on the column that stores the attribute value.

   For example, for the `PRODUCTS` dimension described in "Dimension Example" on page 14-12, 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.

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.

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.

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, see *Oracle OLAP User's Guide*.

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.

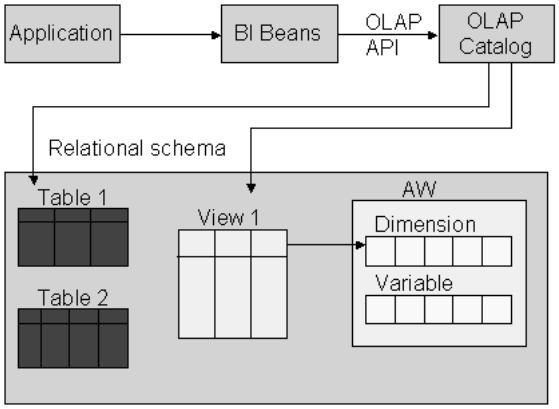
When you deploy a dimensional object using Warehouse Builder, you can specify if the dimensional object metadata should be stored in the OLAP catalog.

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 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 14–1 describes how the OLAP catalog enables applications to access data stored in relational tables and analytic workspaces.

Figure 14–1 Using the OLAP Catalog to Access Dimensional Objects



This graphic illustrates the use of the OLAP catalog to access relational data. In this graphic, the data originates in an application, represented in the top left corner by a box labeled Application. To the right of this box are two boxes, from left to right, labeled BI Beans and OLAP Catalog. The data flows from the Application to BI Beans and then to the OLAP Catalog. The data flow is represented by arrows connecting these boxes.

In the lower part of the screen is the Relational Schema. It contains the following: Table 1, Table 2, View 1, and an AW that contains the following: Dimension and Variable. An arrow connects the OLAP Catalog to Table 1. An arrow also connects the OLAP Catalog to View 1. View 1 is connected to Dimension that is contained in the AW.

****

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. To specify the type of implementation for dimensional objects, you set the deployment option. The configuration parameter Deployment Options enables you to set the deployment option.

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 Discoverer for OLAP to access the dimensional object data after you deploy data only. Use this option if you previously deployed with "Data Objects Only" and now want to deploy the CWM Catalog definitions without re-deploying the data objects again.
- **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 14-3. For more information on performing binding, see "Binding" on page 14-3.

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 17-18. For information on loading cubes, see "Cube Operator" on page 17-10.

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 be either a surrogate identifier, a business identifier, a parent identifier, or a regular attribute.
- A regular attribute can also play only one of the following roles at a time: effective date, expiration date, or triggering attribute.
- 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 Slowing Changing Dimension (SCD) must have the effective date, expiration date, and at least one triggering attribute.
- A Type 3 Slowing Changing Dimension (SCD) must have the effective date and at least one triggering attribute.

Limitations of Deploying Dimensions to the OLAP Catalog

For dimensions with a ROLAP implementation, there are implications and limitations related to the various dimension structures when either reporting on the underlying

tables or deploying to the OLAP catalog. Although the dimension may be successfully deployed, errors could occur when other applications, such as Oracle Discoverer access the OLAP catalog.

The following are items that are affected by this limitation:

- No reporting tool has metadata about all aspects of dimensional metadata we capture, so this must be incorporated into the query/reports. Otherwise you will see odd information because of the way the data is populated in the implementation tables.

  The dimension and cube implementation tables store solved rows which contain negative key values. You can filter out these rows in your queries or reports. When you create a query or report, use the view that is associated with a dimension instead of the dimension itself. Each dimension has a view that is associated with it. The view name is specified in the configuration property View Name of the dimension or cube.

- Skip-level hierarchies and ragged hierarchy metadata is not deployed to the OLAP catalog.

  If you create a dimension that contains skip-level or ragged hierarchies, the metadata for these is stored in the Warehouse Builder repository but is not deployed to the OLAP catalog.

- Dimensions with multiple hierarchies must have all dimension attributes mapped along all the hierarchies.

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 14-13.

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 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 fulfillment time. The sales cube can refer to the order time, ship time, and fulfillment time dimensions.

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 14–1 lists the levels in the PRODUCTS dimension and the surrogate identifier and business identifier for each of the levels in the dimension.

Table 14–1 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

Control Rows

Warehouse Builder creates control rows that enable you to link fact data to a dimension at any level. For example, you may want to reuse a Time dimension in two different cubes to record the budget data at the month level and the actual data at the day level. Because of the way dimensions are loaded with control rows, you can perform this without any additional definitions. Each member in a dimension hierarchy is represented using a single record.

All control rows have negative dimension key values starting from -2. For each level value of higher levels, a row is generated that can act as a unique linking row to the fact table. All the lower levels in this linking or control rows are nulled out.

Consider the Products dimension described in "Dimension Example" on page 14-12. You load data into this dimension from a table that contains four categories of products. Warehouse Builder inserts control rows in the dimension as shown in

Table 14–2. These rows enable you to link to a cube at any dimension level. Note that the table does not contain all the dimension attribute values.

Table 14–2 Control Rows Created for the Products Dimension

Dimension Key	Total Name	Categories Name	Product Name
-3	TOTAL		
-9	TOTAL	Hardware	
-10	TOTAL	Software	
-11	TOTAL	Electronics	
-12	TOTAL	Peripherals	

To obtain the real number of rows in a dimension, count the number of rows by including a `WHERE` clause that excludes the NULL rows. For example, to obtain a count on Products, count the number of rows including a `WHERE` clause to exclude NULL rows in 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 relationships 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 four levels between employee A and the CEO, whereas, there may be six 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 14-41.

> **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 14-3.

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 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:

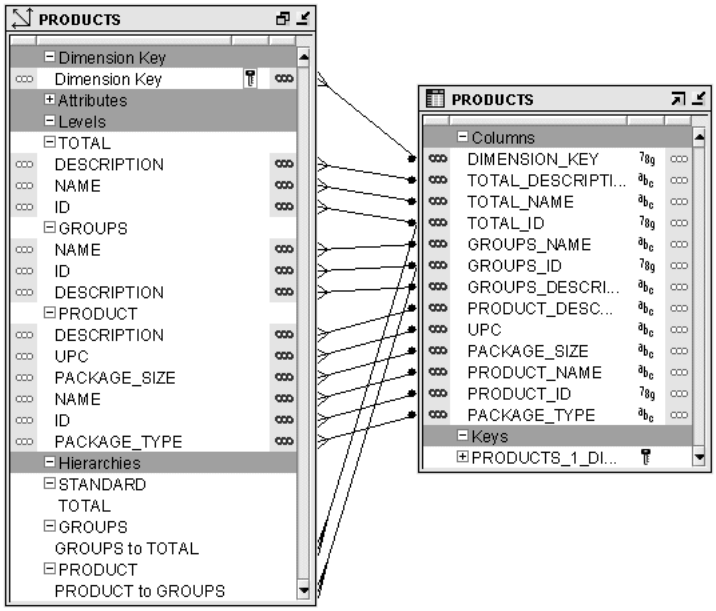
- If the level attribute name is not unique, Warehouse Builder prefixes it with the name of the level.
- If the level attribute name is unique, Warehouse Builder does not use any prefix.

> **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 14–2 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 14–2 Star Schema Implementation of Products Dimension



The screenshot is a diagrammatic representation showing the table columns to which each attribute in the Products dimension is mapped. On the left is the PRODUCTS dimension. On the right is the PRODUCTS implementation table. There are arrows from attributes in the PRODUCTS dimension to the columns in the PRODUCTS implementation table.

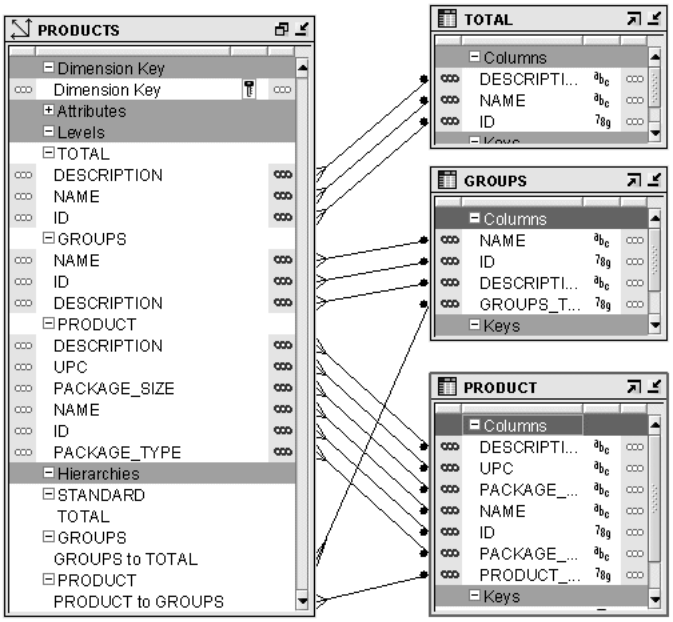
****

For relational or ROLAP dimensions that use a star implementation, you can bind attributes from more than one levels to the same database column. A database column that is bound to attributes from more than one dimension levels is referred to as a *shared column*. For a Type 2 SCD, you cannot set the level attributes that are bound to a shared column as triggering attributes.

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 14–3 displays the snowflake implementation of the `PRODUCTS` dimension. Each level in the dimension is mapped to a different table.

Figure 14–3 Snowflake Schema Implementation of the Products Dimension



This screenshot is a diagrammatic representation of the attribute bindings of the dimension when it is implemented using a snowflake schema. To the left is the PRODUCTS dimension. To the right are three tables ordered from top to bottom: TOTAL, GROUPS, and PRODUCT. There are arrows from attributes in the PRODUCTS dimension to columns in the tables TOTAL, GROUPS, and PRODUCTS.

****

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 14-3.

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 14-14.

  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 14-14.

  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.

For information about how to perform auto binding, see "Auto Binding" on page 14-4. Auto binding uses the implementation settings described in "Relational and ROLAP Implementation of a Dimension" on page 14-14.

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.

For information about how to perform manual binding, see "Manual Binding" on page 14-4.

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 14-5.

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 type 3 SCDs require the Oracle Warehouse Builder Enterprise ETL Option as described in *Oracle Database Licensing Information*.

Table 14–3 describes the three types of SCDs.

Table 14–3 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 attributes for which historical values must 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 Slowly Changing Dimensions

In a Type 1 Slowly Changing Dimension (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 need not 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
- Implementing a Dimension

About Type 2 Slowly Changing Dimensions

A Type 2 Slowly Changing Dimension (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 Slowly Changing Dimension

To define a Type 2 Slowly Changing Dimension (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.

Type 2 SCD Example

Consider the `Customers` Type 2 SCD that contains two levels, Household and Customer. Table 14–4 lists level attributes of the `Customers` Type 2 SCD.

Table 14–4 Customers Type 2 SCD Attributes

Attribute Name	Identifier
`ID`	Surrogate identifier
`BUSN_ID`	Business identifier
`ADDRESS`	
`ZIP`	
`MARITAL_STATUS`	
`HOME_PHONE`	
`EFFECTIVE_DATE`	Effective Date
`EXPIRATION_DATE`	Expiration Date

Customer is the leaf level and Household is the non-leaf level.

The Household level implements the following attributes: `ID`, `BUSN_ID`, `ADDRESS`, `ZIP`, `EFFECTIVE_DATE`, and `EXPIRATION_DATE`. The Customer level implements the following attributes: `ID`, `BUSN_ID`, `MARITAL_STATUS`, `HOME_PHONE`, `EFFECTIVE_DATE`, and `EXPIRATION_DATE`.

The table that implements this Type 2 SCD (for a relational or ROLAP implementation) contains the following columns: `DIMENSION_KEY`, `H_ID`, `H_BUSN_ID`, `H_ADDRESS`, `H_ZIP`, `H_EFFECTIVE_DATE`, `H_EXPIRATION_DATE`, `C_ID`, `C_BUSN_ID`, `C_MARITAL_STATUS`, `C_HOME_PHONE`, `C_EFFECTIVE_DATE`, and `C_EXPIRATION_DATE`.

To create the `CUSTOMERS` Type 2 SCD:

- Specify that the `ZIP` attribute of the Household level and the `MARITAL_STATUS` attribute of the Customer level are the triggering attributes.
- Use two additional attributes to store the effective date and the expiration date of the level records. When you use the Create Dimension wizard, Warehouse Builder

creates these additional attributes for the lowest level only. If you use the Data Object Editor, you must explicitly create these attributes and apply them to the required levels.

Note: While loading some Type 2 SCDs, if the target is an Oracle 9*i* database, only row-based mode is supported.

A workaround is to switch on hierarchy versioning, by setting the parent surrogate ID reference attribute as a trigger for all levels.

Hierarchy Versioning

When the non-leaf level of a dimension contains versioned attributes, the versioning of this non-leaf level results in the versioning of its corresponding child records, if they have effective date and expiration date attributes. For example, when the value of the `H_ZIP` is updated in a particular Household level record, the child records corresponding to this Household level are automatically versioned.

Hierarchy versioning is not enabled by default for Type 2 SCDs. When you create a Type 2 SCD using the Create Dimension Wizard, hierarchy versioning is disabled. You must use the Data Object Editor to enable hierarchy versioning.

To enable hierarchy versioning:

1. Right-click the Type 2 SCD in the Project Explorer and select **Open Editor**.

   The Data Object Editor is displayed.

2. Navigate to the SCD tab.

3. Click **Settings** to the right of the Type 2: Store the Complete change history option.

   The Type 2 slowly changing dimension dialog box is displayed. The attributes of each level are displayed under the level node.

4. In the child level that should be versioned when its parent attribute changes, for the attribute that represents the parent attribute of this child level, select **Trigger History** in the Record History column.

   For example, you create the `Customers` Type 2 SCD using the Create Dimension Wizard. Open the Data Object Editor for this Type 2 SCD and navigate to the Type 2 slowly changing dimension dialog box as described in steps 1 to 3. The `Customer` level has an attribute called `HOUSEHOLD_ID`. This attribute represents the parent attribute of each `Customer` record. For the `HOUSEHOLD_ID` attribute, select **Trigger History** in the Record History column.

Updating Type 2 Slowly Changing Dimensions

All the levels in a dimension need not store historical data. Typically, only the lowest level, also called the leaf level, stores historical data. However, you can also store historical data for other dimension levels.

When a record in a Type 2 SCD is versioned, the old record is marked as closed and a new record is created with the updated values. The expiration date of the record is set to indicate that it is closed. The new record is referred to as the current record and, by default, has a default expiration of NULL. While loading data into the Type 2 SCD, you can set the expiration date by using the configuration parameters for the Dimension operator. For more information, see "Dimension Operator" on page 17-14.

You can update the following in a Type 2 SCD:

- Leaf level attribute

- Leaf level versioned attribute
- Non-leaf level attribute
- Non-leaf level versioned attribute
- Leaf level parent attribute

The following sections describe the Warehouse Builder functionality for these update operations.

Updating a Leaf Level Attribute

When you update a leaf level attribute, the value of this attribute is updated in the corresponding record.

For example, if you update the value of `C_HOME_PHONE` in a `Customer` level record, the record is updated with the changed phone number.

Updating a Leaf Level Versioned Attribute

When you update a leaf level versioned attribute, the current record is marked as closed. A new record is created with the updated value of the versioned attribute.

For example, if you update the marital status of a customer, the current record is marked as closed. A new record with the updated marital status is created for that customer.

Updating a non-leaf Level Attribute

When you update an attribute in a non-leaf level, the open records of the non-leaf level and the child records corresponding to this non-leaf level are updated with the new value.

For example, when you update the `H_ADDRESS` attribute in a `Household` level record, the current open record for that household is updated. All open child records corresponding to that particular household are also updated.

Updating a non-leaf Level Versioned Attribute

The update functionality depends on whether hierarchy versioning is enabled or disabled.

Hierarchy Versioning Disabled

The non-leaf level record corresponding to the versioned attribute is closed and a new record is created with the updated value. The child records of this non-leaf level record are updated with the changed value of the non-leaf level versioned attribute.

For example, when the value of `H_ZIP` in a `Household` level record is updated, the current open record for that household is closed. A new record with the updated value of `H_ZIP` is created. The value of `H_ZIP` is updated in all the child records corresponding to the updated household record.

Hierarchy Versioning Enabled

The non-leaf level record corresponding to the versioned attribute is closed and a new record is created with the updated value. Child records corresponding to this non-leaf level record are also closed and new child records are created with the updated value.

For example, when the value of `H_ZIP` in a `Household` level record is updated, the current open record for that household and its corresponding child records are closed. New records are created, with the updated value, for the household and for the child records corresponding to this household.

Updating the Leaf Level Parent Attribute

In addition to updating the level attributes in a Type 2 SCD, you can also update the parent attribute of a child record. In the Customers Type 2 SCD, the attribute H_BUSN_ID in a Customer record stores the parent attribute of that customer. The update functionality for the leaf level parent attribute depends on whether hierarchy versioning is enabled or disabled.

Hierarchy Versioning Disabled

The child record is updated with the new parent attribute value.

For example, when you update the value of the H_BUSN_ID attribute representing the parent record of a Customer record, the Customer record is updated with the new values.

Hierarchy Versioning Enabled

The child record is closed and a new record with the changed parent attribute value is created.

For example, when you update the H_BUSN_ID attribute of a customer record, the current customer record is closed. A new customer record with the updated H_BUSN_ID is created.

About Type 3 Slowly Changing Dimensions

A Type 3 Slowly Changing Dimension (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 Slowly Changing Dimension

To define a Type 3 Slowly Changing Dimension (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.

Type 3 SCD Example

The PRODUCTS dimension described in "Dimension Example" on page 14-12 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 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
- Defining a Time Dimension
- Implementing a Time Dimension
- Using a Time Dimension in a Cube Mapping
- Populating a Time Dimension

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 14-11. 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 14–5 describes the dimension attributes of the Warehouse Builder time dimension.

Table 14–5 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 14–6 lists the attributes of each level in the Warehouse Builder time dimension. For a description of each attribute, refer to Appendix B.

Table 14–6 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.
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 14-13.

Using a Time Dimension in a Cube Mapping

A time dimension created using the Create 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 18-11.

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

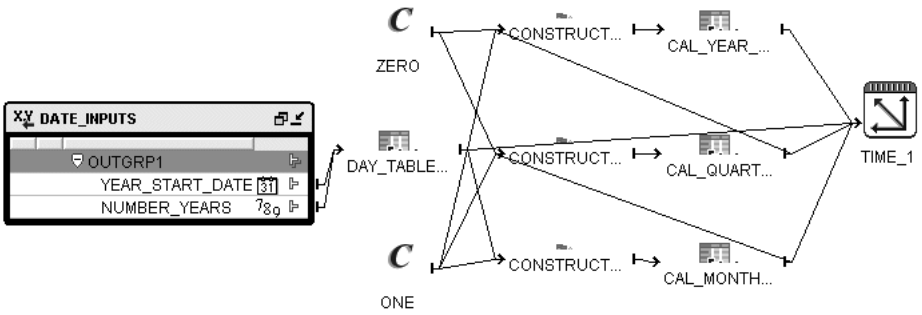
- Number of years of the data
- Start day and month of fiscal year (only for fiscal time dimensions)
- Start day of fiscal week (only for fiscal time dimensions)
- Fiscal type (only for fiscal time dimensions)

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 14-42.

Figure 14–4 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 14–4 Mapping that Populates a Time Dimension



This screenshot shows a mapping that populates a calendar time dimension. On the left is a Mapping Input operator `DATE_INPUTS`. To the right of this are two Constant operators and one Table operator, DAY_TABLE. There are arrows from DATE_INPUTS to DAY_TABLE.

To the right of these operators are three Construct Object operators. To the right of the Construct Object operators are the Table operators CAL_YEAR, CAL_QUARTER, and CAL_MONTH. To the extreme right is the time dimension TIME_1.

There are arrows from the Constant operators to the Construct Object operators and from the Construct Object operators to the Table operators CAL_YEAR, CAL_QUARTER, and CAL_MONTH. There are arrows from the Table operators DAY_TABLE, CAL_YEAR, CAL_QUARTER, and CAL_MONTH to the Dimension operator TIME_1.

****

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.

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 Promotions 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 14-11.

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 MOLAP 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 14-51 and "Dimensions Tab" on page 14-54. For more information on sparsity, see "Advanced Dialog Box" on page 14-54.

Default Aggregation Method

You can define aggregations that should be performed on the cube. For ROLAP cubes, you can only define a single aggregation method for the cube. For MOLAP cubes, you can define a different aggregation method for each dimension of each measure. 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 14–7 describes the dimensionality of the Sales cube. It lists the name of the dimension and the dimension level that the cube references.

Table 14–7 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.

The types of implementation you can use for cubes are:

- Relational implementation
- ROLAP implementation
- MOLAP implementation

To set the type of implementation for a cube, use the **Deployment Option** configuration property. For more details on setting this option, see "Configuring Cubes" on page 14-64.

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 14-3.

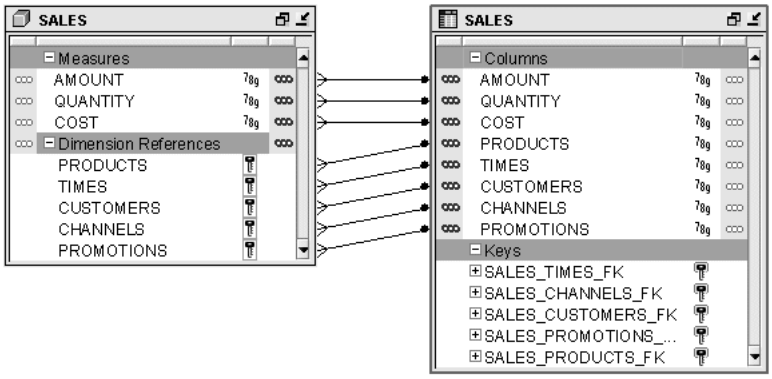
To implement a cube:

- 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 14–5 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 14–5 Implementation of the Sales Cube



The screenshot shows the relational implementation of the SALES cube. To the left is the SALES cube. To the right is a SALES table. There are arrows from the SALES cube to the SALES table connecting the cube attributes and dimensions to the table columns.

****

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 14-3.

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 14-4.

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.

For information about how to perform manual binding, see "Manual Binding" on page 14-4.

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 14-5.

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.

Creating Dimensions

To create a dimension, 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, many settings are defaulted 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 workspace using auto binding.

  For more information about the defaults used by the Dimension wizard, see "Defaults Used By the Create Dimension Wizard" on page 14-38.

- 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. 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 about the Time Dimension wizard, see "Creating Time Dimensions" on page 14-64.

Using the Create Dimension Wizard

To create a dimension using the Dimension wizard:

1. From the 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 14-33

Name and Description Page

Use the Name and Description page to describe your dimension. Enter the following information on this page:

- **Name:** This is the name used 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:

- 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, the implementation tables used to store the dimension data are created. 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, the dimension is stored 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 *Oracle Warehouse Builder Release Notes*.

Dimension Attributes Page

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, the following attributes are created 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 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, two attributes are created, 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, two attributes are created, 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, an attribute is created in both the Product and Groups levels, that reference the business identifier of the parent level.

Note: To create additional hierarchies, use the Hierarchies tab of the Data Object Editor as described in Hierarchies Tab on page 14-42.

Level Attributes Page

The Level Attributes page 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.

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, the following defaults are used:

- 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

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 about slowly changing dimensions concepts, see "About Slowly Changing Dimensions" on page 14-17.

> **Note:** Use of this feature requires the Oracle Warehouse Builder Enterprise ETL Option as described in *Oracle Database Licensing Information*.

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, an additional attribute is created 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 workspace:

- 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.

Warehouse Builder creates the definitions of these objects in the workspace 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 14-5.

> **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, default values are set for some of the attributes that are used to create the dimension. The following sections describe the defaults used.

Storage For a relational storage, the star schemas used as the default implementation method.

When you choose multidimensional storage, the dimension is stored 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, it is created. 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 14–8.

Table 14–8 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, an 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, additional attributes called PREV_DESCRIPTION, PREV_PACKAGE_TYPE, PREV_PACKAGE_SIZE, and PREV_UPC are created 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, an analytic workspace with the same name as the Oracle module that contains the dimension is created. 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, it is first created and then the dimension is stored in this analytic workspace.

For time dimensions, irrespective of the storage type, a map that loads the time dimension is created. 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 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 box.
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 about the following tabs of the Dimension Details panel:

   - Name Tab on page 14-40
   - Storage Tab on page 14-40
   - Attributes Tab on page 14-41
   - Levels Tab on page 14-42
   - Hierarchies Tab on page 14-42
   - SCD Tab on page 14-43
   - Data Viewer Tab on page 14-44

   When you use the Data Object Editor to create a dimension that has a relational implementation, the physical structures that store the dimension data are not automatically created. 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 14-44.

Name Tab

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. Use the **Description** field to enter an optional description for the dimension.

Dimension Roles Use the Dimension Roles section to define dimension roles. For more information about dimension roles, see "Dimension Roles" on page 14-11. 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

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, these storage settings are used 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, a unique key that includes the business identifiers of all three levels is created. 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, the level name is added 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, the data type of the surrogate identifier is changed to `VARCHAR2`.

Attributes Tab

Use the Attributes tab to define the dimension attributes. The Attributes tab contains two sections: Sequence and Dimension Attributes.

Sequence The Sequence attribute is required 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 box. This dialog box 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 14-34.

Levels Tab

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. 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

Use the Hierarchies tab to create dimension hierarchies. The Hierarchies tab contains two sections: Hierarchies and Levels.

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 select **MOLAP: Multidimensional data structures** on the Storage tab. For more information about value-based hierarchies, see "Value-based Hierarchies" on page 14-13.
- **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 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** 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

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 select **Relational: relational data structures** on the Storage tab. The options that you can select for slowly changing policy are:

- **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 14-45.

- **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 14-46.

> **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

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, the attributes in the dimension are mapped 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 about the auto binding rules, see "Auto Binding" on page 14-4.

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 box. 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 about deploying dimensions, see "Deploying Dimensions" on page 14-38.

Creating Slowly Changing Dimensions Using the Data Object Editor

You can create an SCD either using the Dimension wizard or the Data Object Editor.

Note: Creating SCD type 2 or 3 requires the Oracle Warehouse Builder Enterprise ETL Option as described in *Oracle Database Licensing Information*.

To create an SCD using the Dimension Wizard, 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 about the Slowly Changing Dimension page, see "Slowly Changing Dimension Page" on page 14-36.

To create SCDs using the Data Object Editor, see the following sections:

- Creating a Type 2 SCD on page 14-45
- Creating a Type 3 SCD on page 14-46

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 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 about the Name tab of the Data Object Editor as described in the "Name Tab" on page 14-40.
5. On the Attributes tab, for each level, create two additional attributes to store the effective date and the expiration date. For more information about creating attributes, see "Attributes Tab" on page 14-41.
6. Provide information about the following tabs of the Data Object Editor:
   - Levels Tab on page 14-42
   - Hierarchies Tab on page 14-42

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 box. Specify the details of the Type 2 SCD as described in "Type 2 Slowly Changing Dimension Dialog Box" on page 14-46.
9. Provide information about the Storage Tab of the Data Object Editor.

Type 2 Slowly Changing Dimension Dialog Box

Use the Type 2 Slowly Changing Dimension dialog box to specify the effective date attribute, expiration date attribute, and the versioned attribute. This dialog box displays a table that contains the following columns: Levels, Identifying Attribute, Data Type, and Record History.

- **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 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 the `PRODUCTS` Type 2 SCD, the attributes that store the effective date and expiration date are `EFFECTIVE_TIME` and `EXPIRATION_TIME` respectively. 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 **Trigger history** under the Record History column. 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:

1. From the 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 about the Name tab of the Data Object Editor as described in "Name Tab" on page 14-40.
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 14-41.

   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 about the following tabs of the Data Object Editor:
   - Levels Tab on page 14-42
   - Hierarchies Tab on page 14-42
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 box. Specify the details of the Type 2 SCD using this dialog box as described in "Type 3 Slowly Changing Dimension Dialog Box" on page 14-47.
8. Provide information about the Storage Tab of the Data Object Editor.

Type 3 Slowly Changing Dimension Dialog Box

Use the Type 3 Slowly Changing Dimension dialog box to specify the implementation details. Use this dialog box 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.

This dialog box 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. Use the 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** list.

  The surrogate ID of a level cannot be versioned.

Consider the `PRODUCTS` Type 3 SCD. 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** column, 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

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 about the tabs in the Data Object Editor, see "Using the Data Object Editor" on page 14-39.

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 about deployment options, see "Deployment Options for Dimensions" on page 14-48.

   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".

   Visible: This property is not used in code generation.

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 13-45.

Deployment Options for Dimensions

Use the **Deployment Option** configuration property to specify how the dimension metadata is deployed. The options you can select for a dimension are:

Following are the options that you can set for this property.

- Deploy All
- Deploy Data Objects Only
- Deploy to Catalog Only

For more information about these options, see "Deploying Dimensional Objects" on page 14-7.

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 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 14-50
   - Storage Type Page on page 14-50
   - Dimensions Page on page 14-51
   - Measures Page on page 14-51
   - Summary Page on page 14-52

Name and Description Page

Use the Name and Description page to describe the cube. Enter the following details on this page:

- **Name:** The name of the cube. 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:

- 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, the implementation table used to store the cube data is created.

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, the name used to store the cube in the analytic workspace is generated. If no analytic workspace exists, one is created 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 workspace. 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, only dimensions that have a relational implementation are listed. 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 for character data types only.
- **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

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 workspace:

- The cube object.
- The definition of the table that stores the cube data.

  For a relational or ROLAP implementation, the definition of the database table that stores the cube data is created. Additionally, foreign keys are created 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, the analytic workspace that stores the cube data is created. The wizard only creates the definitions for these objects in the workspace. 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 14-5.

Note: When you delete a cube, the associated objects such as the database table or analytic workspace are 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, the following defaults are used:

- **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 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 box.
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 about the following tabs of the Cube Details panel:

   - Name Tab on page 14-53
   - Storage Tab on page 14-53
   - Dimensions Tab on page 14-54
   - Measures Tab on page 14-56
   - Aggregation Tab on page 14-59
   - Data Viewer Tab on page 14-60

   When you use the Data Object Editor to create a cube, the physical objects that store the cube data are not automatically created. 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 14-60. You need to perform this step only for cubes that use a ROLAP implementation.

Name Tab

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, see *Oracle Database Data Warehousing Guide*.

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. 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 box. This dialog box 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, the analytic workspace is stored in the default users tablespace of the owner of the Oracle module.

Dimensions Tab

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 box. Since you can define sparsity only for MOLAP cubes, the Advanced button is enabled only if the Storage type is MOLAP. For more information about the Sparsity dialog box, see "Advanced Dialog Box" on page 14-54.

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 box. This dialog box 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, only those dimensions that use a relational implementation are displayed. If you define a MOLAP implementation for the cube, only the dimensions that use a MOLAP implementation are displayed.
- **Level:** The **Levels** displays all the levels in the dimension selected in the Dimension field. Select the dimension level that the cube references.
- **Role:** The **Role** 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 Box Use the Advanced dialog box to specify the sparsity of the dimensions that the cube references. Sparsity is applicable for MOLAP cubes only. For more information about sparsity, see *Oracle OLAP User's Guide*.

This dialog box 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 box, you must change the order in which the dimensions are listed on the Dimensions Tab of the Data Object Editor.
- **Sparsity:** Sparsity specifies that the cube data is sparse along a particular dimension. Select **Sparsity** for a dimension reference if the cube data is sparse

along that 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, Products, Promotions, and Channels. This is the order in which the dimensions are listed in the Advanced dialog box. The cube data is sparse along the dimensions Promotions and Channels, with Promotions being the most sparse. Then all these dimensions should appear as a group in the following order: Times, Products, Channels, and Promotions. You cannot have any other dimension listed in between these dimensions.

Use the following guidelines to order dimensions:

- List the time dimension first to expedite data loading and time-based analysis. Time is often a dense dimension, although it may be sparse if the base level is Day or the cube has many dimensions.
- List the sparse dimensions in order from the one with the most members to the one with the least. For a compressed cube, list the sparse dimensions in order from the one with the least members to the one with the most.

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, see *Oracle OLAP User's Guide*.

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

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 about the Calculated Measure wizard, see "Calculated Measure Wizard" on page 14-56.

  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:

    (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:

    (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:

(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:

    (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:

    ((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:

    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 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 about the Expression Builder, see "The Expression Builder" on page 18-3.

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, aggregation is performed for every alternate level starting from the lowest level. The default aggregate function is SUM. For more details on the strategies for summarizing data, see the chapter about summarizing data in the *Oracle OLAP User's Guide*.

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 box is displayed. Use this dialog box 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, aggregation is implemented 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 created to implement ROLAP aggregation are not displayed under the Materialized Views node in the Project Explorer.

Data Viewer Tab

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 display the query builder. 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, the measures and dimension references of the cube are automatically mapped 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 box.

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 about the auto binding rules, see "Auto Binding" on page 14-4.

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 box. 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 about deploying cubes, see "Deploying Cubes" on page 14-52.

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. This section describes implementation details for MOLAP cubes.

Ragged Cube Data

If you select **Use natural keys from data source** on the Storage tab of a dimension, mapping code (AWXML mapping code) that can handle ragged fact data for any cube that uses this dimension is generated. 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 Box 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 Box 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 The following calculation plans are generated:

- 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 the 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 information about the error, see 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.

Use the tabs of the Data Object Editor to edit the cube definition. For more information about the tabs in the Data Object Editor, see "Using the Data Object Editor" on page 14-52.

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**.

   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 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.

   Visible: This property is not used in code generation.

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, the implementation objects that store the time dimension data are also created.

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 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 14-65
   - Storage Page on page 14-65
   - Data Generation Page on page 14-66
   - Levels Page (Calendar Time Dimension Only) on page 14-66
   - Levels Page (Fiscal Time Dimension Only) on page 14-67
   - Pre Create Settings Page on page 14-67
   - Time Dimension Progress Page on page 14-67
   - Summary Page on page 14-67

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 about these options, see "Storage Type Page" on page 14-33.

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 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 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, the following two options are provided 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 about using the Data Object Editor, see "Editing Time Dimension Definitions" on page 14-68.

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. 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 workspace. 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, the table, sequence, and the mapping associated with the time dimension are not deleted. 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, the following defaults are used:

- **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:** A standard hierarchy that contains all the levels listed on the Levels page of the Create Dimension wizard is created. 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 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 about the following tabs:
   - Name Tab on page 14-68
   - Storage Tab on page 14-69
   - Attributes Tab on page 14-69
   - Levels Tab on page 14-70
   - Hierarchies Tab on page 14-70

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.

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

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 13-6.
- **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 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 list.

Storage Tab

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, the analytic workspace is stored 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. Use the **Select** to the right of this field to select a sequence from the Available Sequences dialog box. This dialog box 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 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 about the contents of the Levels tab, see "Level Attributes Page" on page 14-36.

Hierarchies Tab

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 recreate 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 box is displayed. Use this dialog box 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.

15

Data Transformation

One of the main functions of an Extract, Transformation, and Loading (ETL) tool is to transform data. This chapter provides an overview of data transformation in Warehouse Builder.

This chapter contains the following topics:

- About Data Transformation in Warehouse Builder
- About Mappings
- About Operators
- About Transformations
- About Transformation Libraries

About Data Transformation in Warehouse Builder

After you import your source data and define the target, you can consider how to transform the source data into the output desired for the target. In Warehouse Builder, you specify how to transform the data by designing *mappings* in the Mapping Editor. A mapping is a Warehouse Builder entity that describes the sequence of operations required to extract data from sources, transform the data, and load the data into one or more targets.

The fundamental unit of design for a mapping is the *operator*. For each distinct operation you want to perform in the mapping, you represent that operation with an operator. To indicate the order of operations, you connect the mappings with data flow connections.

To specify data transformation in a mapping, select from the many prebuilt *transformation operators* or design a new transformation operator. The prebuilt transformation operators enable commonly performed operations such as filtering, joining, and sorting. Warehouse Builder also includes prebuilt operators for complex operations such as merging data, cleansing data, or profiling data.

If none of the prebuilt transformation operators meet your needs, you can design a new one. You can design the new transformation operator based on the Oracle database library of PL/SQL functions, procedures, package functions, and package procedures.

Extraction and loading operations are represented by any of the numerous *source and target operators*. A table operator represents a table and a flat file operator represents a flat file. Whether that operator specifies an extraction or loading operation depends on how you connect the operator relative to other operators in the mapping.

An important distinction to understand is the difference between the operator in the mapping and the object it represents. The operator and the object are separate entities until you *bind* the two together. For example, when you add a table operator to a mapping, you can bind that operator to a specific table in the repository. With the operator bound to the table, you can *synchronize* changing definitions between the two. If the table operator represents a target and you change the operator in the mapping, then you can propagate those changes back to the table in the repository. If the operator represents a source that incurred a change in its metadata definition, then you can reimport the table in the Design Center and then propagate those changes to the table operator in the Mapping Editor.

About Mappings

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.

About Operators

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 "Understanding Performance and Advanced ETL Concepts" on page 22-1.

- **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 17-34.
- **ABAP Mappings:** When you define a SAP/R3 source, Warehouse Builder generates ABAP code. For mapping design considerations for SAP sources, see Chapter 7, "Retrieving Data From SAP Applications" on page 7-1.

Types of Operators

As you design a mapping, you select operators from the Mapping Editor palette 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 17-31.
- 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 15–1 lists each source and target operator alphabetically, gives a brief description.

Table 15–1 Source and Target Operators

Icon	Operator	Description
	Constant operator	Produces a single output group that can contain one or more constant attributes.
	Construct Object operator	Produces object types and collection types.
	Cube operator	Represents a cube that you previously defined.
	Data Generator operator	Provides information such as record number, system date, and sequence values.
	Dimension operator	Represents a dimension that you previously defined.
	Expand Object operator	Expands an object type to obtain the individual attributes that comprise the object type.
	External Table operator	Represents an external table that you previously defined or imported.

Table 15–1 (Cont.) Source and Target Operators

Icon	Operator	Description
	Flat File operator	Represents a flat file that you previously defined or imported.
	Materialized View operator	Represents a materialized view that you previously defined.
	Sequence operator	Generates sequential numbers that increment for each row.
	Table operator	Represents a table that you previously defined or imported.
	Varray Iterator operator	Iterates through the values in the table type.
	View operator	Represents a view that you previously defined or imported.

Data Flow Operators

Use data flow operators to transform data in a mapping.

Table 15–2 lists each data flow operator alphabetically, gives a brief description. For more information on these transformation operators, see "Data Flow Operators" on page 18-1.

Table 15–2 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.
	Deduplicator operator	Removes duplicate data in a source by placing a DISTINCT clause in the select code represented by the mapping.
	Expression 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.

Table 15–2 (Cont.) Data Flow Operators

Icon	Operator	Description
	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 15–3 lists the Pre/Post Process operators and the Mapping Parameter operators.

Table 15–3 Pre/Post Processing Operators

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

Pluggable Mapping Operators

A pluggable mapping is a reusable grouping of mapping operators that behaves as a single operator.

Table 15–4 lists the Pluggable Mappings operators.

Table 15–4 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.

Table 15–4 (Cont.) Pluggable Mapping Operators

Icon	Operator	Description
	Pluggable Mapping Output Signature operator	A combination of output attributes that flow out of the pluggable mapping.

About Transformations

Transformations are PL/SQL functions, procedures, packages, and types 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 workspace and can be used in the project in which they are defined.

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 15-8.

Predefined transformations are organized into the following categories:

- Administration
- Character
- Control Center
- Conversion
- Date
- Numeric
- OLAP
- Others
- SYS
- Spatial
- Streams
- XML

For more information about the transformations that belong to each category, see "Oracle Warehouse Builder Transformations" on page 19-1.

Custom Transformations

A custom transformation is one this 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. This category is available under the Custom node of the Public Transformations node in 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. This category is available under the Custom node of the Public Transformations node in the Global Explorer. It 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:** The Packages category contains packages, which in turn contain functions, procedures, and PL/SQL types. This category is available under the Custom node of the Public Transformations node in the Global Explorer. It is also automatically created under the Transformations node of each Oracle module in the Global Explorer.

  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.

- **PL/SQL Types:** The PL/SQL Types category contains any standalone PL/SQL types. This includes PL/SQL record types, REF cursor types, and nested table types. The PL/SQL Types category is automatically created in each package that you define using the Packages node in the Transformations node of the Project Explorer. It is also available under every package that you define in the following path of the Global Explorer: Public Transformations -> Custom -> Packages.

For further instructions, see "Defining Custom Transformations" on page 19-1.

In addition to the above categories, 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 instructions, see "Importing PL/SQL" on page 19-50.

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.

Transformation libraries are available under the Public Transformations node of the Global Explorer in the Design Center.

Types of Transformation Libraries

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 Transformations node. Each category of predefined transformations is represented by a separate node. 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 workspace.

  The transformations in the Global Shared Library are available under the Custom node of the Public Transformations node. Any transformation that you create under this node is available across all projects in the workspace. For information on creating transformations in the Global Shared Library, see "Defining Custom Transformations" on page 19-1.

  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. The Private node under TRANSFORMLIBS contains transformations that are available only in the current project. These transformations are created under the Transformations 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.

- Add Transformation Operator Dialog Box

  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 Function Wizard, Create Procedure Wizard, Edit Function Dialog Box, or Edit Procedure Dialog Box

  The Implementation page on the these wizards or the Implementation tab of these editors enable 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.

16

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:

- Instructions for Defining Mappings
- Creating a Mapping
- Adding Operators
- Editing Operators
- Connecting Operators
- Using Pluggable Mappings
- Setting Mapping Properties
- Setting Operator, Group, and Attribute Properties
- Synchronizing Operators and Workspace Objects
- Example: Using a Mapping to Load Transaction Data
- Debugging a Mapping

Instructions for Defining Mappings

Before You Begin

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

Also 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 16-3
2. Adding Operators on page 16-7

   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 16-2.
3. Editing Operators on page 16-9
4. Connecting Operators on page 16-14

5. Using Pluggable Mappings on page 16-17
6. Setting Mapping Properties on page 16-20
7. Setting Operator, Group, and Attribute Properties on page 16-22
8. "Configuring Process Flows Reference" on page 22-41
9. For PL/SQL mappings, you can also refer to "Best Practices for Designing PL/SQL Mappings" on page 22-1.
10. Debugging a Mapping on page 16-32
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 starts another mapping. For more information, see "Designing Process Flows" on page 20-1.

Deploy the mapping, and any associated process flows you created, and then execute the mapping as described in "Deploying to Target Schemas and Executing ETL Logic" on page 25-1.

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.

   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 16-3.
3. Drag and drop a flat file operator onto the canvas.
4. On the Add Flat File Operator dialog box, 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 16-9.

   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 box 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 16-1.

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 select **New** to create a module for the flat files as described in "Creating Flat File Modules" on page 8-3.
2. Right-click the flat file module and select **Import** to import file definitions as described in "Importing Definitions from Flat Files" on page 8-9.
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 17-34 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" on page 8-3 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 8-26.
6. In the Project Explorer, right-click the external table and select **Configure.** On the Data Files 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 box, 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.
2. Right-click Mappings and then select **New.**

   Warehouse Builder opens the Create Mapping dialog box.
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 16-11.
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 folder.
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.

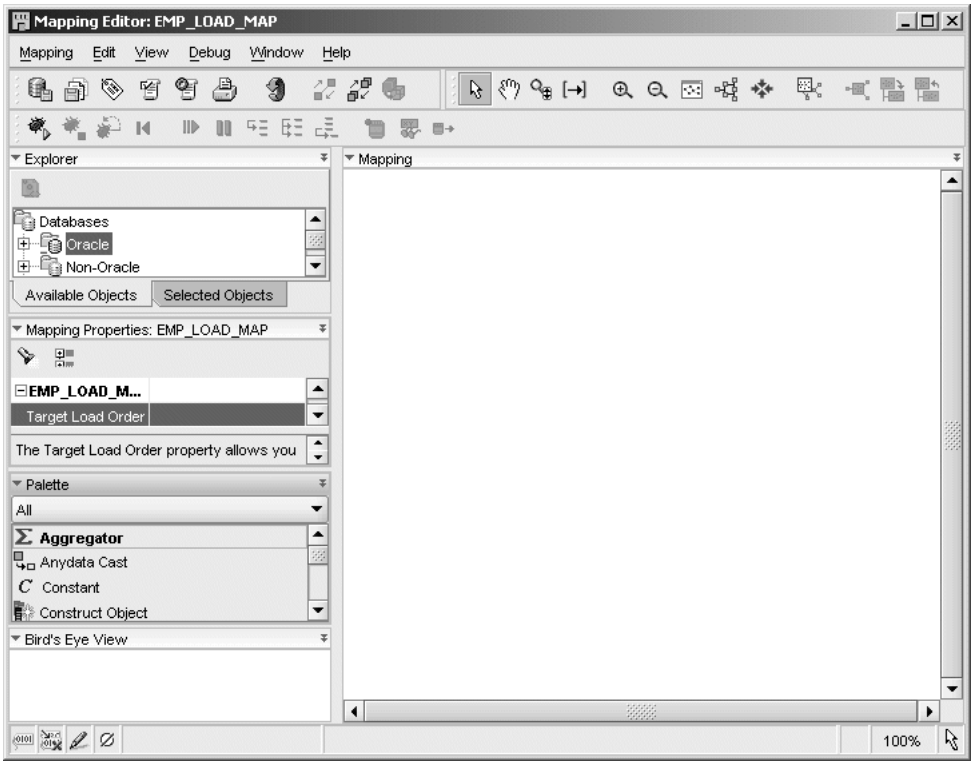
> **Note:** When you open a mapping that was created using OMB Plus, despite the mapping having multiple operators, it may appear to contain only one operator. To view all the operators, click the Auto Layout icon in the Mapping Editor toolbar.

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 16–1 displays the Mapping Editor canvas.

Figure 16–1 Mapping Editor Canvas



This screenshot displays the Mapping Editor. At the top is the Title Bar, below which is the Menu Bar. Below the Menu Bar are two rows of toolbars used to perform various operations in the Mapping Editor.

Below the toolbars, the Mapping Editor is divided into two halves. To the left-hand side are the following panes from top to bottom: Explorer, Mapping Properties, Palette, and Bird's Eye View. To the right-hand side is the Mapping Canvas. At the bottom of the Mapping Editor window is the Indicator Bar.

****

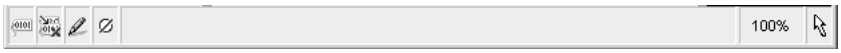
Standard Editor Components

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.

  Figure 16–2 displays the Indicator Bar of the mapping Editor.

Figure 16–2 Indicator Bar on the Mapping Editor



This screenshot shows the Indicator Bar on the Mapping Editor. This indicator bar is located at the bottom of the editor window. This indicator bar has four indicator icons to the left-hand side of the bar. They are: Naming Mode: Physical, Rename Mode, "Read/Write", and validation indicator. On the right-hand side are the following indicators: percentage and arrow.

****

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 start 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 start 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 start an editor, Warehouse Builder displays the palette along the left side and it contains operators 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'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.

Data Viewer

The Data Viewer enables you to view the data stored in the data object. See "Data Viewer" on page 13-8 for more information about the Data Viewer.

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 generated scripts 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 call 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 call 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**. Warehouse Builder displays the Options dialog box that enables you to set display options for the Mapping Editor canvas.

The Options dialog box contains the following options. You can either select or deselect any of these options.

- **Input Connector:** Select this option to display an arrow icon on the left of attributes that you can use as input attributes.
- **Key Indicator:** Select this option to display a key icon to the left of the attribute that is a foreign key attribute in an operator.
- **Data Type:** Select this option to display the data type of attributes in all operators.
- **Output Connector:** Select this option to display an arrow icon on the right of attributes that you can use as output attributes.
- **Enable Horizontal Scrolling:** Select this option to enable horizontal scrolling for operators.
- **Automatic Layout:** Select this option to use an automatic layout for the 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 workspace 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 workspace 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 workspace. The separate versions are said to be *bound* together. That is, the version in the mapping is bound to the version in the workspace.

To distinguish between the two versions, this chapter refers to objects in the workspace either generically as *workspace objects* or specifically as *workspace tables*, *workspace 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 workspace as the *workspace dimension*.

Warehouse Builder maintains separate workspace objects for some operators so that you can synchronize changing definitions of these objects. For example, when you reimport a new metadata definition for the workspace 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 workspace table. You can accomplish these tasks by a process known as synchronizing. In Warehouse Builder, you can synchronize automatically as described in "Managing Metadata Dependencies" *Oracle Warehouse*

Builder Installation and Administration Guide. Alternatively, synchronize manually from within the Mapping Editor as described in "Synchronizing Operators and Workspace Objects" on page 16-22.

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 Palette and drop it onto the Mapping Editor canvas.

   If you select an operator that you can bind to a workspace object, the Mapping Editor displays the **Add Mapping <operator name>** dialog box. For details on how to use this dialog box, see "Add Operator Dialog Box" on page 16-8.

   If you select an operator that you cannot bind to a workspace object, Warehouse Builder may display a wizard or dialog box 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. 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.

Adding Operators that Bind to Workspace Objects

You can bind the following operators to associated objects in the workspace using the Add Operator Dialog Box:

- Cube operators
- Dimension operators
- External Table operators
- Flat File operators
- Materialized View operators
- Pre Mapping Process operators
- Post Mapping Process operators
- Sequence operators
- Table operators
- Transformation operators
- View operators

Add Operator Dialog Box

When you add an operator that you can bind to a workspace object, the Mapping Editor displays the **Add <operator name> Operator** dialog box. 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 workspace 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 16-9. Next, to create the new workspace 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 16-15.

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 workspace.

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 workspace objects within the same module as the mapping or from other modules. If you select a workspace 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 workspace 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 16-22.

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.

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 16-11.

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.

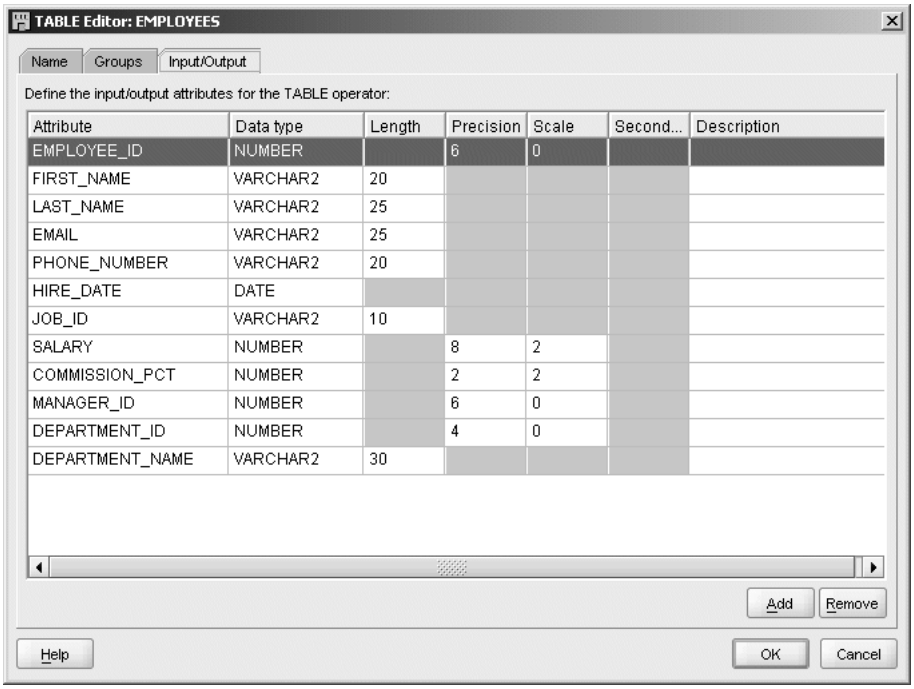
Input and Output Tabs

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 16–3 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 16–3 Input/Output Tab on the Operator Editor



This screenshot shows the Operator Editor window. This window contains three tabs. They are ordered from left to right: Name, Groups, and Input/Output. The Input/Output tab is currently selected.

The Input/Output tab contains a table where you can define the Input/Output attributes for the Table operator. The table contains the following columns: Attribute, Data type, Length, Precision, Scale, Second, and Description. Below the table, on the right hand-side, are the Add and Remove buttons. At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page are the following buttons, ordered from left to right: OK and Cancel.

****

You can add, remove, and edit attributes. The Mapping Editor grays 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 3-7. Warehouse Builder maintains a business and a physical name for each object in the workspace. 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, 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 through 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 Database SQL Language Reference*.

In addition to physical and business names, some operators also have bound names. Every operator associated with a workspace object has a bound name. During code generation, Warehouse Builder uses the bound name to reference the operator to its workspace 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 workspace.
- 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 16–1 describes the default display sets.

Table 16–1 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

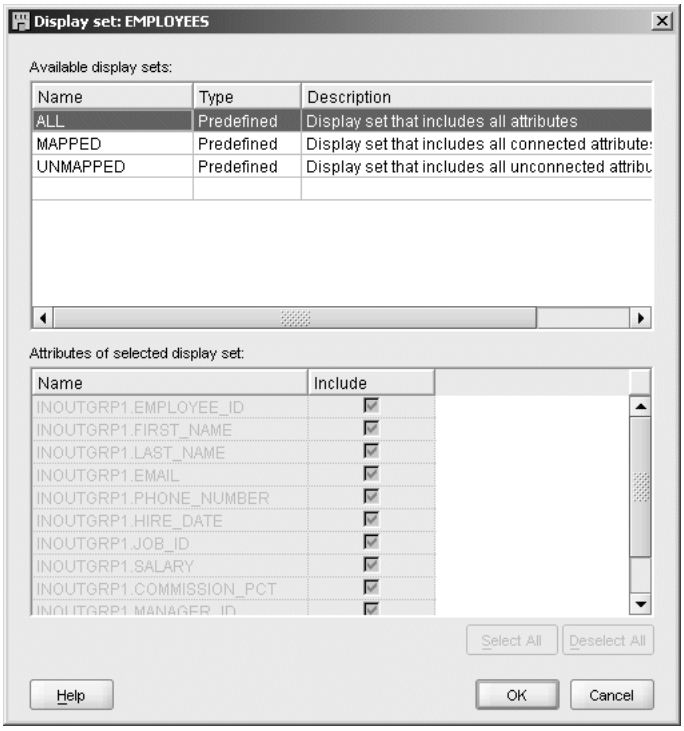
You can define display sets for any operator in a mapping.

To define a display set:

1. Right-click an operator, and select **Display Set**.

   The Display Set dialog box is displayed as shown in Figure 16–4.

Figure 16–4 Display Set Dialog Box



This screenshot shows the Set dialog box. This dialog box is divided into two halves. The upper half contains a box which contains information for the available display set in a tabular format. The table contains the following columns, from left to right: Name, Type, and Description. The lower half displays a table that contains the attributes of the selected display set. This table contains two columns, from left to right: Name, and Include.

At the lower right-hand side of the table are two buttons, ordered from left to right: Select All (currently dimmed) and Deselect All (currently dimmed). At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page, are the following buttons, ordered from left to right: Ok and Cancel.

****

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 and 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.

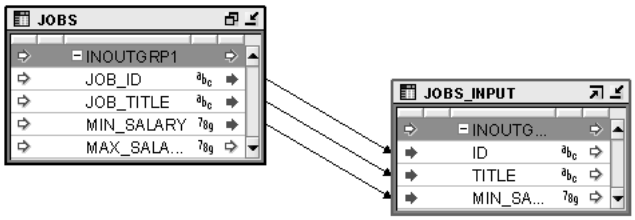
To connect attributes:

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.
3. Release the mouse over the input attribute.
4. Repeat steps one through three until you create all the required data flow connections.

   Figure 16–5 displays a mapping with attributes connected.

Figure 16–5 Connected Operators in a Mapping



This screenshot shows the connected operators in a Mapping. To the left is the JOBS table. To the right is the JOBS_INPUT table. The columns of the JOBS table are connected as inputs for the JOBS_INPUT table using arrows.

****

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 Box" on page 16-16.

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" on page 16-15.

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:

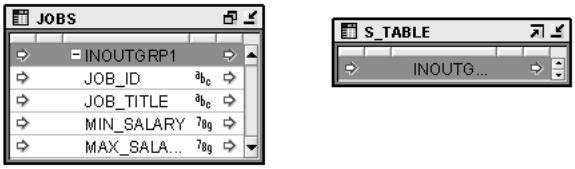
1. In the Mapping Editor, add a source table.

   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 box to select and bind the source table operator in the mapping. From the Add Table Operator dialog box, select **Create unbound operator with no attributes.**

   The mapping should now resemble Figure 16–6 with one source table and one staging area table without attributes.

Figure 16–6 Unbound Staging Table without Attributes and Source Table



This screenshot shows the Unbounded Staging table without attributes and source tables. To the left is JOBS table containing with its columns listed one below another. To the right is the unbound table S_TABLE that does not contain any columns.

****

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 Create And Bind dialog box.
6. In the Create in field, 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 Box

If you connect from one operator to a target operator with existing attributes, the Mapping Editor starts the Connect Operators dialog box.

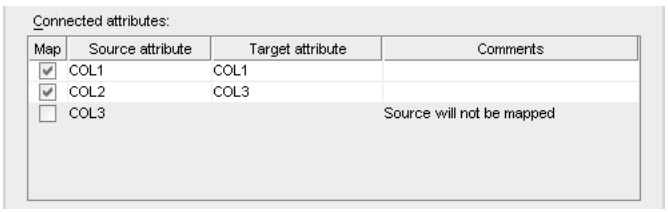
Select one of the following criteria for copying and connecting attributes:

- Copy Source Attributes to Target Group and Match
- Match by Position of Source and Target Attributes
- Match by Name of Source and Target Attributes

After you select one of the three options, select **Go.** The Connect Operators dialog box displays a list of connected attributes.

Figure 16–7 displays the Connected attributes section.

Figure 16–7 Connected Attributes



This screenshot shows the connected attributes. The attributes are listed in a table. The table contains the following columns, from left to right: Map, Source attributes, Target attributes, and Comments. The Map column contains check boxes.

****

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 box. 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. 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.

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 Oracle Warehouse Builder Enterprise ETL Option as described in *Oracle Database Licensing Information*.

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.

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 standalone pluggable mapping:

1. Expand the Pluggable Mappings node in the Project Explorer.
2. Right-click Standalone, and select **New**.
3. 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 box. You can select the desired pluggable mapping and add it to the mapping.

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:

1. Expand the Pluggable Mappings node in the Project Explorer.
2. Right-click Pluggable Mapping Folders, and select **New**. This opens the Create Pluggable Mapping Folder dialog box.
3. Enter a name for the folder and provide a description (optional).
4. Click **OK** to save the folder and exit the wizard.

The folder appears on the Project Explorer. 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. You can also move a pluggable mapping to any folder on the tree.

At the time of creating the Pluggable Mapping folder, 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 option, only the pluggable mapping folder gets created. To create a pluggable mapping under this folder:

1. Under the Pluggable Mappings Folders node, right-click the folder and select **New**.
2. This opens the Create Pluggable Mapping wizard, which guides you through the process of creating a new pluggable mapping.

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 input as well as 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 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
- About the Mapping Editor
- Adding Operators
- Editing Operators
- Connecting Operators
- Setting Operator, Group, and Attribute Properties
- Synchronizing Operators and Workspace Objects

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

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.

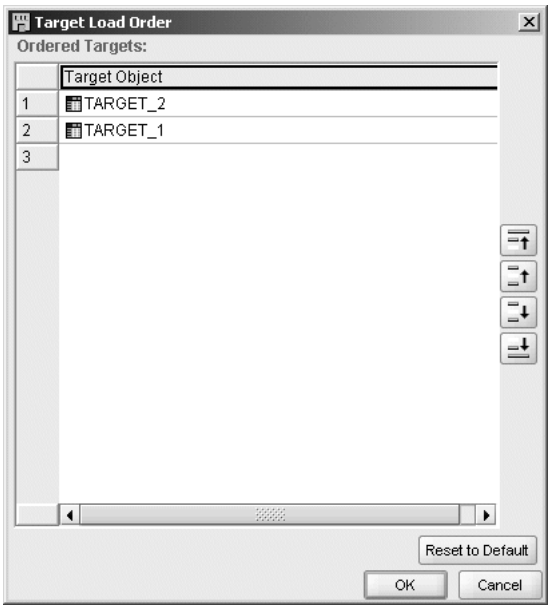
To specify the loading order for multiple targets:

1. Click whitespace in the mapping canvas to view the mapping properties in the Mapping Properties panel in the upper left corner.
2. Go to the Map Targets Load Order property and click the Ellipsis button on the right of this property.

   Warehouse Builder displays the Targets Load Order dialog box which shows TARGET2 loading before TARGET1.

   Figure 16–8 displays the Target Load Order dialog box.

Figure 16–8 Target Load Order Dialog Box



This screenshot shows the Target Load Order dialog box. This dialog box lists the ordered targets in a tabular format. The table has a column named Target Object. To the center right hand-side corner are four buttons, ordered from top to bottom: Move the Target to the Top, Move the Target to one level up, Move the Target to one level down, and Move the Target to the Bottom.

Below the table, on the lower-right corner is the Reset to Default button. Below this button are the following two buttons, ordered from left to right: OK and Cancel.

****

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 panel along the left side.

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 Workspace Objects

Many of the operators you use in a mapping have corresponding definitions in the Warehouse Builder workspace. 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 workspace object.

You have the following choices in deciding the direction in which you propagate changes:

Synchronizing From a Workspace 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 "Managing Metadata Dependencies" in the *Oracle Warehouse Builder Installation and Administration Guide*. 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 Workspace Object to an Operator" on page 16-23.

Synchronizing from an Operator to a Workspace Object: When you make changes to an operator in a mapping, you may want to propagate those changes to its corresponding workspace 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 workspace 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 workspace 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 box displays.
3. By default, Warehouse Builder selects the option for you to synchronize your selected operator with its associated object in the workspace. You can accept the default or select another workspace object from the list box.

   In this step you also specify whether to perform inbound or outbound synchronization. Inbound synchronization synchronizes the data object with the mapping operator. Outbound synchronization synchronizes the mapping operator with the data object.
4. As an optional step, set the Matching Strategies and Synchronize strategy.
5. Click **OK.**

Synchronizing From a Workspace Object to an Operator

In the Mapping Editor, you can synchronize from a workspace object for any of the following reasons:

- **Manually propagate changes:** Propagate changes you made in a workspace object to its associated operator. Changes to the workspace object can include structural changes, attribute name changes, attribute data type changes. To automatically propagate changes in a workspace object across multiple mappings, see "Managing Metadata Dependencies" in the *Oracle Warehouse Builder Installation and Administration Guide*.
- **Synchronize an operator with a new workspace object:** You can associate an operator with a new workspace 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.
- **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 workspace object types such as views, materialized views, or cubes.

Synchronizing Operators based on Workspace Objects

Table 16–2 lists operators and the types of workspace objects from which you can synchronize.

Table 16–2 Operators Synchronized with Workspace Objects

To: Operator	From: Workspace 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 workspace 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 8-27.

Synchronizing from an Operator to a Workspace Object

As you make changes to operators in a mapping, you may want to propagate those changes back to a workspace 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 workspace object for any of the following reasons:

- **Propagate changes:** Propagate changes you made in an operator to its associated workspace 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 workspace objects:** Synchronize to replace an existing workspace object.

Synchronizing from an operator has no impact on the dependent relationship between other operators and the workspace object. Table 16–3 lists the operators from which you can synchronize.

Table 16–3 Outbound Synchronize Operators

Mapping Objects	Create Workspace Objects	Propagate Changes	Replace Workspace Objects	Notes
External Tables	Yes	Yes	Yes	Updates the workspace 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 8-27.
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	Not applicable.
Views	Yes	Yes	Yes	Copies attributes and data types as columns.

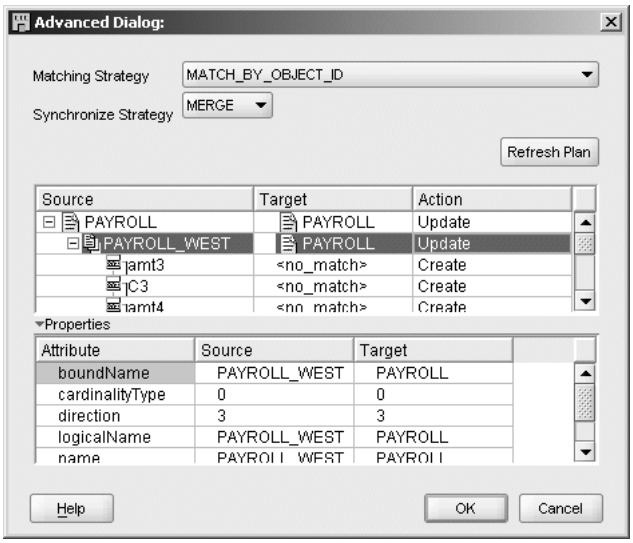
Advanced Options for Synchronizing

Use the Synchronization Plan dialog box 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.

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, in Figure 16–9, 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 16–9 Advanced Synchronizing Options



This screenshot shows the dialog box that enables you to perform advanced synchronization options. There are two lists, at the top of the dialog box, labeled Matching Strategy and Synchronize Strategy. Below the list on the right-hand side is the Refresh Plan button.

Below the Refresh Plan button are two tables, displayed one below the other. The table at the top has the following columns, from left to right: Source, Target, and Action. The table below is labeled Properties and it contains the following columns, from left to right: Attributes, Source, and Target.

At the lower left-hand corner of the dialog box is the Help button. At the lower right-hand corner of the dialog box, are the following buttons, ordered from left to right: Ok and Cancel.

****

Matching Strategies

Set the matching strategies to determine how Warehouse Builder compares an operator to a workspace 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 workspace object. Match by object identifier is not available for

synchronizing an operator and workspace object of different types such as a view operator and a workspace 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 workspace 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 workspace object attributes. Matching is case-sensitive.

Use this strategy if you want bound names to be consistent with physical names in the workspace object. You can also use this strategy with a different workspace object if there are changes in the workspace object that would change the structure of the operator.

Warehouse Builder removes attributes of the operator that cannot be matched with those of the workspace object. Attributes of the selected workspace 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 workspace 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 workspace object by position. The first attribute of the operator is synchronized with the first attribute of the workspace object, the second with the second, and so on.

Use this strategy to synchronize an operator with a different workspace object and you want to preserve the business names in the operator attributes. This strategy is most effective when the only changes to the workspace 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.

Example: Using a Mapping to Load Transaction Data

Scenario

Your company records all its transactions as they occur, resulting in inserts, updates, and deletes, in a flat file called `record.csv`. These transactions must be processed in the exact order they were stored. For example, if an order was first placed, then updated, then canceled and re-entered, this transaction must be processed exactly in the same order.

An example data set of the source file `record.csv` is defined as:

```
Action,DateTime,Key,Name,Desc
I,71520031200,ABC,ProdABC,Product ABC
I,71520031201,CDE,ProdCDE,Product CDE
I,71520031202,XYZ,ProdXYZ,Product XYZ
U,71620031200,ABC,ProdABC,Product ABC with option
```

```
D,71620032300,ABC,ProdABC,Product ABC with option
I,71720031200,ABC,ProdABC,Former ProdABC reintroduced
U,71720031201,XYZ,ProdXYZ,Rename XYZ
```

You want to load the data into a target table such as the following:

```
SRC_TIMESTA KEY NAME    DESCRIPTION
----------- --- ------- --------------------------
71520031201 CDE ProdCDE Product CDE
71720031201 XYZ ProdXYZ Rename XYZ
71720031200 ABC ProdABC Former ProdABC reintroduced
```

Solution

Warehouse Builder enables you to design ETL logic and load the data in the exact temporal order in which the transactions were stored at source. To achieve this result, you design a mapping that orders and conditionally splits the data before loading it into the target. Then, you configure the mapping to generate code in row-based operating mode. In row-based operating mode, Warehouse Builder generates code to process the data row by row using if-then-else constructions, as shown in the following example.

```
CURSOR
   SELECT
       "DATETIME$1"
   FROM
       "JOURNAL_EXT"
   ORDER BY "JOURNAL_EXT"."DATETIME" ASC
LOOP
  IF "ACTION" = 'I' THEN
      INSERT this row
  ELSE
   IF "ACTION" = 'U' THEN
      UPDATE this row
  ELSE
          DELETE FROM
                "TARGET_FOR_JOURNAL_EXT"
END LOOP;
```

This ensures that all consecutive actions are implemented in sequential order and the data is loaded in the order in which the transaction was recorded.

Case Study

This case study shows you how to create ETL logic to load transaction data in a particular order using Warehouse Builder.

Step 1: Import and Sample the Source Flat File, record.csv

In this example, the flat file `record.csv` stores all transaction records and a timestamp. Import this flat file from your source system using the Warehouse Builder Import Metadata Wizard. Proceed to define the metadata for the flat file in Warehouse Builder using the Flat File Sample Wizard.

> **Note:** You can replace this flat file with a regular table if your system is sourced from a table. In this case, skip to Step 3.

Step 2: Create an External Table

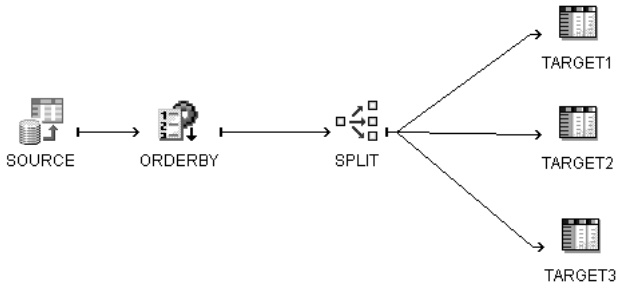
To simplify the use of sampled flat file object in a mapping, create an external table (JOURNAL_EXT) using the Create External Table Wizard, based on the flat file imported and sampled in Step 1.

The advantage of using an external table instead of a flat file is that it provides you direct SQL access to the data in your flat file. Hence, there is no need to stage the data.

Step 3: Design the Mapping

In this mapping, you move the transaction data from an external source, through an operator that orders the data, followed by an operator that conditionally splits the data before loading it into the target table. Figure 16–10 shows you how the source is ordered and split.

Figure 16–10 ETL Design



This screenshot shows a mapping. The operators, from left to right, are as follows: SOURCE, ORDERBY, and SPLIT. The SOURCE operator is connected to ORDERBY operator using an arrow. An arrow connects the ORDERBY operator to the SPLIT operator.

On the right, ordered from top to bottom, are three Table operators: TABLE_1, TABLE_2, and TABLE_3. Arrows connect SPLIT to TARGET1, TARGET2, and TARGET3.

****

The Sorter operator enables you to order the data and process the transactions in the exact order in which they were recorded at source. The Splitter operator enables you to conditionally handle all the inserts, updates, and deletes recorded in the source data by defining a split condition that acts as the if-then-else constraint in the generated code. The data is conditionally split and loaded into the target table. In this mapping, the same target table is used three times to demonstrate this conditional loading. The mapping tables TARGET 1, TARGET 2, and TARGET 3 are all bound to the same workspace table TARGET. All the data goes into a single target table.

The following steps show you how to build this mapping.

Step 4: Create the Mapping

Create a mapping called LOAD_JOURNAL_EXT using the Create Mapping dialog box. Warehouse Builder then opens the Mapping Editor where you can build your mapping.

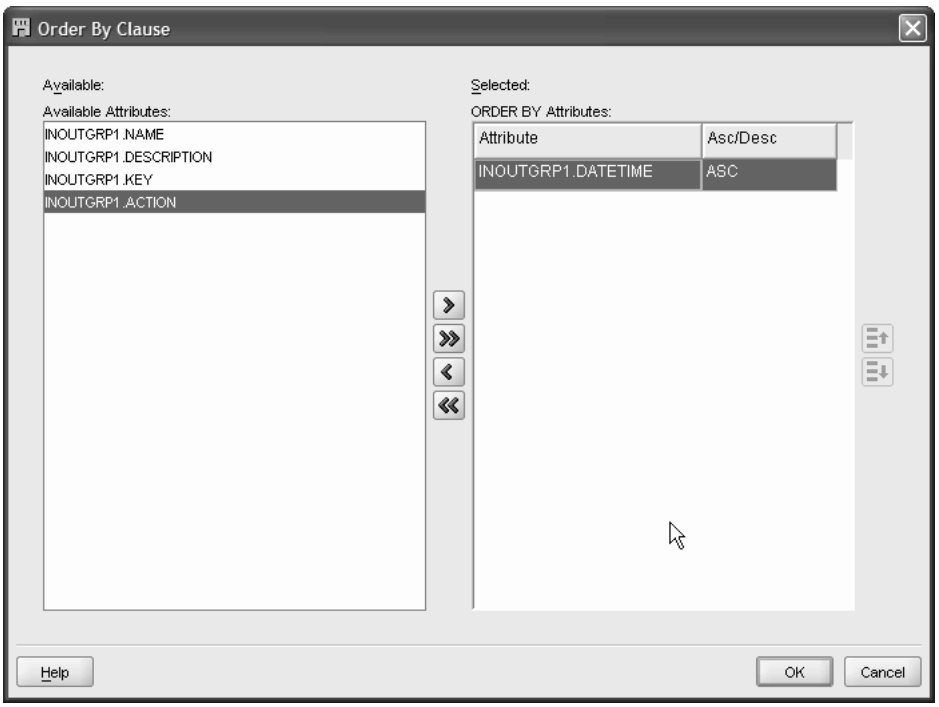
Step 5: Add an External Table Operator

Drag and drop a mapping external table operator onto the mapping editor and bind it to the external table JOURNAL_EXT.

Step 6: Order the Data

Add the Sorter operator to define an order-by clause that specifies the order in which the transaction data must be loaded into the target. Figure 16–11 shows you how to order the table based on the timestamp of the transaction data in ascending order.

Figure 16–11 Order By Clause Dialog Box



This screenshot shows the Order By Clause dialog box. There are two lists: Available Attributes on the left, and ORDERBY Attributes on the right. The Available Attributes list contains a list of attributes. The ORDER BY Attributes list contains two column names, ordered from left to right: Attribute and Asc/Desc. Between the Available Attributes and ORDERBY Attributes lists are the following four buttons, ordered from top to bottom: Move All Right, Move All, Move Left, and Move All Left.To the center right of the ORDER BY are two button, ordered from top to bottom, Up and Down.

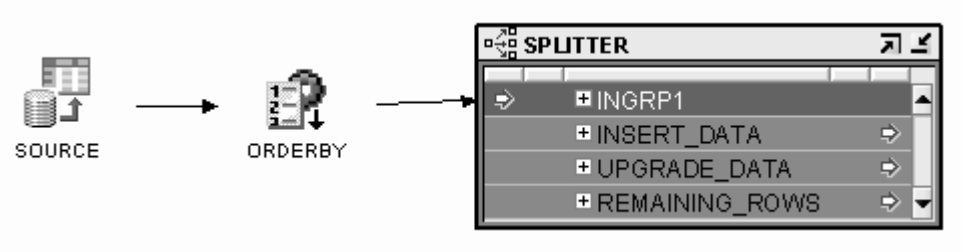
At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page, the following buttons, are ordered from left to right: Ok and Cancel.

****

Step 7: Define a Split Condition

Add the Splitter operator to conditionally split the inserts, updates, and deletes stored in the transaction data. This split condition acts as the if-then-else constraint in the generated code. Figure 16–12 shows how to join the SOURCE operator with the ORDERBY operator which is linked to the SPLITTER operator.

Figure 16–12 Adding the Splitter Operator



This screenshot displays the SOURCE operator which is linked to the ORDERBY operator. The ORDERBY operator is linked to the SPLITTER operator. The SPLITER operator has four collapsible rows, ordered from top to bottom: INGRP1, INSERT_DATA, UPDATE_DATA, REMAINING_ROWS.

****

Define the split condition for each type of transaction. For outgroup INSERT_DATA, define the split condition as `INGRP1.ACTION = 'I'`. For UPDATE_DATA, define the split condition as `INGRP1.ACTION = 'U'`. In Warehouse Builder, the Splitter operator contains a default group called REMAINING_ROWS that automatically handles all Delete ('D') records.

Step 8: Define the Target Tables

Use the same workspace target table three times for each type of transaction: one for INSERT_DATA, one for UPDATE_DATA, and one for REMAINING_ROWS.

Step 9: Configure the Mapping LOAD_JOURNAL_EXT

After you define the mapping, you must configure the mapping to generate code. Because the objective of this example is to process the data strictly in the order it was stored, you must select row-based as the default operating mode. In this mode, the data is processed row by row and the insert, update, and delete actions on the target tables take place in the exact order in which the transaction was recorded at source.

Do not select set-based mode as Warehouse Builder then generates code that creates one statement for all insert transactions, one statement for all update transactions, and a third one for all delete operations. The code then calls these procedures one after the other, completing one action completely before following up with the next action. For example, it first handles all inserts, then all updates, and then all deletes.

To configure a mapping for loading transaction data:

1. From the Project Explorer, right-click the `LOAD_JOURNAL_EXT` mapping and select **Configure**.
2. Expand the Runtime parameters node and set the Default Operating Mode parameter to Row based.

In this example, accept the default value for all other parameters. Validate the mapping before generating the code.

Step 10: Generate Code

After you generate a mapping, Warehouse Builder displays the results in the Generation Results window.

When you inspect the code, you will see that Warehouse Builder implements all consecutive actions in row-based mode. This means that the data is processed row by

row and Warehouse Builder evaluates all conditions in sequential order using if-then-else constructions, as shown in Figure 16–10 on page 16-29. The resulting target table thus maintains the sequential integrity of the transactions recorded at source.

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 Debug Start 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 16-34.
- **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 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 list in the upper right corner, above the input or output groups. Use this 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 box is displayed.
2. In the Define Test Data dialog box, 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 box for more information.

Creating New Tables to Use as Test Data

When you create a new table using the Define Test Data dialog box, 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 box. 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 workspace. 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 box 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.

17

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

Recall that 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 Source and Target Operators

   When you select an object on the canvas, the editor displays its associated properties in the Properties panel along the left side.
7. Configuring Mappings Reference
8. Debugging a Mapping

This chapter describes step 6, Setting Properties for Source and Target Operators, and also provides details on how to use each operator.

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 workspace. 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. 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:** Checks the target for existing rows. If there are no existing rows, the incoming rows are inserted into the target.
- **DELETE:** The incoming row sets are used to determine which of the rows on the target are to delete.
- **DELETE/INSERT:** Deletes all rows in the target and then inserts the new rows.
- **INSERT:** 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, the insert operation is performed 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 22-32 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:** No operation is performed on the target. This setting is useful for testing. Extraction and transformations run but have no effect on the target.
- **TRUNCATE/INSERT:** Truncates the target and then inserts the incoming row set. If you select this option, the operation cannot be rolled back even if the execution of the mapping fails. Truncate permanently removes the data from the target.
- **UPDATE:** 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, the target is updated 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, the target is updated in row based mode.
- **UPDATE/INSERT:** For each incoming row, the update is performed first. If you select UPDATE/INSERT and the Default Operating Mode on page 22-32 for the target is set-based, a MERGE statement is generated.

For dimensions and cubes, the Loading Type property has the 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 Oracle 9i *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 17–1 lists the INTO TABLE clauses associated with each load type and their affect on data in the existing targets.

Table 17–1 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 list containing the constraints defined on that operator and the built-in loading options.

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 17–2.

Table 17–2 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 17–4.

Table 17–3 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.** A list containing the loading options is displayed. Warehouse Builder defaults the settings based on the constraint type you select.

For example, if you want to reset the match properties for all key attributes, click **Advanced,** select **No Constraints,** and click **OK.** The manual load settings are overwritten and the data is loaded based on the settings displayed in Table 17–4.

Table 17–4 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
Match Column when Deleting Row	NO	NO

Alternatively, if you click **Advanced** and select **All Constraints,** the manual load settings are overwritten and data is loaded based on the settings displayed in Table 17–5

Table 17–5 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 workspace 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, an error is displayed. 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 a matching row is located. An update operation is performed on the target attribute using the data of the source attribute. Table 17–6 lists the update operations you can specify and describes the update operation logic.

Table 17–6 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.

This image displays the icon for the operator described in the surrounding text.

****

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 about public transformations, see Chapter 15, "Data Transformation".

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 box 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 box.
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 box is displayed. Use this dialog box 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.

This image displays the icon for the operator described in the surrounding text.

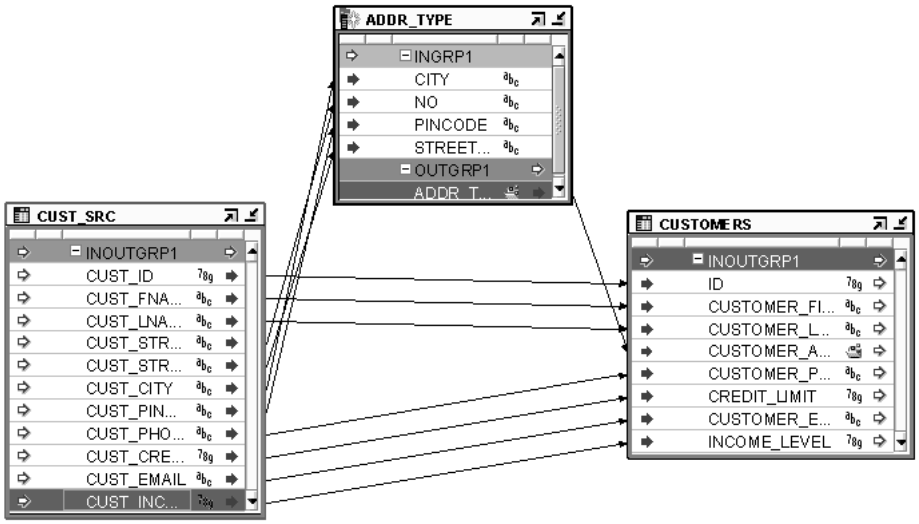
****

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 17–1 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 workspace.

Figure 17–1 Construct Object Operator in a Mapping



The description of this image is contained in the surrounding text.

****

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 box to create or select an object. For more information about these options, see Adding Operators that Bind to Workspace Objects on page 16-8.
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

Use the Cube operator to source data from or load data into cubes.

This image displays the icon for the operator described in the surrounding text.

****

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 box.

2. Use the Add Cube dialog box to create or select a cube. For more information about these options, see "Adding Operators that Bind to Workspace Objects" on page 16-8.

   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. For example, when you map the attributes from the dimension operator to the cube operator, a Key Lookup operator is created 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.

The Cube operator contains an attribute called ACTIVE_DATE. This attribute represents the point in time that is used to determine which record in a Type 2 SCD is the active record. This property is applicable only when the cube you are loading has one or more Type 2 SCDs.

If you do not map an attribute from the source to the ACTIVE_DATE, `SYSDATE` is used as the default.

If you map a source attribute to ACTIVE_DATE, the value of the source attribute is used to determine which version of the Type 2 SCD record is referenced by the cube record.

For any cube that references a dimension in which the level is of a Type 2 SCD, the WHERE clause generated to determine the dimension member is as follows:

```
...
WHERE
(...
   (<dim_name>.DIMKEY = <lookup_for_dimension_dimkey> AND
           (<level>_EFFECTIVE_DATE <= ACTIVE_DATE AND
                    <level>_EXPIRATION_DATE >= ACTIVE_DATE) OR
           (<level>_EFFECTIVE_DATE <= ACTIVE_DATE AND
                    <level>_EXPIRATION_DATE IS NULL))
...)
```

If a mapping that loads a cube references at least one Type 2 SCD that has the Default Expiration Time of Open Record set to a non-NULL value, then the ACTIVE_DATE attribute of the Cube operator must be mapped from the source that contains the date value that defines the range for the dimension record.

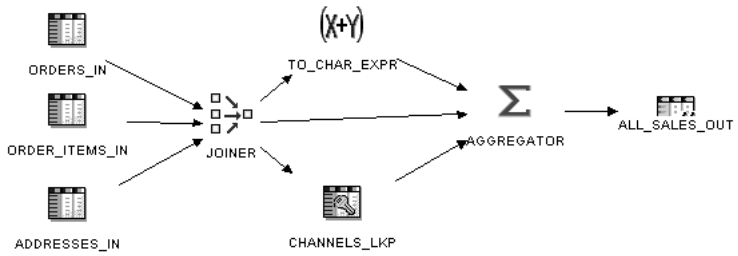
If the ACTIVE_DATE attribute is not mapped from the source, then the SYSDATE value will define the date range for the dimension record.

When the ACTIVE_DATE attribute is mapped from the source, the source attribute value is used to perform a range comparison to determine which dimension record should be loaded.

The logic used to perform the lookup for the dimension member is described in the WHERE clause listed previously.

Figure 17–2 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 17–2 Mapping that Loads a Cube



The description for this image is contained in the surrounding text.

****

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.

This image displays the icon for the operator described in the surrounding text.

****

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:

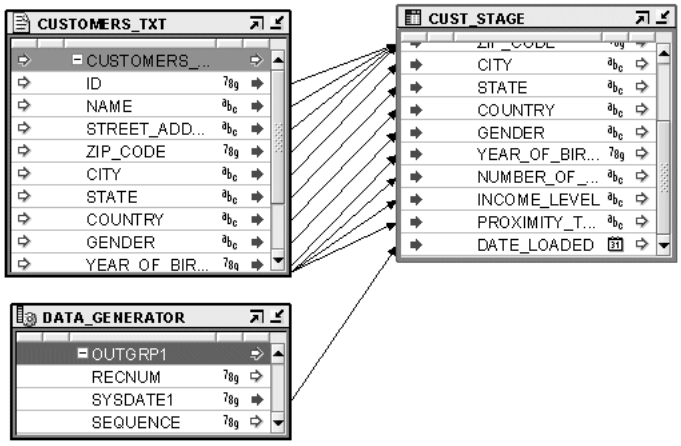
- RECNUM
- SYSDATE
- SEQUENCE

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 17–3 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 17–3 Data Generator in a Mapping



This image displays a mapping that uses the Data Generator operator. The SYSDATE1 output attribute of the Data Generator operator is mapped to the DATE_LOADED attribute of the target table.

****

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 17–7 lists the options available for sequence values.

Table 17–7 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.
incr	The value that the sequence number is to increment after a record is loaded or rejected.

If records are rejected during loading, the sequence of inserts is preserved 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, the valid rows with assigned numbers 10, 14, and 16, not 10, 12, 14 are inserted. When you correct the rejected data and reinsert it, you can manually set the columns to match the sequence.

Dimension Operator

Use the Dimension operator to source data from or load data into dimensions and slowly changing dimensions.

This image displays the icon for the operator described in the surrounding text.

****

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 workspace. To avoid errors in the generated code, ensure that the workspace dimension is deployed successfully before you deploy the mapping that contains the Dimension operator. To synchronize a Dimension operator with the workspace 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 box.

2. Use the Add Dimension dialog box 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, a lookup is performed 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 Initial Record This property is applicable for Type 2 SCDs only. It represents the default value assigned as the effective time for the initial load of a particular dimension record. The default value set for this property is `SYSDATE`.

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 open records, after the initial record. The default value of this property is `SYSDATE`. This value should not be modified.

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.

Type 2 Gap v This property is applicable for Type 2 SCDs only. It represents the time interval between the expiration time of an old record and the effective time of the current record when a record is versioned.

When the value of a triggering attribute is updated, the current record is closed and a new record is created with the updated values. Since the closing of the old and opening of the current record occur simultaneously, it is useful to have a time interval between the expiration time of the old record and the effective time of the open record, instead of using the same value for both.

Type 2 Gap Units This property is applicable for Type 2 SCDs only. It represents the unit of time used to measure the gap interval represented in Type2 Gap property. The options are: Seconds, Minutes, Hours, Days, and Weeks.

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.

Note: If you set the Commit Control property to Manual, ensure that you set the Automatic Hints Enable property to false. Otherwise, your mapping may not execute correctly.

Example of Type 2 SCD Operator Property Values

For example, the mapping that loads the `Products` Type 2 SCD has the following configuration properties:

Default Effective Time of Initial Record: 01-jan-2000

Default Effective time of Open Record: SYSDATE

Default Expiration Time of Open Record: 01-jan-2099

Type2 Gap: 1

Type2 Gap Units: MINUTE

`Product` is the leaf level in this Type 2 SCD. The data type of the effective date and expiration date attributes is `TIMESTAMP`. The first product record is loaded on 20-apr-2007. The product record uses the following values:

Effective Time: 01-jan-2000

Expiration Time: 01-jan-2099

When the triggering attribute of this product record is updated on 22-apr-2007 at 10:45 AM, the current product record is closed and a new record is opened with the updated values.

The closed product record has the following values:

Effective Time: 01-jan-2000

Expiration Time: 22-apr-2007 10:44:00

The currently open `Product` record has the following values:

Effective Time: 20-apr-2007 10:45:00

Expiration Time: 01-jan-2099

Dimension Operator as a Source

You can source data stored in a workspace 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 workspace dimension that contains the source data. Then map the attributes from the dimension levels to the target operator. See Figure 17–2 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

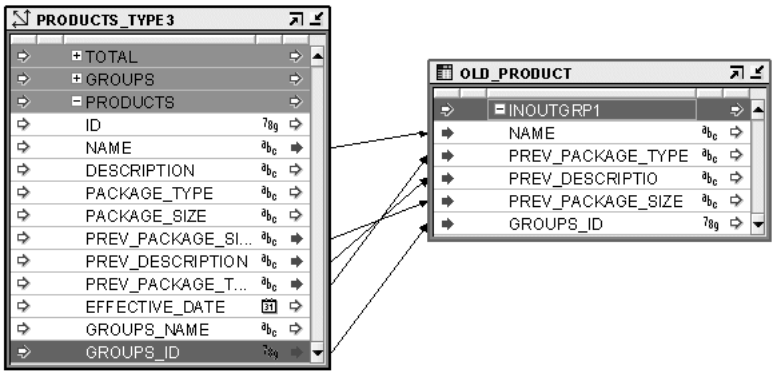
version of a level record, set the Extracting Type property to **Extract Current Only (Type 2 Only)**.

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 workspace 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 17–4 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 17–4 Mapping that Sources Data from a Type 3 SCD



This image displays a mapping that sources data stored in a Type 3 SCD and loads this data into a table.

****

Dimension Operator as a Target

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.

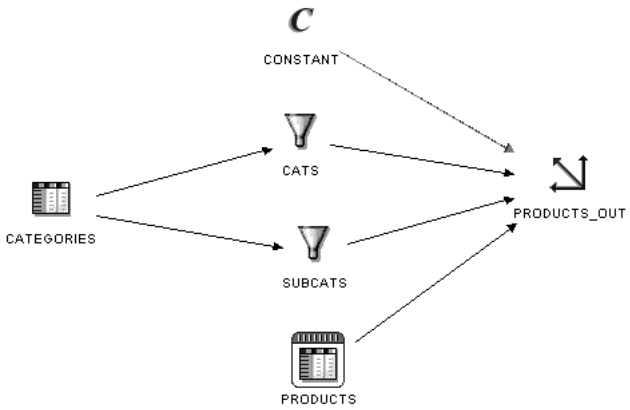
Warehouse Builder loads data into the dimension starting from the highest level.

> **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, a check is made to determine 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, the current record is updated to reflect the new attribute values.

Figure 17–5 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 17–5 Loading the Products Dimension



The description of the image is included in the surrounding text.

****

When you define a data flow to load a dimension, an in-line pluggable mapping that actually loads data into the dimension is created. 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.

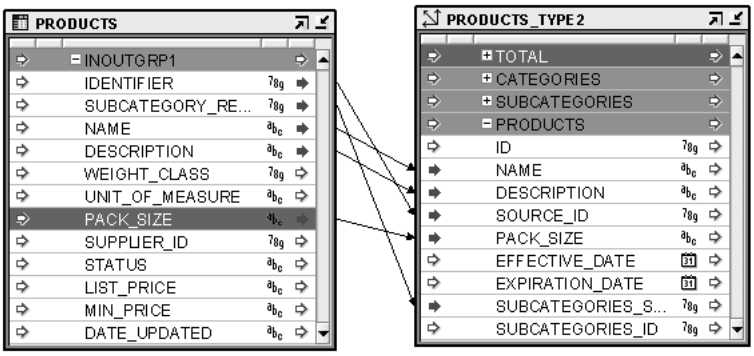
Loading Data Into Type 2 SCDs Before loading records into the Type 2 SCD, a check is first made to determine if a record with the same business identifier exists in the Type 2 SCD. If the record does not exist, it is added to the Type 2 SCD. If the record already exists, the following steps are performed:

- 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 17–6 displays a mapping that loads data into the PRODUCTS level of a Type 2 SCD.

Figure 17–6 Loading Data Into a Type 2 SCD



This image displays a mapping that loads data from a source table into the `PRODUCTS` level of the 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 17–6, 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 17–6, 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.

In a mapping that loads a Type 2 SCD, if you map attributes from the source to the effective date attribute of a level, Warehouse Builder performs the following

- While loading the initial record, if the value of the source attribute is earlier than the value specified by the Default Effective Time of Initial Record property of the dimension operator (bound to the Type 2 SCD), the value from the source is used as the effective date of the record; else the value specified in the Default Effective Time of Initial Record property is used as the effective date of the record.
- During subsequent loads for the record, if the record is being versioned, the effective time of the new record is taken from the source. If no value is given for the effective time, `SYSDATE` is used. The expiration time of the closed record is set to the effective time of the new record minus the gap.

If you do not map attributes from the source to the effective date attribute of a level, Warehouse Builder performs the following:

- While loading the initial record, the value specified in the Default Effective Time of Initial Record property is used as the effective date of the record.

- During subsequent loads for the record, if a new version is being created, the effective time of the new record is taken from the source. If no value is given for the effective time, SYSDATE is used. The expiration time of the previous version is taken as the effective time of the new version minus the gap.

  For more information about the gap, see "Type 2 Gap" on page 17-16 and "Type 2 Gap Units" on page 17-16.

  Note: Mapping to the Expiration Date attribute of a level is not allowed. While loading a record, the Default Expiration Time of Open Record property is used as the expiration date. The default value of this property is NULL.

For example, you create a mapping that loads the Products Type 2 SCD. The leaf level of this Type 2 SCD, Product, is loaded from a source table. The Dimension operator has the following properties:

Default Effective Time of Initial Record: 01-jan-2000

Default Effective Time of Open Record: SYSDATE

Default Expiration Time of Open Record: 01-jan-2099

The effective date attribute of the Product level is mapped from the source attribute EFF_DATE. Consider a source Product level record with the value of EFF_DATE as 21-mar-2007.

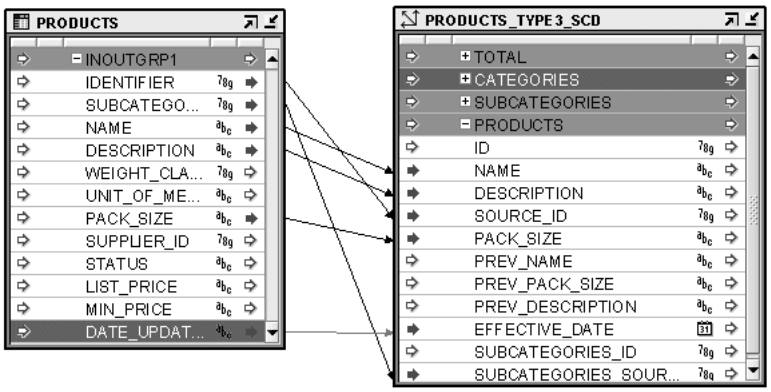
When the initial Product level record is loaded, the effective date stored in this record is 01-jan-2000 and the expiration date is 01-jan-2099. When this Product level record is versioned during a subsequent load on 21-mar-2007, the value of the source attribute overrides the Default Effective Time of Open Record property. The effective date stored in the new Product level record is 21-mar-2007 and expiration date is set to 01-jan-2099. The initial Product level record is closed with the value of expiration date set to 21-mar-2007 minus the Type 2 Gap value.

Loading Data Into Type 3 SCDs Before loading data into a Type 3 SCD, a check is made to determine 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, the following steps are performed:

- 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 17–7 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 17–7 Loading a Type 3 SCD



This image displays a mapping that loads data from a source table into one level of 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 17–7, you cannot map an attribute to the `PREV_PACK_SIZE` and `PREV_DESCRIPTION` attributes of the `PRODUCTS` level.

External Table Operator

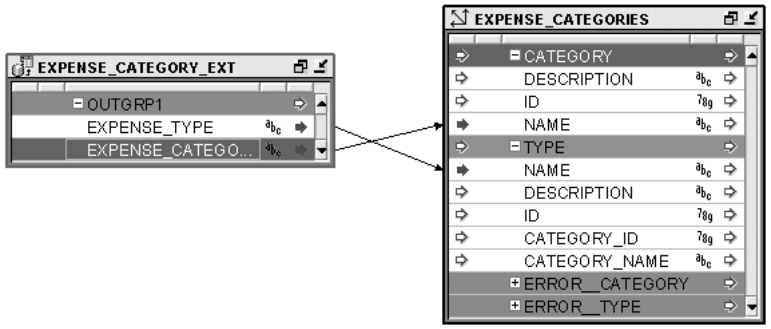
The External Table operator enables you to source data stored in external tables in the workspace. You can then load the external table data into another workspace object or perform transformations on the data. For example, you can source data stored in an external table, transform the data using mapping operators, and then load the data into a dimension or a cube.

This image displays the icon for the operator described in the surrounding text.

****

Figure 17–8 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 workspace. The data stored in this external table is used to load the dimension `EXPENSE_CATEGORIES`.

Figure 17–8 External Table Operator in a Mapping



The image displays a mapping that uses an External Table operator to load data from the external table into a dimension. The output attributes of the External Table operator are mapped to the input attributes of the dimension operator.

****

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 box.

2. Use the Add External Table dialog box to create or select an external table. For more information about these options, see Adding Operators that Bind to Workspace Objects on page 16-8.

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.

This image displays the icon for the operator described in the surrounding text.

****

You can bind and synchronize an Expand Object operator with a workspace object type. To avoid generation errors in the mapping, ensure that you deploy the workspace object type before you deploy the mapping.

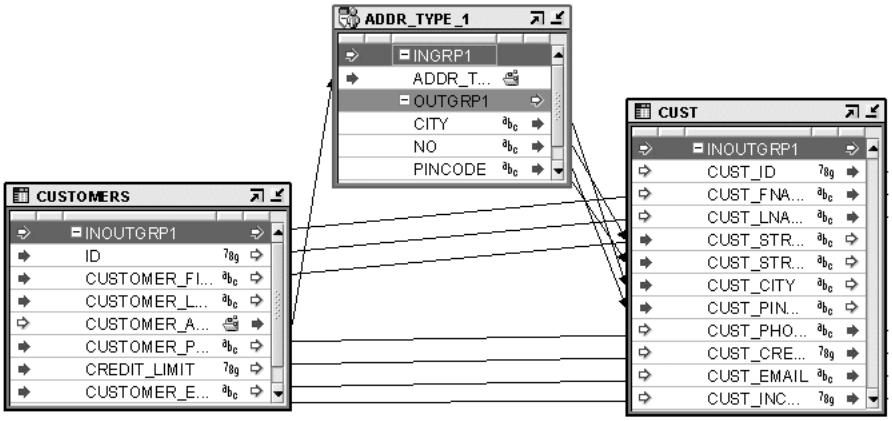
The Expand Object operator has one input group and one output group. The input group represents the object type that you want to expand in order to obtain its individual attributes. When you bind an Expand Object operator to a workspace object, the output group of the operator contains the individual 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 17–9 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 17–9 Expand Operator in a Mapping



The description of this image is contained in the surrounding text.

****

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 box to create or select an object. For more information about these options, see Adding Operators that Bind to Workspace Objects on page 16-8.
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.

This image displays the icon for the operator described in the surrounding text.

****

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, a PL/SQL package is created. 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.

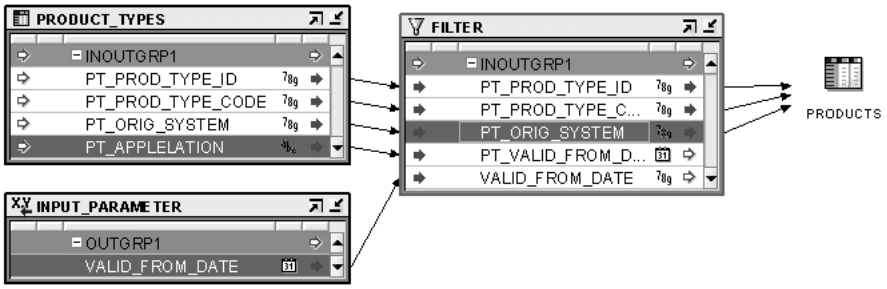
To define a Mapping Input Parameter operator in a mapping:

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 17–10.

Figure 17–10 Mapping Editor Showing A Mapping Input Parameter



This image displays a mapping that uses a Mapping Input Parameter operator to provide a value to the `VALID_FROM_DATE` attribute of the Filter operator.

****

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.

This image displays the icon for the operator described in the surrounding text.

****

A Mapping Output Parameter operator is not valid for a SQL*Loader mapping. When you generate mapping, a PL/SQL package is created. 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:

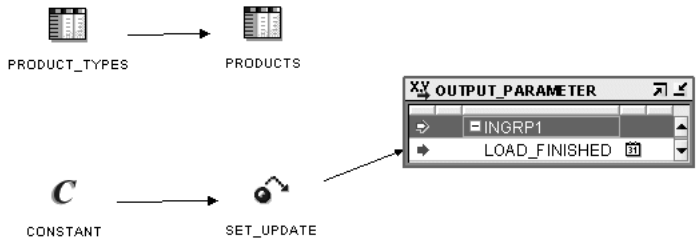
1. Drag and drop a Mapping Output Parameter operator onto the Mapping Editor canvas.

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 17–11 displays an example of a Mapping Output Parameter operator used in a mapping.

Figure 17–11 Mapping Editor Showing An Output Parameter Operator



This image displays a mapping that contains a Mapping Output Parameter operator. This operator is used to store the time at which the load completed. You can pass the value from this operator to applications external to Warehouse Builder.

****

Materialized View Operator

The Materialized View operator enables you to source data from or load data into a materialized view stored in the workspace.

This image displays the icon for the operator described in the surrounding text.

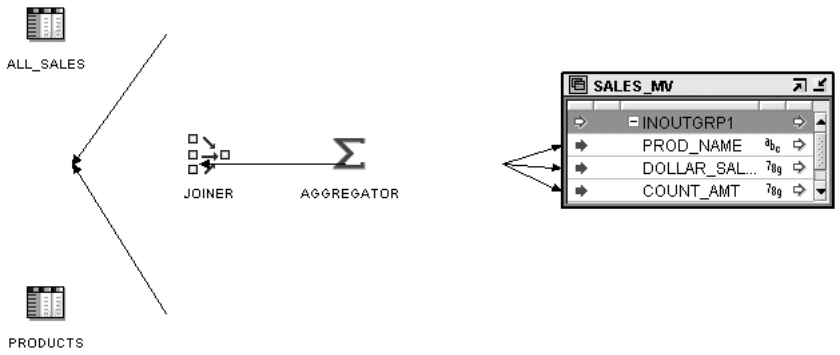
****

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 workspace table. The workspace 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 17–12 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 17–12 Mapping that Contains a Materialized View Operator



This description of this image is provided in the surrounding text.

****

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 box.

2. Use the Add Materialized View dialog box to create or select a materialized view. For more information about these options, see "Adding Operators that Bind to Workspace Objects" on page 16-8.

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.

This image displays the icon for the operator described in the surrounding text.

****

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.

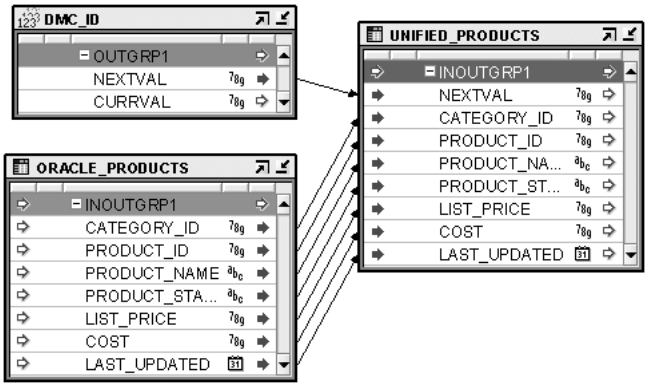
You can bind and synchronize Sequences to a workspace sequence in one of the modules. The workspace sequence must be generated and deployed before the

mapping containing the Sequence is deployed to avoid errors in the generated code package. See "Adding Operators that Bind to Workspace Objects" on page 16-8 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 22-31 for more information about configuring mode settings.

Figure 17–13 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 17–13 Mapping Sequence Operator in a Mapping



The image displays a mapping that uses a Mapping Sequence operator. The NEXTVAL attribute of the Sequence operator is mapped to the target table attribute that requires sequentially generated values.

****

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 box.

2. Use the Add Sequence dialog box to create or select a sequence. For more information about these options, see "Adding Operators that Bind to Workspace Objects" on page 16-8.

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 workspace.

The image displays the icon for the operator described in the surrounding text.

****

You can bind and synchronize a Table operator to a workspace table. To avoid errors in the generated code package, the workspace table must be deployed before the mapping that contains the Table operator is generated.

Figure 17–13 displays a mapping that uses Table operators as both source and target. Figure 17–3 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 box.

2. Use the Add Table dialog box to create or select a table. For more information about these options, see "Adding Operators that Bind to Workspace Objects" on page 16-8.

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.

Merge Optimization for Table Operators

Beginning with Warehouse Builder 10.2.0.3, you can enable the Merge Optimization property for table operators. When set to True, this property optimizes the invocation or execution of expressions and transformations in the MERGE statement.

For example, consider a mapping in which the target table contains a column that is part of the update operation only and is mapped to a transformation. In previous releases, Warehouse Builder would execute the transformation for all rows, including rows that did not require transformation. Beginning in this release, if Merge Optimization is enabled, then Warehouse Builder calls the transformation only in the update part of the MERGE statement.

Creating Temporary Tables While Performing ETL

Warehouse Builder enables you to use temporary tables while extracting and loading data. Temporary tables are useful in scenarios where you need to extract data from remote sources into multiple targets.

Temporary staging tables are used in the dimension loading logic that is automatically generated by the Dimension operator submapping expansion. This prevents problems that would be caused by doing lookups on the target table.

The properties described below enable you to create temporary tables while performing ETL.

Is Temp Stage Table

When you set the Is Temp Stage Table property to True, any existing bindings for the Table operator are ignored. A temporary staging table is deployed with the mapping and all loading and extracting operations from this Table operator are performed from the staging table.

The name of the deployed table in the database is based on the operator name, with a unique identifier appended to the name to prevent name clashes. The table is automatically dropped when the map is dropped or redeployed. Before each execution of the mapping, the table is automatically truncated.

When you set this property to its default value of False, it has no effect.

Extra DDL Clauses

Use this property to add additional clauses to the DDL statement that is used to create the table. For example, use the following `TABLESPACE` clause to allocate storage for

the temporary table in the `MY_TBLSPC` tablespace, instead of in the default tablespace: `TABLESPACE MY_TBLSPC`.

If you do not provide a value for the Extra DDL Clauses property, this property has no effect on the table creation.

Temp Stage Table ID

Use the Temp Stage Table ID property to specify the internal identifier used for the temporary staging table by the code generator. If any other temporary staging table in the mapping has the same value for the Temp Stage Table ID property, then the same deployed temporary staging table will be bound to both operators. This enables multiple usages of the same temporary staging table in the same mapping.

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.

The image displays the icon for the operator described in the surrounding text.

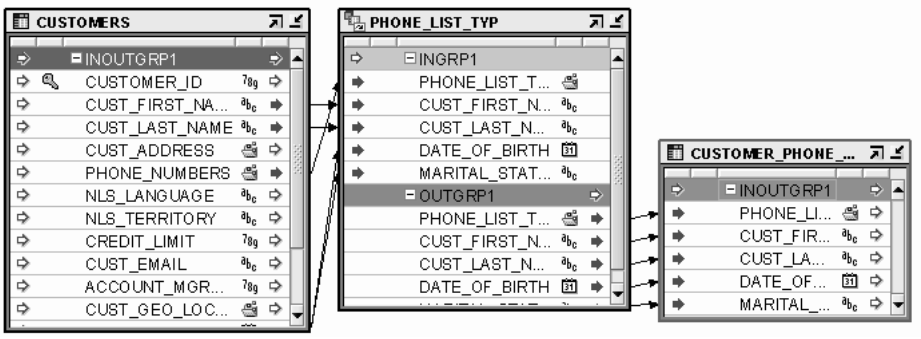
****

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 17–14 Varray Iterator Operator in a Mapping



This image displays a mapping that uses a Varray Iterator operator. This shows how Varray Iterator operator can be used to denormalize a table type.

****

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 box.

2. From the **Add Varray Iterator** dialog box, select 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 select 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 box. 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 workspace.

The image displays the icon for the operator described in the surrounding text.

****

You can bind and synchronize a View operator to a workspace view. The workspace 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 box.
2. Use the Add View dialog box to create or select a view. For more information about these options, see "Adding Operators that Bind to Workspace Objects" on page 16-8.
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. Use database links to access these targets. The database links are created using the locations. SAP targets are not supported, in that it is nor possible to 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, default workarounds are used 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 used 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 Oracle 9*i* and to UPDATE in Oracle 10*g*, a MERGE statement is generated 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, code is generated 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 code is generated for a mapping, this parameter is set to TRUE if the mapping contains a remote or non-Oracle target.

- For a multi-table insert to a remote or non-Oracle target, an INSERT statement is generated per table instead of a multi-table insert statement.

- While performing bulk inserts on a remote or non-Oracle Database, bulk processing code is not generated. Instead, code that processes one row at a time is generated. 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 17-2.

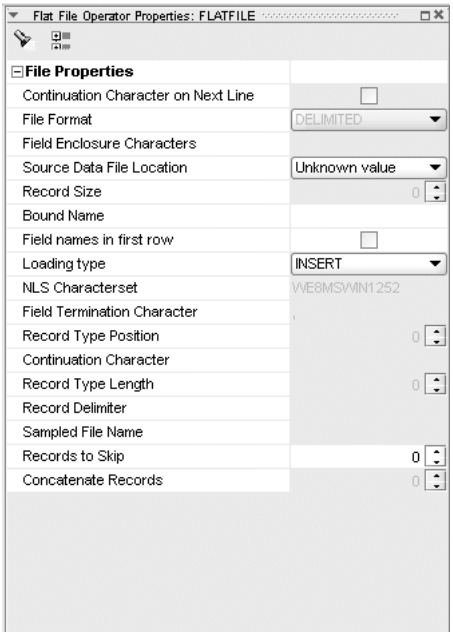
Using Flat File Source and Target Operators

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 17–15.

Figure 17–15 Properties Window for Flat File Operator



This image displays the Properties panel of a Flat File operator.

****

Loading Types for Flat Files

Select a loading type from the list:

- **Insert:** Creates a new target file. In the case that there is an existing target file, then the newly created file replaces the previous file.
- **Update:** Creates a new target file if one does not already exist. In the case that there is an existing target file, then that file is appended.
- **None:** No operation is performed on the data in the target file. This setting is useful for testing purposes. All transformations and extractions are run without affecting 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.

This image displays the icon for the operator described in the surrounding text.

****

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" 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 18-12
- Constant Operator on page 17-8
- Data Generator Operator on page 17-12
- Sequence Operator on page 17-27
- Expression Operator on page 18-11
- Transformation Operator on page 18-41
- 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 22-20. 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 16-2.

18

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 start the Expression Builder as an aide to writing SQL expressions.

This chapter includes the following topics:

- List of Data Flow Operators
- Operator Wizards
- The Expression Builder

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, wizards 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 any of the wizard pages.

The following operators have wizards to assist you:

- Key Lookup Operator
- Match-Merge Operator, see "Using the Match-Merge Operator to Eliminate Duplicate Source Records"
- Name and Address Operator, see "Using the Name and Address Operator to Cleanse Source Data"
- 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 start 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 a tab format rather than wizard pages.

Whether you are using an operator wizard or the Operator Editor, you must 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 grays 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 right arrow button between the two panels to move your selections to the Mapped Attributes panel.

   You can 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 data flow connection 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.

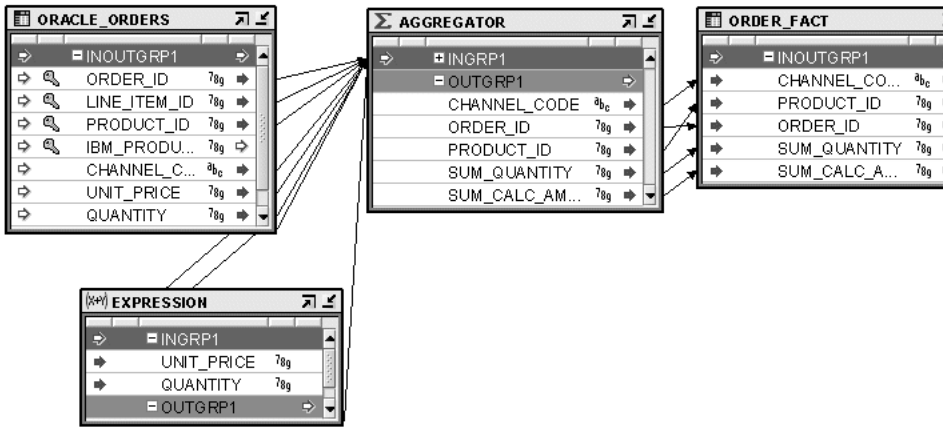
You can open the Expression Builder from the Attribute Properties panel of the Mapping Editor in the operators such as expressions, data generators, splitters, and constants.

To open the Expression Builder:

1. From the Properties panel of the operator or an attribute, click the Ellipsis button in the Expression field.

   The Expression Builder displays as shown in Figure 18–1.

Figure 18–1 Expression Builder Interface



This image displays the Expression Builder interface. The left panel of the Expression Builder displays the attributes. The right panel displays the created expression. The buttons for arithmetic and logical operators are displayed below the Expression panel.

****

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.
- **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 list of available SQL clauses that are appropriate for the active expression type.

  Beginning in Oracle 9*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, it is promoted 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, the function

  ```
  DECODE (T1.A, 1, 'ABC', 2, 'DEF', 3, 'GHI', 'JKL')
  ```

  is converted 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 workspace 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 workspace, 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

This image displays the icon for the operator described in the surrounding text.

****

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.

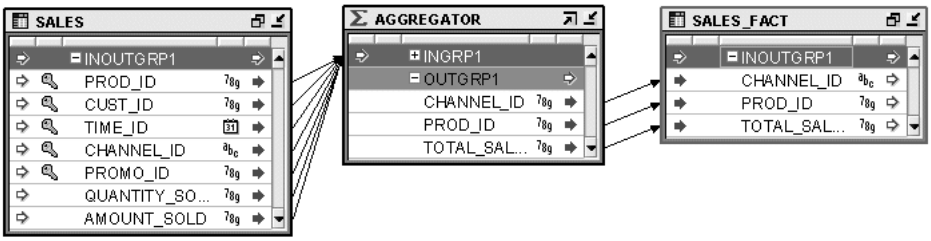
The Aggregator operator has one input group and one output group. For the output group, define a GROUP BY clause that specifies the attributes over which the aggregation is performed. You can optionally specify a HAVING clause that restricts the aggregated data. Each attribute in the output group has 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 18–2 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 18–2 Aggregator Operator in a Mapping



This image displays a mapping that uses the Aggregator operator. The data from the source table SALES is aggregated using the Aggregator operator. The output of this Aggregator operator is mapped to the target table.

****

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 18–2, 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 18-8.

   In the example displayed in Figure 18–2, 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 18-6 and "Having Clause" on page 18-7.
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 box is displayed.

3. Move the attributes 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:

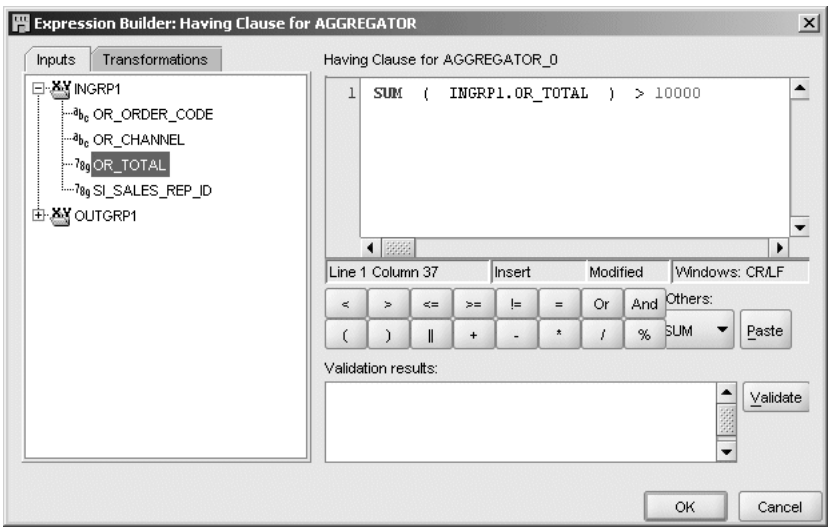
1. Select the Aggregator operator on the mapping canvas.

   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 box for the Having Clause displays as shown in Figure 18–3.

Figure 18–3 Having Clause Dialog Box



The image displays the Expression Builder dialog box for the Having Clause. The aggregation function is specified as `SUM` and the column selected for the aggregation is `QUANTITY`.

****

3. Create an expression for the Having Clause of the Aggregator operator.

   For example, Figure 18–3 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.

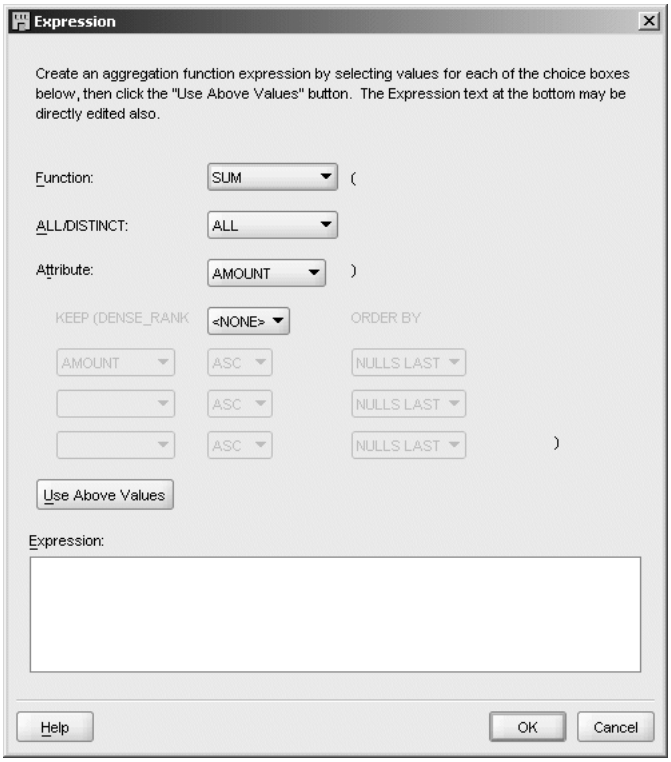
To define expressions for output attributes:

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.

Figure 18–4 Expression Dialog Box



This image displays the Expression dialog box. It contains a list to select the attribute and the aggregate function used for the attribute.

****

3. Select an aggregate function from the **Function** list.

   The aggregate functions you can select 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 18–2, 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** list.

   In the example displayed in Figure 18–2, you select the attribute `amount_sold` from the list.

6. Click **OK.**

Anydata Cast Operator

Anydata Cast operator allows 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.

This image displays the icon for the operator described in the surrounding text.

****

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.

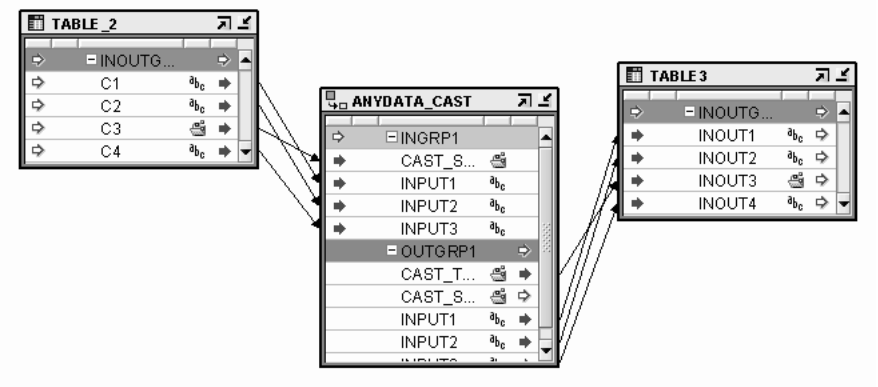
If you change the data type to which you are going to cast the Anydata attribute, then you must:

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 18–5 displays a mapping that uses an Anydata Cast operator.

Figure 18–5 Anydata Cast in a Mapping



The image displays a mapping that contains an Anydata Cast operator.

****

To define a Anydata Cast operator in a mapping:

1. Drop an Anydata Cast operator onto the Mapping Editor canvas.

   The AnyData Cast dialog box is displayed. The tree inside the dialog box 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.

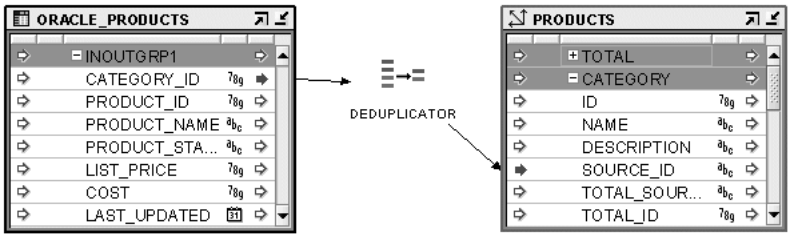
This image displays the icon for the operator described in the surrounding text.

****

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 18–6 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 18–6 Deduplicator in a Mapping



The image displays a mapping that contains a deduplicator operator. The column from the source table are mapped to the columns in the target table through the deduplicator operator.

****

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.

This image displays the icon for the operator described in the surrounding text.

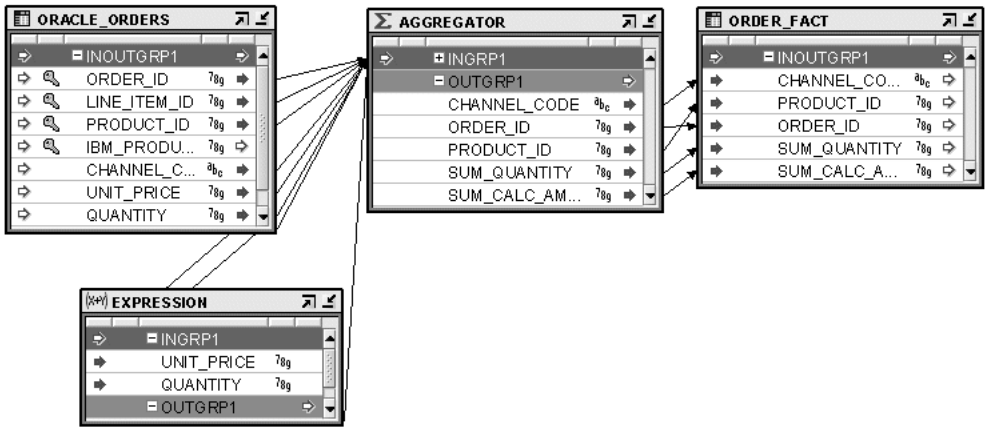
****

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 18–7 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 18–7 Expression Operator in a Mapping



The image displays a mapping that contains an Expression operator. Two input attributes are passed to the Expression operator. The Expression operator uses an expression to compute the total amount of a sale.

****

Do not use the expression operator to write aggregation functions. Use the Aggregator operator. See "Aggregator Operator" on page 18-5.

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.

This image displays the icon for the operator described in the surrounding text.

****

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

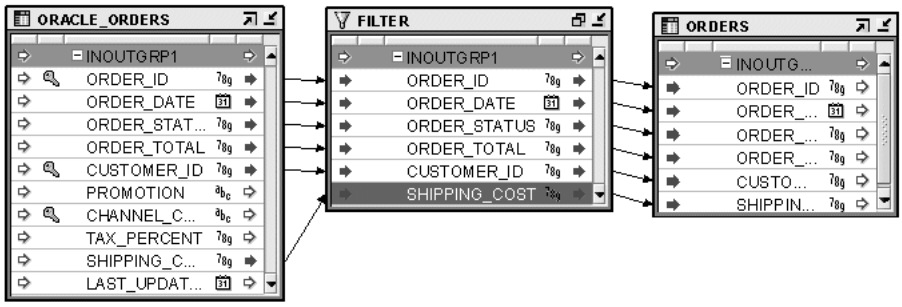
target by placing a WHERE clause in the code represented by the mapping. You specify the filter condition using the Expression Builder. The filter condition can be based on all supported data types and can contain constants.

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 be directly inserted in the target.

For a mapping that contains a Filter operator, code that displays the filter condition expression as a WHERE clause for set-based view mode is generated. 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 18–8 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 18–8 Filter in a Mapping



The image displays a mapping that contains a filter operator. Data from the source table is passed to the filter operator. The output of the filter operator is passed to the target table.

****

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 to the right of the Filter Condition property.

   Warehouse Builder displays the Expression Builder dialog box 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.

Adding Self Joins in a Mapping

The Mapping Editor enables you to recursively join a table, view, or other source data operators onto itself.

Also known as *tree walking*, recursively joining a table back onto itself enables you to retrieve records in a hierarchy. For example, consider a table that contains employee data including the manager for each employee. Using tree walking, you could determine the hierarchy of employees reporting up to a given manager.

To perform tree walking:

1. Create a mapping and add the desired source data operator such as a table, view, or a materialized view operator, which contains the hierarchal definition.
2. Connect that source data operator to a Filter operator.
3. In the filter operator, define the filter condition with CONNECT BY as the first two words. Make sure that you include only the connect by logic in the filter operator. That is, do not include any AND or OR logic in the filter.

Joiner Operator

The Joiner operator joins multiple row sets from different sources with different cardinalities, and produces a single output row set. You can use the Joiner operator to create inner joins, outer joins, equijoins, and non- equijoins. You can also create self joins by using a Filter operator as described in Adding Self Joins in a Mapping on page 18-14.

This image displays the icon for the operator described in the surrounding text.

****

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. When run on Oracle 9*i*, full outer joins are supported. For more information about joins, see *Oracle Database SQL Language 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, the Cartesian product of the two tables is returned and each row of one table is combined 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 pointing 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

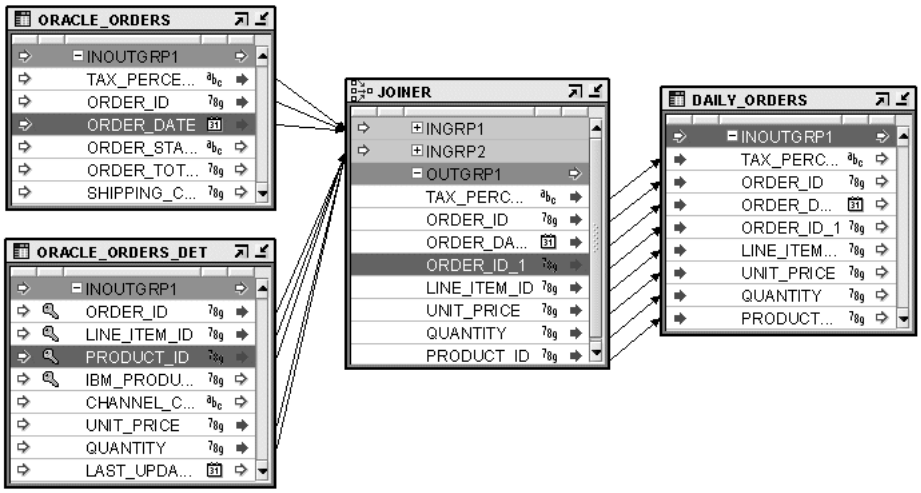
```
T2.A = T2.A AND T1.B = T2.B AND T2.C = T1.C
```

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, ABAP code can be generated by interpreting the PL/SQL join condition in the ABAP context. ABAP can only join over defined foreign key relationships.

Figure 18–9 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 18–9 Joiner in a Mapping



The image displays a mapping that contains a joiner operator. In the mapping, the two source tables are joined using the joiner operator. The output of the joiner operator is passed to the target table.

****

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 box 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 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 the join order, while 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, the statement is interpreted 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 Oracle 9*i* or a later version, the 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, any condition with the double (+) sign is translated 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 Oracle 9*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 Oracle 9*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 18-15 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 14-9.

This image displays the icon for the operator described in the surrounding text.

****

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.

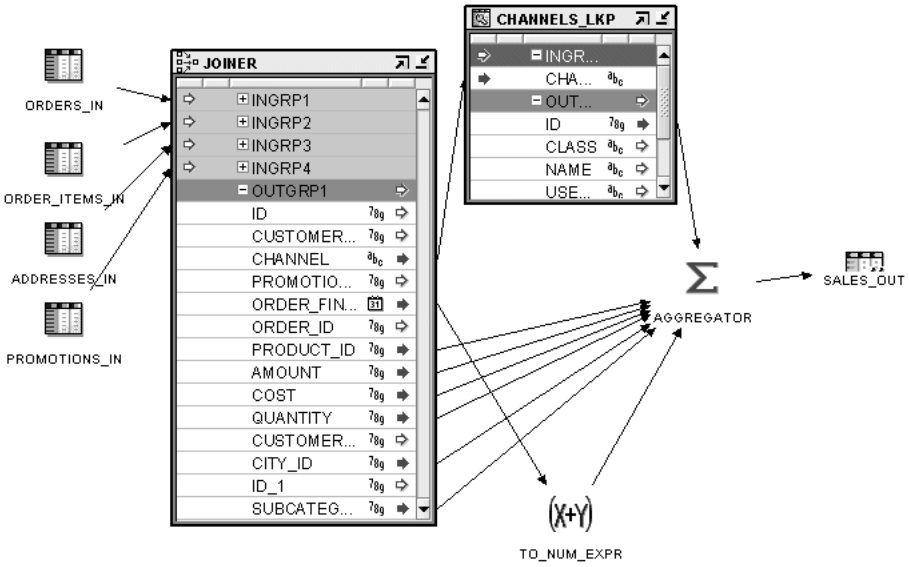
The output of the Key Lookup operator corresponds to the columns in the lookup object. To ensure that only a single lookup row is found for each key input row, use keys in your match condition.

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 workspace object to which it is bound. But you cannot synchronize the workspace object with the Key Lookup operator. For more information about synchronizing operators, see "Synchronizing Operators based on Workspace Objects" on page 16-23.

Figure 18–10 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 18–10 Key Lookup in a Mapping



The description of this image is provided in the surrounding text.

****

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 displays 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.

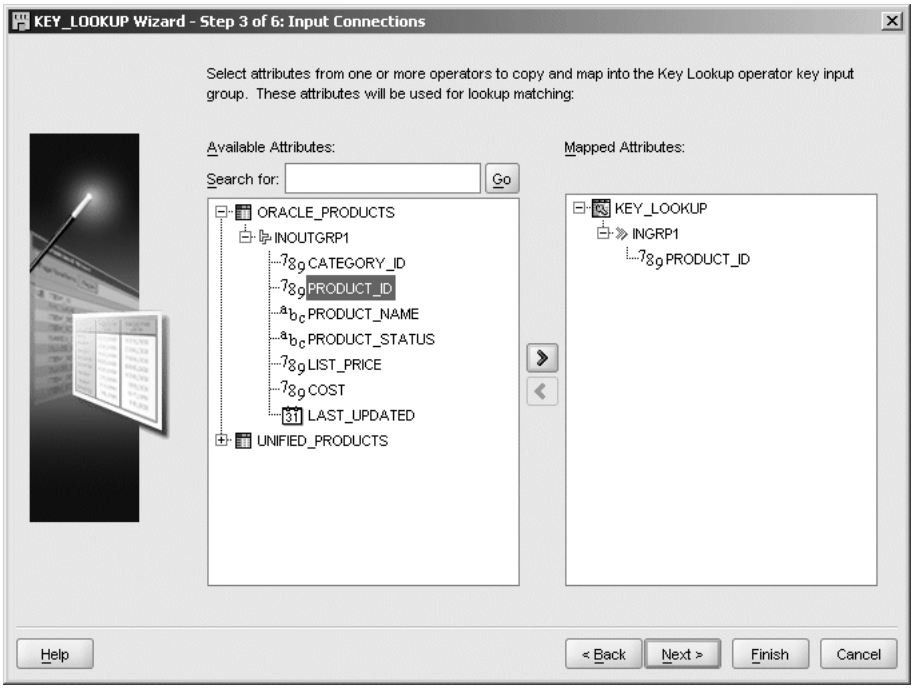
You can rename and add descriptions to the input and output groups. Since each Key Lookup operator must have exactly one input and one output group, the wizard prevents you from adding or removing groups or changing group direction.

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 18–11 shows an attribute from the table `ORACLE_PRODUCTS` table selected as input for the key lookup operator.

Figure 18–11 Input Connections Page of the Key Lookup Operator



The image displays the Input Connections page of the Key Lookup operator. The left panel on this page displays the operators in the mapping. The right panel displays the list of selected attributes.

****

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.

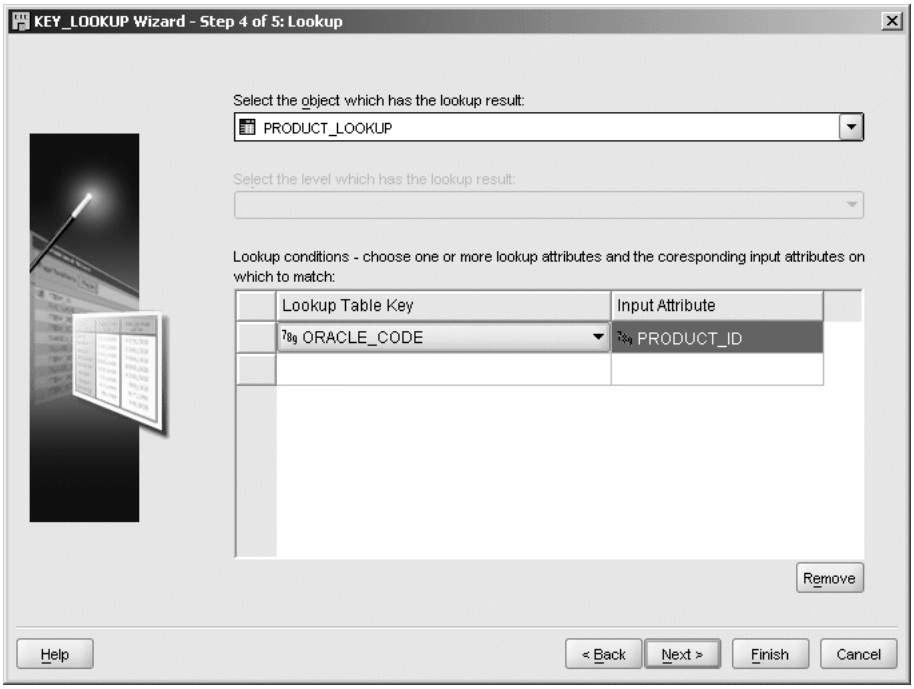
2. Use the right arrow button in the middle of the page to move your selections to the right side of the wizard page.

   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 data flow connection 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 18–12 Lookup Page of the Key Lookup Operator



This image displays the Lookup page of the Key Lookup wizard. Select the data object that contains the lookup result and then specify the lookup condition.

****

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** list to select the object on which you want to perform the lookup. This 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** 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** list displays the unique constraints in the lookup result. The **Input Attribute** list displays the attributes you selected on the Input Connections page. Use these 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 select 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** 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** 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.

This image displays the icon for the operator described in the surrounding text.

****

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 18–13, contains data from a flat file. There is a row for each sales representative and separate columns for each month. For more information about external tables, see "Using External Tables" on page 8-25.

Figure 18–13 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				

This image displays sample data from the flat file that is stored in the external table SALES_DAT.

****

Table 18–1 shows a sample of the data after a pivot operation is performed. 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 18–1 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 18–14.

Figure 18–14 Pivot Operator in a Mapping

This description of this image is included in the surrounding text.

****

In this mapping, the data is read from the external table once, pivoted, aggregated, and written 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 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, a single input attribute is transformed 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 18–2 shows the data from the first row from SALES_DAT after the data is pivoted with 'MONTH' as the row indicator.

Table 18–2 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 displays 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.

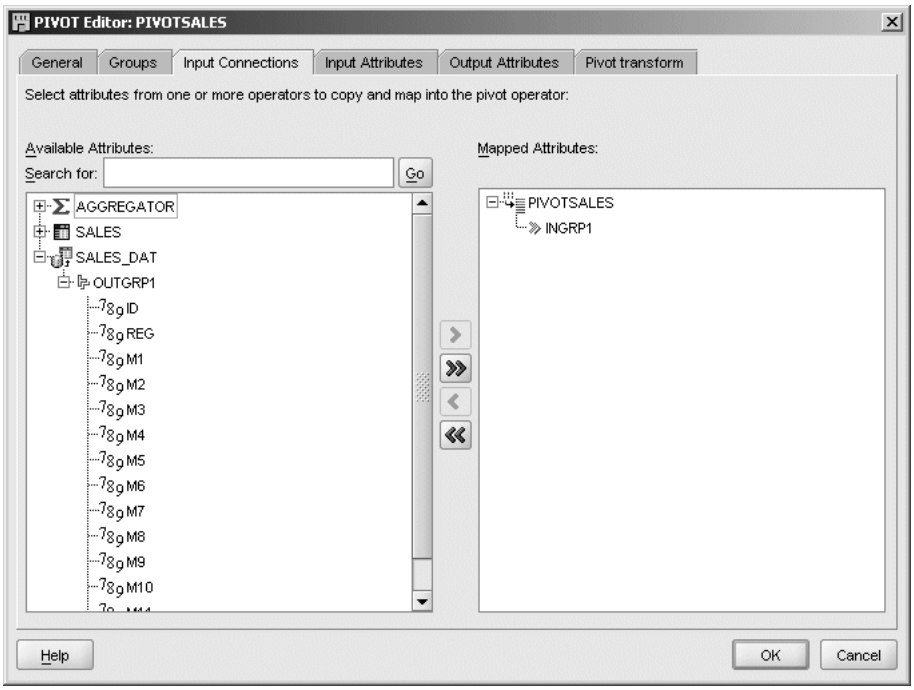
In a pivot operator, the input group represents the data from the source that is contained across multiple attributes. The output group represents that data transformed into rows.

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 18–15 Pivot Operator Input Connections Tab



This image displays the Input Connections page of the Pivot operator. The page contains the Available Attributes panel and the Mapped Attributes panel.

****

To complete the Input Connections page for a Pivot operator:

1. Select complete groups or individual attributes from the left panel.

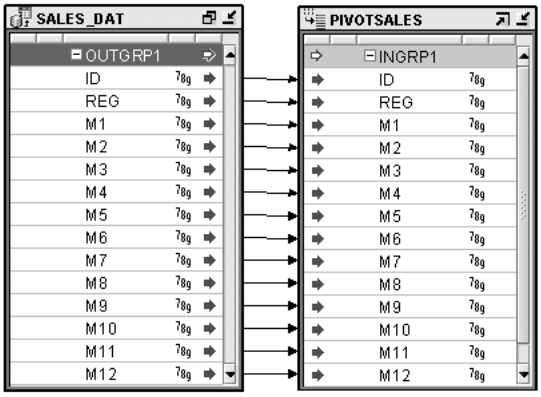
   To search for a specific attribute or group by name, type the text in **Search for** and select **Go.** To find the next match, select **Go** again.

   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 data flow connection between the source operator and the pivot operator.

Figure 18–16 Attributes Copied and Mapped into Pivot In Group



This image shows a group from the external table SALES_DAT that has been copied and mapped to the PIVOTSALES 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 Input Attributes page:

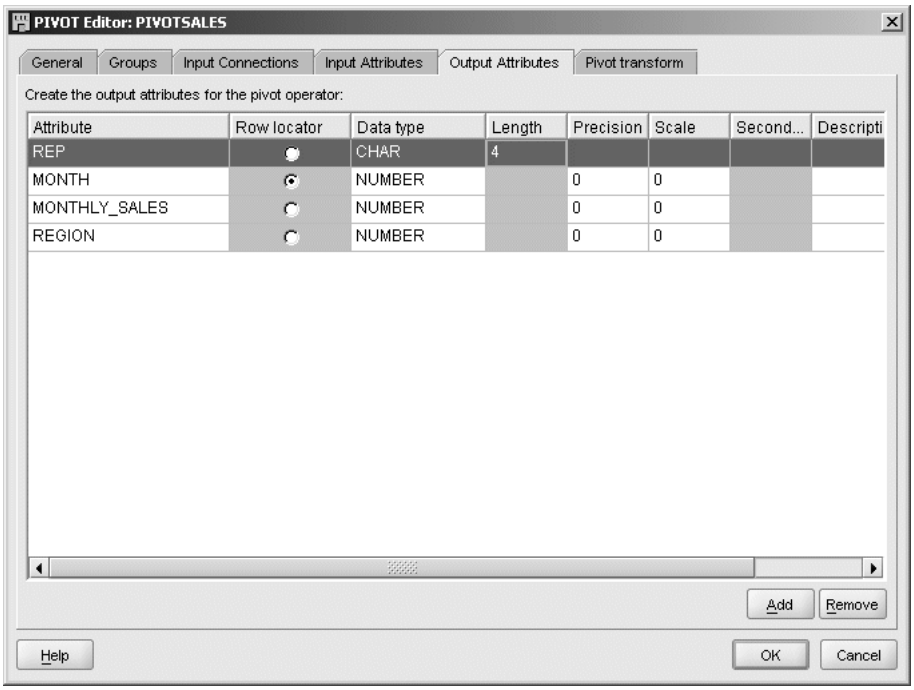
- **Add attributes:** Use the Add button to add input attributes.
- **Change attribute properties:** You can change the attribute name, data type, length, precision, and scale.
- **Add an optional description:** Type a description for the input attributes.
- **Designate attribute keys:** As an option, 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, those input attributes are displayed as output attributes that you cannot edit or delete.

Figure 18–17 displays the output attributes with MONTH selected as the row locator.

Figure 18–17 Pivot Output Attributes Tab



This image displays the Output Attributes page of the Pivot operator. This page displays the following properties of the output attributes: row locator, data type, length, precision, scale, and description. Use the Row Locator property to indicate that an attribute is the row locator.

****

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.

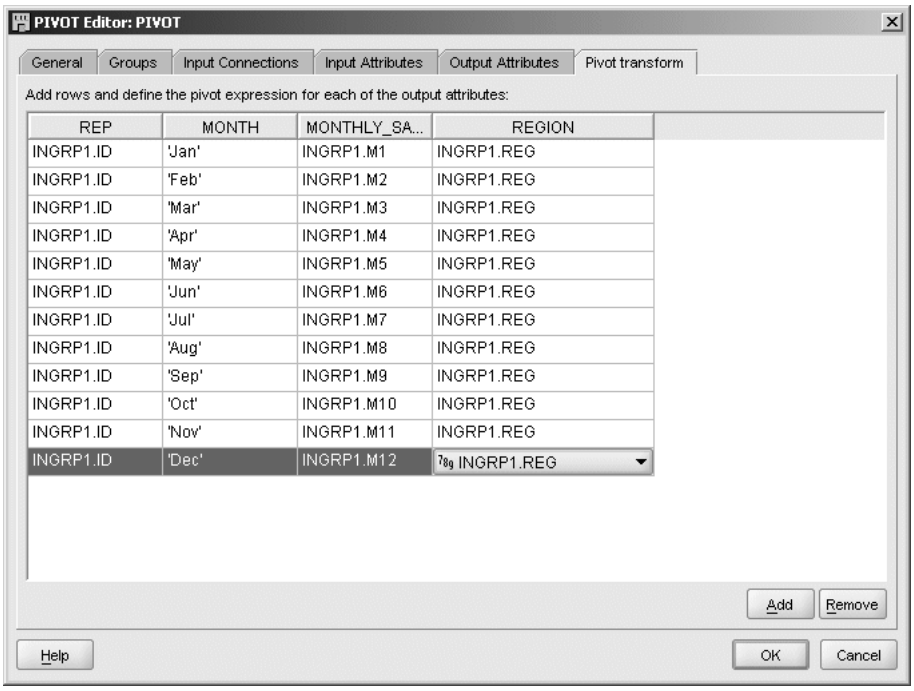
Pivot Transform

Use the Pivot Transform page to write expressions for each output attribute.

By default, two rows are displayed. 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 18–18 shows the Pivot Transform tab with the pivot expressions defined for a source with an attribute for each month.

Figure 18–18 Pivot Transform Tab



This image displays the Pivot Transform page of the Pivot operator.

****

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:** 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.

This image displays the icon for the operator described in the surrounding text.

****

The Post-Mapping Process operator calls a function or procedure 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 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 box 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 box.

2. Use the Add Post-Mapping Process dialog box to select or create a transformation. For more information about how to use the Add Post-Mapping Process dialog box, see "Adding Operators that Bind to Workspace Objects" on page 16-8.

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.

This image displays the icon for the operator described in the surrounding text.

****

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 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 box 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 box 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 box is displayed.

2. Use the Add Pre-Mapping Process dialog box to select or create a transformation. For more information about how to use this dialog box, see "Adding Operators that Bind to Workspace Objects" on page 16-8.

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: Runs the mapping after the process completes, regardless of the errors.

   On Success: Runs the mapping only if the process completes without errors.

   On Error: 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.

This image displays the icon for the operator described in the surrounding text.

****

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

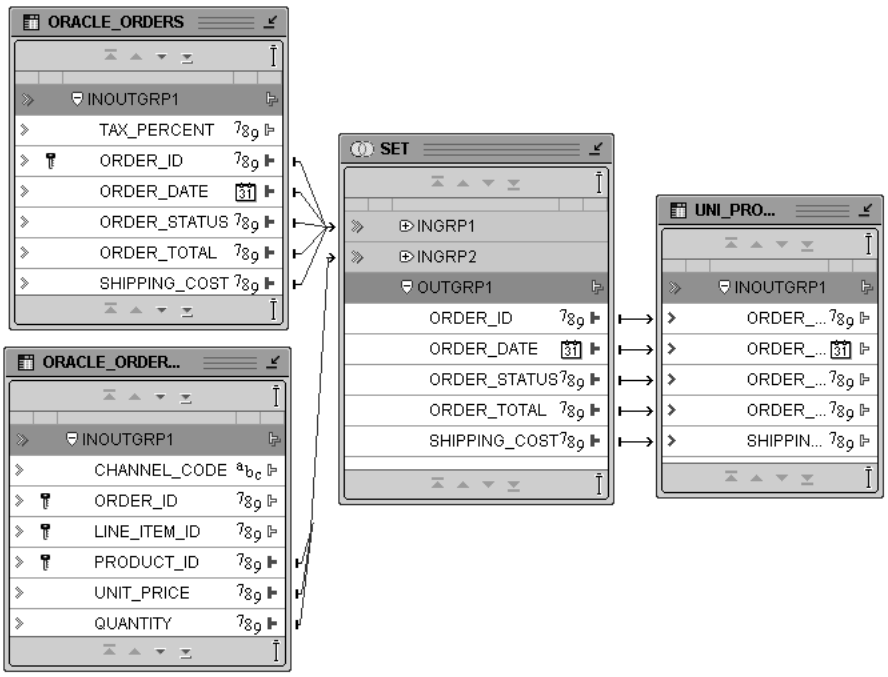
By default, the Set Operation operator contains two input groups and one output group. You can add input groups by using the operator editor. The number of attributes in the output group matches the number of attributes in the input group containing the most number of attributes.

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 18–19 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 18–19 Set Operation Operator in a Mapping



This image displays a mapping that uses a Set Operation operator. The Set Operation operator performs the Intersection set operation on the data from two source tables. The output of the Set Operation operator is mapped to the target table.

****

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 list on the **Set Operation** property and select an operation from the list.
5. Connect the Set Operation output group to a target input group.

Synchronizing the Attributes in a Set Operator

The set operator in the Mapping Editor assists you in matching attributes between two data streams. To match attributes from two data streams in a mapping, define the data streams as input groups into the set operator. On the Input Attributes tab, click **Synchronize from <Input Group Name>**. The synchronize operation rearranges and adds attributes to the target group such that the target group most closely matches the source group. The synchronize operation uses the following rules to find or create a match in the target:

1. Looks for an existing attribute in the target that matches name and datatype.
2. Looks for an existing attribute in the target whose description matches the source name, and the datatype matches source datatype.

3. If (1) and (2) fail, then a new attribute is created with the source name and datatype, and is inserted in the correct matching position. Any unmatched target group attributes are indicated by UNMATCHED in the attribute description.

   To force a target attribute to match a specified source attribute, type the source group attribute as the target attribute description.

Sorter Operator

You can produce a sorted row set using the Sorter operator.

This image displays the icon for the operator described in the surrounding text.

****

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

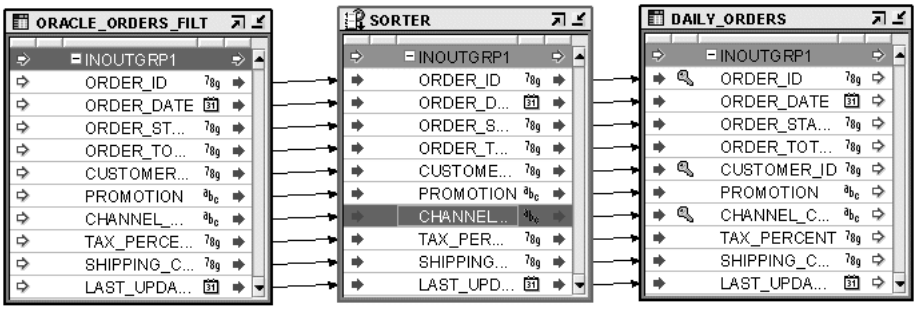
The Sorter operator contains the Order By clause. This clause is an ordered list of attributes in the input/output group to specify that sorting is performed in the same order as the ordered attribute list. You can set ascending or descending sorting for each attribute.

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 18–20 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 18–20 Sorter Operator in a Mapping



The image displays a mapping that contains a Sorter operator. The data from the source table is passed to the Sorter operator. The sorter operator sorts the rows in the desired order and then passes this output to the target table.

****

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 18–20.
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 box 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** 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.

This image displays the icon for the operator described in the surrounding text.

****

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 18-37.

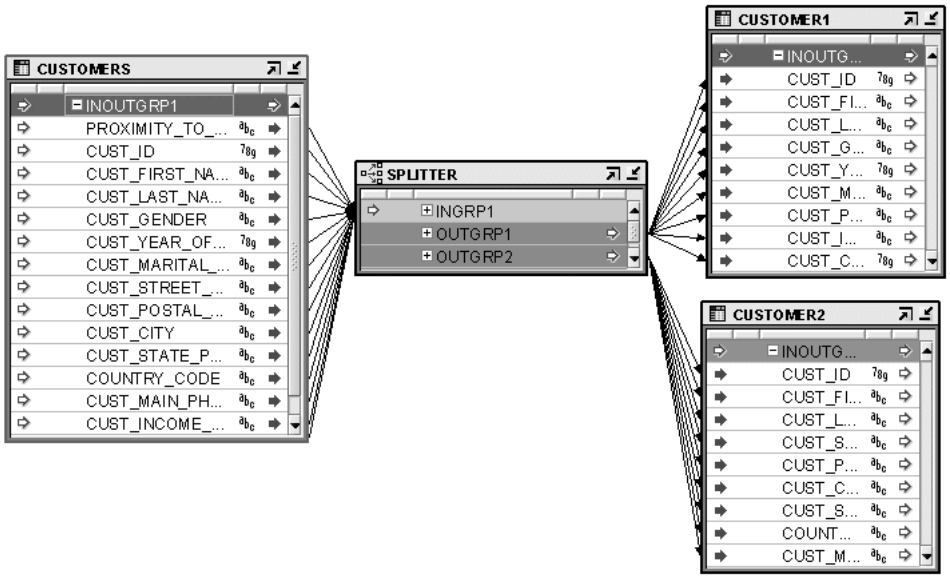
The Splitter operator contains one input group and three output groups. The output groups are OUTGRP1, OUTGRP2, and REMAINING_ROWS. You can create additional output groups, if required. You can delete the REMAINING_ROWS output group, but you cannot edit it.

In most cases, the output group REMAINING_ROWS contains all input rows that are not included in any output group. However, when the split condition contains an attribute whose value is null, the corresponding rows are not moved to the REMAINING_ROWS output group.

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 18–21 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 18–21 Splitter Operator in a Mapping



The image displays a mapping that uses a Splitter operator. The data from the source table is mapped to the Splitter operator. This operator splits the data from the source table based on the split condition and then loads the data into two different target tables.

****

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 box 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, separate insert statements for each target are generated.

To optimize a multiple target mapping, you must take additional steps to generate a single insert statement for all targets combined. In this case, a `multi_table_insert` SQL statement is generated 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 Oracle 9*i* or higher SQL.

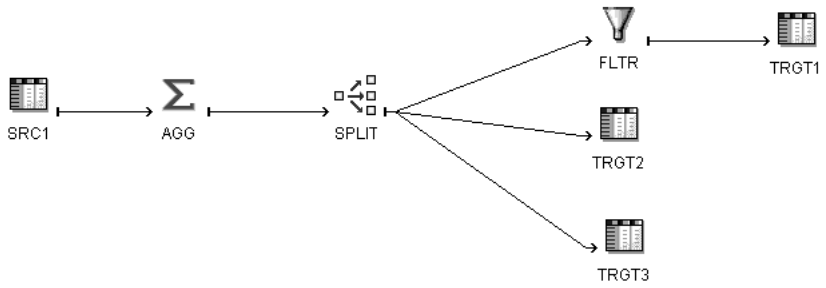
   Right-click the target module on the Project Explorer and select **Configure.** Under **Deployment System Type** and **PL/SQL Generation Mode,** select Oracle 9*i* or higher.

2. In the Mapping Editor, design a mapping with a single source, a Splitter operator, and multiple targets.

   For the mapping to be optimized, 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 18–22, but not a Joiner or Aggregator operator. These restrictions only apply if you choose to optimize the mapping.

Figure 18–22 Example Mapping with Multiple Targets



This image displays a mapping that uses a Splitter operator to move data from the source table into different 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 box 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, one total SELECT and INSERT count for all targets is returned.

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.

This image displays the icon for the operator described in the surrounding text.

****

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.

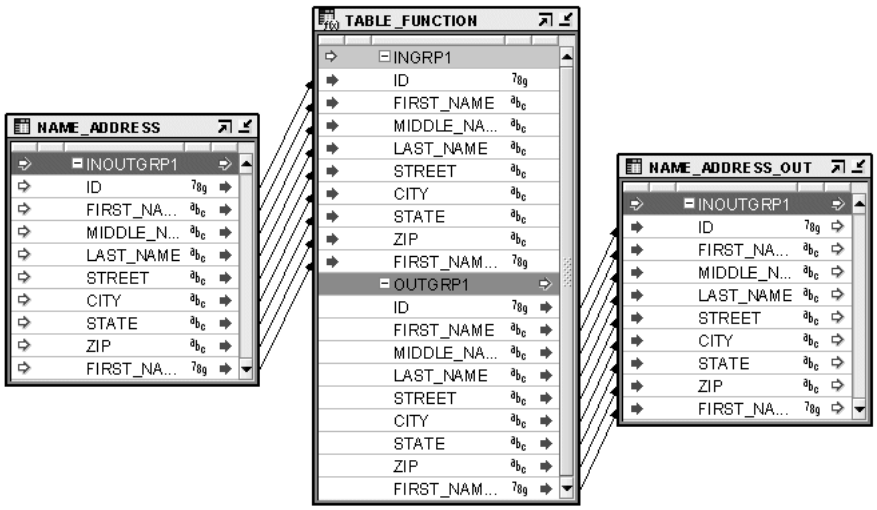
3. Right-click the Table Function operator and select **Open Details**.

   The Table Function Editor is displayed.

4. From the Groups tab, select **Add** to add an output group.

   Figure 18–23 shows a mapping that uses a Table Function operator to load data into a table.

Figure 18–23 Table Function Operator in a Mapping



The image displays a mapping that contains table function operators. The output of the table function operators is used to load the time dimension T_TIME.

****

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 supported scalar data types (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 supported scalar data types).
- Supported scalar data types.
- 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 supported scalar data type.

- **SCALAR_TYPE:** Supported scalar data types.
- **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 call the table function.

Transformation Operator

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.

This image displays the icon for the operator described in the surrounding text.

****

The Transformation operator must be bound to a function or procedure contained by one of the modules in the workspace. The inputs and outputs of the Transformation operator correspond to the input and output parameters of the bound workspace 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 an RTRIM is applied 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 box is displayed.

2. Use the Add Mapping Transformation dialog box to create a new transformation or select one or more transformations. For more information about these options, see "Adding Operators that Bind to Workspace Objects" beginning on page 16-8.
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.

This image displays the icon for the operator described in the surrounding text.

****

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 18–3 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 18–3 Data in a Crosstab Format

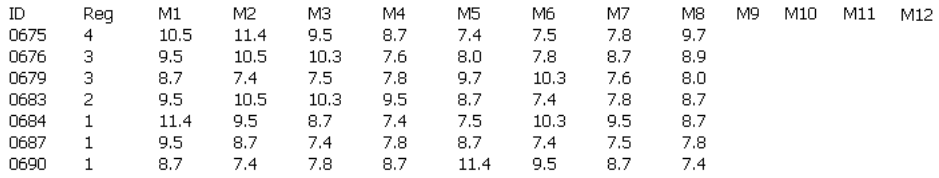
REP	MONTH	MONTHLY_SALES	REGION
0675	Jan	10.5	4
0676	Jan	9.5	3

Table 18–3 (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 18–24 depicts data from the relational table 'SALES' after unpivoting 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 18–24 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				

This image is described in the surrounding text.

****

The Row Locator

When you use the Unpivot operator, multiple input rows are transformed 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 displays 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 data flow connection 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.

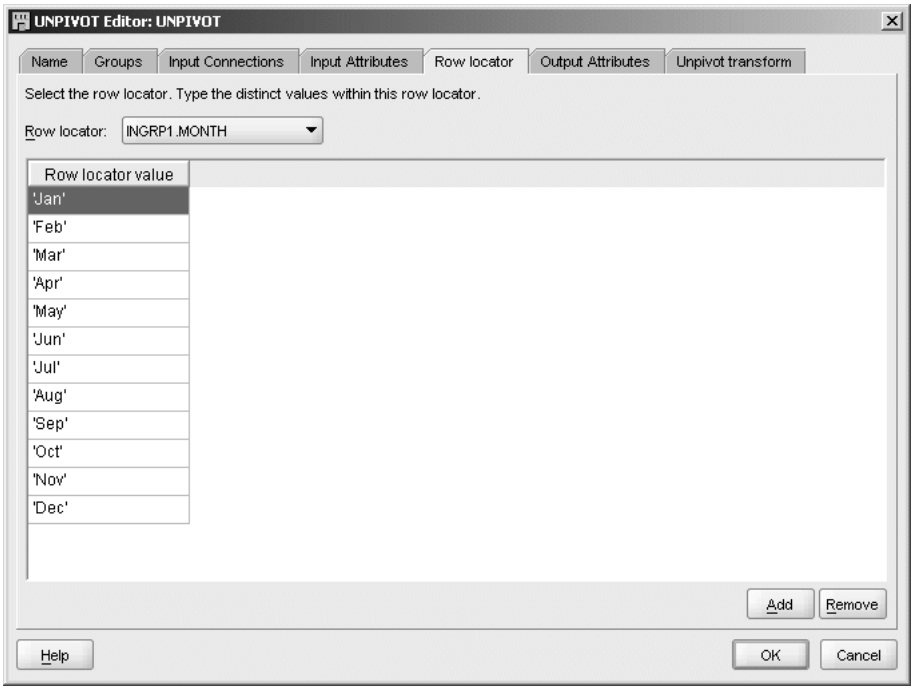
- **Add an optional description:** Type a description for the input attributes.
- **Designate key attribute(s):** You must designate one or more key attributes for unpivot operators. Use the **Key** check box to indicate the attribute(s) that uniquely identifies the input group. Input rows with the same 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 18–25 shows the attribute MONTH selected as the row locator with values such as 'Jan', 'Feb', or 'Mar'.

Figure 18–25 Unpivot Row Locator Page



This image displays the Row Locator page of the Unpivot operator. This page displays the row locator value for each row.

****

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.

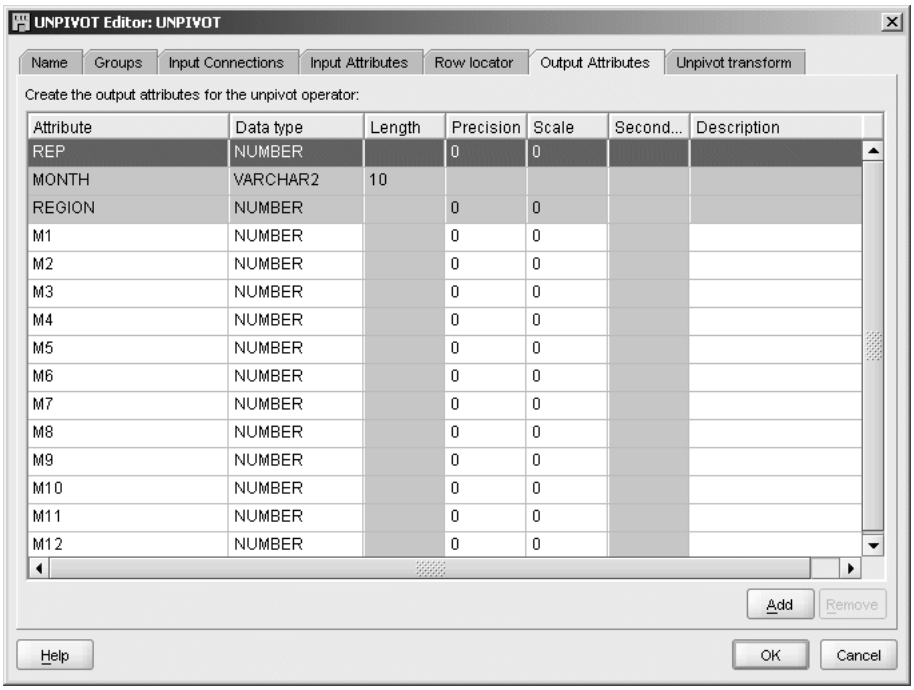
3. For each row locator value, type in the value as it appears in your source dataset.

   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 18–3 are 'Jan', 'Feb', and 'Mar.'

Output Attributes

Use the Output Attributes tab to create the output attributes for the Unpivot operator.

Figure 18–26 Unpivot Output Attributes Page



This image displays the Output Attributes page of the Unpivot operator. This page contains the following properties for each output: data type, length, precision, scale, and description.

****

If you designated any input attributes as keys on the Input Attributes tab or wizard page, those input attributes are displayed as output attributes that you cannot edit or remove.

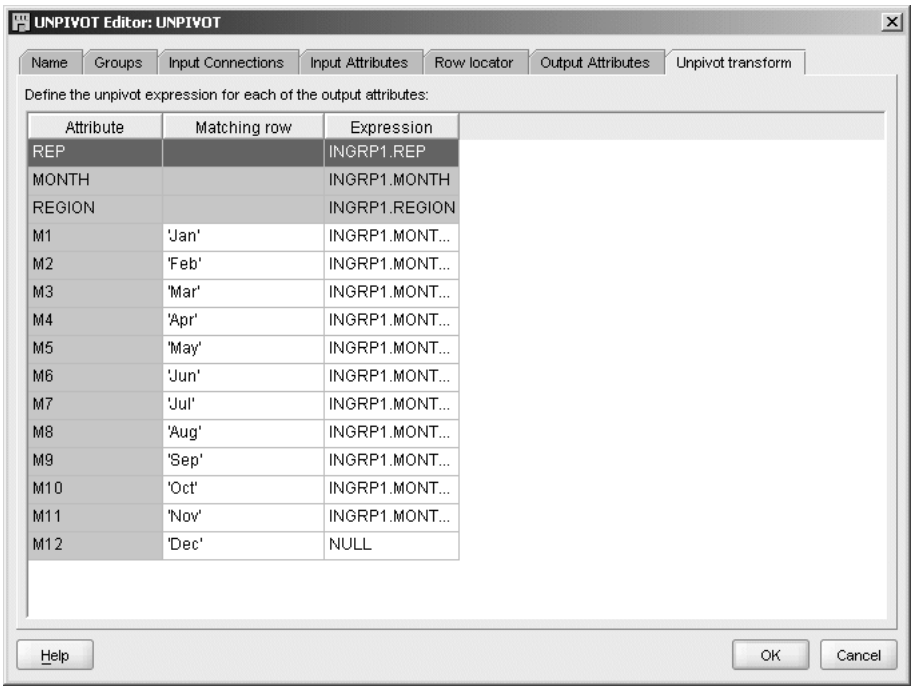
You can perform the following tasks from the Unpivot Output Attributes page:

- **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 tab to write expressions for each output attribute.

Figure 18–27 Unpivot Transform Page



This image displays the Transform page of the Unpivot operator. The page contains the Matching Row and Expression for each output attribute.

****

For attributes you designated as keys, the matching row and expression is defined 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.

19

Oracle Warehouse Builder Transformations

As you design mappings and process flows, you may want to use specialized transformations to transform data. This chapter how to create your own transformations as well as how to use the predefined transformations provided by Warehouse Builder.

This chapter contains the following topics:

- Defining Custom Transformations
- Editing Custom Transformations
- Administrative Transformations
- Character Transformations
- Control Center Transformations
- Conversion Transformations
- Date Transformations
- Number Transformations
- OLAP Transformations
- Other Transformations
- Spatial Transformations
- Streams Transformations
- XML Transformations
- Importing PL/SQL
- Example: Reusing Existing 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 module in a 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 workspace in which they are defined. For example, you create a function called `ADD_EMPL` in the Global Shared Library of the workspace `REP_OWNER`. This procedure can be used across all the projects in `REP_OWNER`.

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 workspace. Figure 19–1 displays the Global Explorer used 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 19-3 for more information about the other pages in the wizard. For more information about creating PL/SQL types, see "Defining PL/SQL Types" on page 19-5.

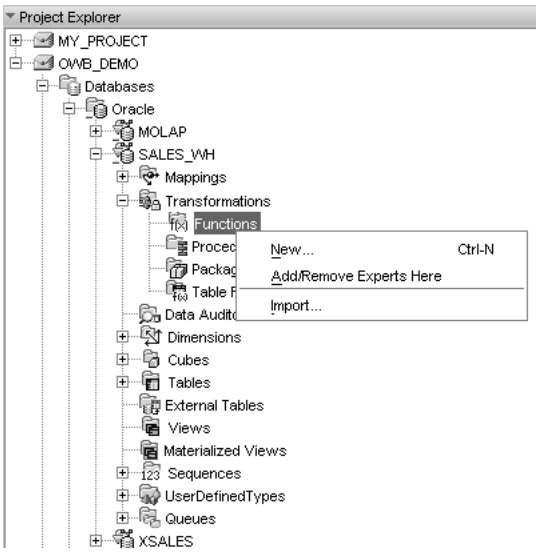
   For packages, Warehouse Builder displays the Create Transformation Library dialog box. Provide a name and an optional description for the package and click **OK**. The new package is added to the Packages node. You can subsequently create procedures, functions, or PL/SQL types that belong to this package.

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 workspace. For example, consider the workspace 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 19–1 displays the Project Explorer from which you can create custom transformations that are accessible within the project in which they are defined.

Figure 19–1 Creating Custom Transformations in an Oracle Module



This screenshot shows the Project Explorer with the Transformations node in an Oracle module expanded. There are two project nodes, MY_PROJECT and OWB_DEMO. The OWB_DEMO node is expanded and displays the following nodes, from top to bottom, under the Oracle node: MOLAP and SALES_WH. The SALES_WH node is expanded and within this node, the Transformations node is expanded to display the following, from top to bottom: Functions, Procedures, and Packages.

****

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 19-3 for information about the remaining wizard pages. For more information about creating PL/SQL types, see "Defining PL/SQL Types" on page 19-5.

   For packages, Warehouse Builder opens the Create Transformation Library dialog box. Provide a name and an optional description for the package and click **OK**. The package is added under the Packages node. You can subsequently create procedures, functions, or PL/SQL types that belong to this package.

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

- Parameters Page
- Implementation Page
- Summary Page

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 13-6.
- **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 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 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. 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 wizard pages. 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

Use the Create PL/SQL Type Wizard to create PL/SQL types. PL/SQL types must be defined within a package and 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, see *Oracle Database PL/SQL Language 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. 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 data type. 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.

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.

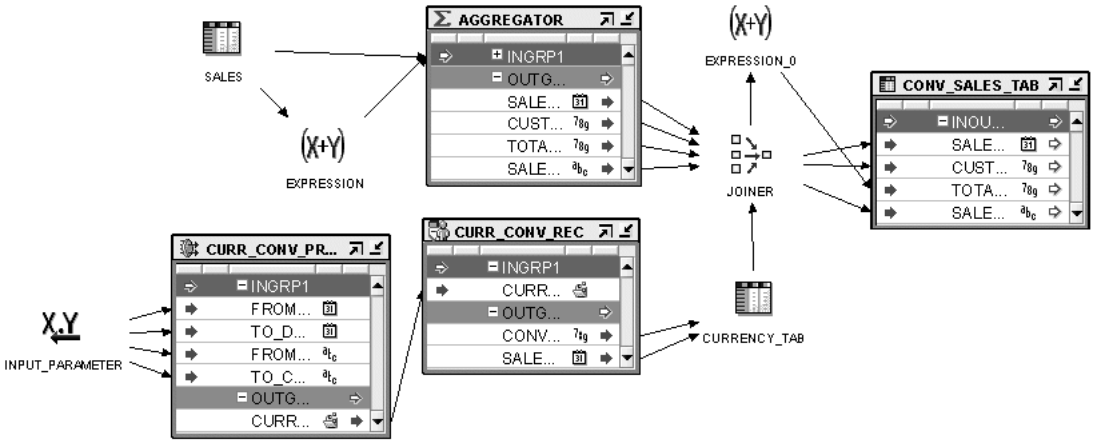
Usage Scenario for PL/SQL Types

The SALES table stores the daily sales of an organization that has offices across the world. This table contains the sale ID, sale date, customer ID, product ID, amount sold, quantity sold, and currency in which the sale was made. Management wants to analyze global sales for a specified time period using a single currency, for example the US Dollar. Thus all sales values must be converted to US Dollar. Since the currency exchange rates can change every day, the sales amounts must be computed using the exchange rate of the sale currency on the sale date.

Solution Using PL/SQL Record Types

Figure 19–2 displays the mapping that you use to obtain the sales amount in a specified currency using PL/SQL record types

Figure 19–2 PL/SQL Record Type in a Mapping



This screenshot displays the PL/SQL Record Type used in a mapping. To the left is a Mapping Input operator. To the right of this operator are the following, from top to bottom: Table operator SALES, Expression operator, and Transformation operator. To the right of the these operators are the following operators, ordered from top to bottom: Aggregator operator and Expand Object operator. To the right of these operators are the following ordered from top to bottom: Expression operator called EXPRESSION_0, Joiner operator, and Table operator bound to the CURRENCY_TAB table. To the right of these operators is a Table operator bound to the CONV_SALES_TABLE table.

The data flow between the operators in the mapping is as follows. Arrows from the Mapping Input Parameter operator to the Transformation operator represents data being input to the Transformation operator. Arrows from the SALES table and the Expression operator to the Aggregator operator represents data flowing to the Aggregator operator. There are arrows from the Transformation operator to the Construct Object operator, which in turn maps to the Table operator CURRENCY_TAB. Arrows from CURRENCY_TAB and the Aggregator operator to the Joiner operator indicate that the output of these operators is used as an input The output of Aggregator operator serves as input to the Joiner operator. There are arrows from the

joiner operator and the Expression operator EXPRESSION_0 to the CONV_SALES_ TAB table operator.

****

The mapping takes the individual sales data stored in different currencies, obtains the sales value in the specified currency, and loads this data into a target table. Use the following steps to create this mapping.

1. In the Global Explorer, create a package. In this package, create a procedure called CURR_CONV_PROC.

   This procedure obtains the currency conversion values on each date in a specified time interval from a Web site. The input parameters of this procedure are the sales currency, the currency to which the sale value needs to be converted, and the time interval for which the currency conversion is required. This data is stored in a PL/SQL record type of type CURR_CONV_REC. This record type contains two attributes: date and conversion value.

   You create the PL/SQL record type as part of the package.

2. Create a mapping that contains a Transformation operator. This operator is bound to the CURR_CONV_PROC procedure.

3. Use a Mapping Input Parameter operator to provide values for the input parameters of the Transformation operator.

   The output group of the Transformation operator is a PL/SQL record type of type CURR_CONV_REC.

4. Use an Expand Object operator to obtain the individual values stored in this record type and store these values in the table CURRENCY_TAB.

5. Use an Aggregator operator to aggregate sales details for each order.

   The SALES table is a transactional table and stores data in normalized form. To obtain the aggregate sales for each order, use an Aggregator operator to aggregate sales data.

6. Use a Joiner operator to join the aggregated sales details, which is the output of the Aggregator operator, with the data in the CURRENCY_TAB table. The sale date is used as the join condition.

7. Use the Expression operator to multiply the sales amount with the currency exchange rate to get the total sales in the required currency. Load the converted sales data into the CONV_SALES_TAB table.

Creating PL/SQL Types

You can create PL/SQL types in the Project Explorer and Global Explorer of the Design Center. For more details about creating PL/SQL types, see "Defining PL/SQL Types" on page 19-5.

Use the Create PL/SQL Types Wizard to create PL/SQL types. 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
- REF cursor type
- Nested table type

For more information about each PL/SQL type, see "About PL/SQL Types" on page 19-5.

After specifying the name and selecting the type of PL/SQL type 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 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 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 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 Custom Transformations

You can edit the definition of a custom 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 box enables you to edit function definitions. To edit a procedure definition, use the Edit Procedure dialog box.

Use the following steps to edit functions, procedures, or packages:

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.**

   For functions or procedures, the Edit Function or Edit Procedure dialog box is displayed. Use the following tabs to edit the function or procedure definition:

   - Name Tab
   - Parameters Tab
   - Implementation Tab

For packages, Warehouse Builder displays the Edit Transformation Library dialog box. 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 19-4.

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 19-4.

Editing PL/SQL Types

The Edit PL/SQL Type dialog box 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/SQL Type dialog box 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 type, 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.

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 terminate 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 terminate an application. You can use this function in a post mapping process to terminate 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_`.

Character transformations include implementations of basic Oracle Database SQL functions or procedures and custom transformations provided by Warehouse Builder.

Table 19–1 lists the character transformations that are based on Database SQL functions. The transformations are listed in a columnar table that reads down the columns from left to right to conserve space. For descriptions and examples of these transformations, refer to section "Character Functions" in the *Oracle Database SQL Language Reference*.

Table 19–1 Character Transformations Based on SQL character functions

Character Transformation Name	Character Transformation Name (Contd.)	Character Transformation Name (Contd.)
■ ASCII	■ CHR	■ CONCAT
■ INITCAP	■ INSTR	■ INSTR2
■ INSTR4	■ INSTRB	■ INSTRC
■ LENGTH	■ LENGTH2	■ LENGTH4
■ LENGTHB	■ LENGTHC	■ LOWER
■ LPAD	■ LTRIM	■ NLSSORT
■ NLS_INITCAP	■ NLS_LOWER	■ NLS_UPPER
■ REPLACE	■ REGEXP_INSTR	■ REGEXP_REPLACE
■ REGEXP_SUBSTR	■ RPAD	■ RTRIM
■ SOUNDEX	■ SUBSTR	■ SUBSTR2
■ SUBSTR4	■ SUBSTRB	■ SUBSTRC
■ TRANSLATE	■ TRIM	■ UPPER

Following is the list of custom character transformations.

- WB_LOOKUP_CHAR (number) on page 19-20
- WB_LOOKUP_CHAR (varchar2) on page 19-20
- WB_IS_SPACE on page 19-21

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 workspace. 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

Conversion transformations enable Warehouse Builder users to perform functions that allow conditional conversion of values. These functions achieve "if -then" constructions within SQL.

Conversion transformations in Warehouse Builder are implementations of the Oracle Database SQL conversion functions. For descriptions and examples of these transformations, see "Conversion Functions" in the *Oracle Database SQL Language Reference*.

- ASCIISTR
- COMPOSE
- CONVERT
- HEXTORAW
- NUMTODSINTERVAL
- NUMTOYMINTERVAL
- RAWTOHEX
- RAWTONHEX
- SCN_TO_TIMESTAMP
- TIMESTAMP_TO_SCN
- TO_BINARY_DOUBLE
- TO_BINARY_FLOAT
- TO_CHAR (character), TO_CHAR (datetime), TO_CHAR (number)
- TO_CLOB
- TO_DATE
- TO_DSINTERVAL
- TO_MULTIBYTE
- TO_NCHAR (character), TO_NCHAR (datetime), TO_NCHAR (number)
- TO_NCLOB
- TO_NUMBER
- TO_SINGLE_BYTE
- TO_TIMESTAMP
- TO_TIMESTAMP_TZ

- TO_YMINTERVAL
- UNISTR

Date Transformations

Date transformations provide Warehouse Builder users with functionality to perform transformations on date attributes. These transformations include Oracle Database SQL functions that are implemented by Warehouse Builder and custom functions provided with Warehouse Builder. The custom function are in the format `WB_<function name>`.

Following are the date transformations that are implementations of Oracle Database SQL functions. For descriptions and examples of these transformations, refer to the section "Datetime Functions" in the *Oracle Database SQL Language Reference*.

- ADD_MONTHS
- CURRENT_DATE
- DBTIMEZONE
- FROM_TZ
- LAST_DAY
- MONTHS_BETWEEN
- NEW_TIME
- NEXT_DAY
- ROUND
- SESSIONTIMEZONE
- SYSDATE
- SYSTIMESTAMP
- SYS_EXTRACT_UTC
- TRUNC

The custom Warehouse Builder Date transformations are:

- WB_CAL_MONTH_NAME on page 19-29
- WB_CAL_MONTH_OF_YEAR on page 19-29
- WB_CAL_MONTH_SHORT_NAME on page 19-30
- WB_CAL_QTR on page 19-30
- WB_CAL_WEEK_OF_YEAR on page 19-31
- WB_CAL_YEAR on page 19-31
- WB_CAL_YEAR_NAME on page 19-32
- WB_DATE_FROM_JULIAN on page 19-32
- WB_DAY_NAME on page 19-33
- WB_DAY_OF_MONTH on page 19-33
- WB_DAY_OF_WEEK on page 19-34
- WB_DAY_OF_YEAR on page 19-34

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. These include Database SQL functions that are implemented by Warehouse Builder and custom functions defined by Warehouse Builder. The custom functions are prefixed with `WB_`.

Table 19–2 lists the number transformations that are based on Oracle Database SQL numeric functions. The transformations are listed in a columnar table that reads down the columns from left to right to conserve space.

Table 19–2 List of Number Transformations Based on Database SQL Functions

Number Transformation Name	Number Transformation Name (Contd.)	Number Transformation Name (Contd.)
■ ABS	■ ACOS	■ ASIN
■ ATAN	■ ATAN2	■ BITAND
■ CEIL	■ COS	■ COSH
■ EXP	■ FLOOR	■ LN
■ LOG	■ MOD	■ NANVL
■ POWER	■ REMAINDER	■ ROUND (number)
■ SIGN	■ SIN	■ SINH
■ SQRT	■ TAN	■ TANH
■ TRUNC (number)	■ WIDTH_BUCKET	■

For descriptions and examples of these transformations, refer to the section titled "Numeric Functions" in the *Oracle Database SQL Language Reference*.

The custom numeric transformations are:

- WB_LOOKUP_NUM (on a number) on page 19-41
- WB_LOOKUP_NUM (on a varchar2) on page 19-41
- WB_IS_NUMBER on page 19-42

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 18-18.

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.

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:

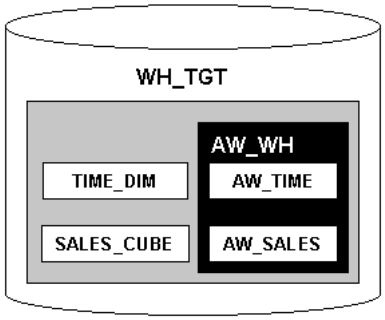
- WB_OLAP_AW_PRECOMPUTE on page 19-43
- WB_OLAP_LOAD_CUBE on page 19-44
- WB_OLAP_LOAD_DIMENSION on page 19-45
- WB_OLAP_LOAD_DIMENSION_GENUK on page 19-45

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 19–3.

Figure 19–3 Example of OLAP Transformations



The image displays a database containing a schema `WH_TGT`. This schema contains the following: relational dimension `TIME_DIM`, relational cube `SALES_CUBE`, and the analytic workspace `AW_WH` that contains the dimension `AW_TIME` and cube `AW_SALES`.

****

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(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 started. 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 started and the master job ID is returned via 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. These transformations are implementations of the Oracle Database SQL functions. For descriptions and examples of these transformations, see *Oracle Database SQL Language Reference*.

- DEPTH
- DUMP
- EMPTY_BLOB
- EMPTY_CLOB
- NLS_CHARSET_DECL_LEN
- NLS_CHARSET_ID
- NLS_CHARSET_NAME
- NULLIF
- NVL
- NVL2
- ORA_HASH
- PATH
- SYS_CONTEXT
- SYS_GUID
- SYS_TYPEID
- UID
- USER
- USERENV
- VSIZE

Spatial Transformations

Spatial transformations are 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
- SDO_AGGR_CONVEXHULL
- SDO_AGGR_MBR
- SDO_AGGR_UNION

For descriptions and examples of these transformations, refer to the *Oracle Spatial Developer's Guide*.

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, see *Oracle Streams Replication Administrator's 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' - i.e. 'abcde'
LCR.GET_VALUE('c2', 'new') will have the value for the column 'c2' - i.e. 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 about 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 by implementing database XML functions. Warehouse Builder also defines custom functions.

Following are the XML transformations that are implemented based on Oracle Database XML functions:

- EXISTSNODE
- EXTRACT
- EXTRACTVALUE
- SYS_XMLAGG
- SYS_XMLGEN
- XMLCONCAT
- XMLSEQUENCE
- XMLTRANSFORM

See Also:

- *Oracle Database SQL Language Reference* for descriptions for these transformations
- *Oracle Spatial Developer's Guide* for examples on using these transformations

The custom XML transformations are:

- WB_XML_LOAD on page 19-49
- WB_XML_LOAD_F on page 19-49

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 about 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 about the types handled and detailed information about `control_files`, see the *Oracle Warehouse Builder User's Guide*.

Importing PL/SQL

Use the Import Metadata Wizard to import PL/SQL functions, procedures, and packages into a Warehouse Builder project. You can edit, save, and deploy the imported PL/SQL functions and procedures. You can also view and modify imported packages.

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 Welcome page of the Import Metadata Wizard.
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 right arrow to move a single object or the Move All 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 module into which you imported the data.

Restrictions on Using Imported PL/SQL

The following restrictions apply to the usage of imported PL/SQL:

- You cannot edit imported PL/SQL packages.
- Wrapped PL/SQL objects are not readable.
- You can edit the imported package body but not the imported package specification.

Example: Reusing Existing PL/SQL Code

Scenario

A movie rental company periodically updates the customer rental activity in its CUST_RENTAL_ACTIVITY table, where it stores the rental sales and overdue charges data for each customer. This table is used for different mailing campaigns. For example, in their latest mailing campaign, customers with high overdue charges are offered the company's new pay-per-view service.

Currently, the movie rental company uses a PL/SQL package to consolidate their data. The existing PL/SQL package needs to be maintained manually by accessing the database. This code runs on an Oracle 8*i* database.

```
CREATE OR REPLACE PACKAGE RENTAL_ACTIVITY AS
  PROCEDURE REFRESH_ACTIVITY(SNAPSHOT_START_DATE IN DATE);
END RENTAL_ACTIVITY;
/
CREATE OR REPLACE PACKAGE BODY RENTAL_ACTIVITY AS
 PROCEDURE REFRESH_ACTIVITY(SNAPSHOT_START_DATE IN DATE) IS
   CURSOR C_ACTIVITY IS
    SELECT
      CUST.CUSTOMER_NUMBER CUSTOMER_NUMBER,
      CUST.CUSTOMER_FIRST_NAME CUSTOMER_FIRST_NAME,
      CUST.CUSTOMER_LAST_NAME CUSTOMER_LAST_NAME,
      CUST.CUSTOMER_ADDRESS CUSTOMER_ADDRESS,
      CUST.CUSTOMER_CITY CUSTOMER_CITY,
      CUST.CUSTOMER_STATE CUSTOMER_STATE,
      CUST.CUSTOMER_ZIP_CODE CUSTOMER_ZIP_CODE,
      SUM(SALE.RENTAL_SALES) RENTAL_SALES,
      SUM(SALE.OVERDUE_FEES) OVERDUE_FEES
    FROM  CUSTOMER CUST, MOVIE_RENTAL_RECORD SALE
    WHERE SALE.CUSTOMER_NUMBER = CUST.CUSTOMER_NUMBER AND
          SALE.RENTAL_RECORD_DATE >= SNAPSHOT_START_DATE
    GROUP BY
    CUST.CUSTOMER_NUMBER,
    CUST.CUSTOMER_FIRST_NAME,
    CUST.CUSTOMER_LAST_NAME,
    CUST.CUSTOMER_ADDRESS,
    CUST.CUSTOMER_CITY,
    CUST.CUSTOMER_STATE,
```

```
     CUST.CUSTOMER_ZIP_CODE;

    V_CUSTOMER_NUMBER NUMBER;
    V_CUSTOMER_FIRST_NAME VARCHAR2(20);
    V_CUSTOMER_LAST_NAME VARCHAR2(20);
    V_CUSTOMER_ADDRESS VARCHAR(50);
    V_CUSTOMER_CITY VARCHAR2(20);
    V_CUSTOMER_STATE VARCHAR2(20);
    V_CUSTOMER_ZIP_CODE VARCHAR(10);
    V_RENTAL_SALES NUMBER;
    V_OVERDUE_FEES NUMBER;

BEGIN
    OPEN C_ACTIVITY;
    LOOP
     EXIT WHEN C_ACTIVITY%NOTFOUND;
     FETCH
      C_ACTIVITY
     INTO
      V_CUSTOMER_NUMBER,
      V_CUSTOMER_FIRST_NAME,
      V_CUSTOMER_LAST_NAME,
      V_CUSTOMER_ADDRESS,
      V_CUSTOMER_CITY,
      V_CUSTOMER_STATE,
      V_CUSTOMER_ZIP_CODE,
      V_RENTAL_SALES,
      V_OVERDUE_FEES;

     UPDATE CUST_ACTIVITY_SNAPSHOT
     SET
      CUSTOMER_FIRST_NAME = V_CUSTOMER_FIRST_NAME,
      CUSTOMER_LAST_NAME = V_CUSTOMER_LAST_NAME,
      CUSTOMER_ADDRESS = V_CUSTOMER_ADDRESS,
      CUSTOMER_CITY = V_CUSTOMER_CITY,
      CUSTOMER_STATE = V_CUSTOMER_STATE,
      CUSTOMER_ZIP_CODE = V_CUSTOMER_ZIP_CODE,
      RENTAL_SALES = V_RENTAL_SALES,
      OVERDUE_FEES = V_OVERDUE_FEES,
      STATUS_UPDATE_DATE = SYSDATE
     WHERE
      CUSTOMER_NUMBER = V_CUSTOMER_NUMBER;

     IF SQL%NOTFOUND THEN
        INSERT INTO CUST_ACTIVITY_SNAPSHOT
        ( CUSTOMER_NUMBER,
          CUSTOMER_FIRST_NAME,
          CUSTOMER_LAST_NAME,
          CUSTOMER_ADDRESS,
          CUSTOMER_CITY,
          CUSTOMER_STATE,
          CUSTOMER_ZIP_CODE,
          RENTAL_SALES,
          OVERDUE_FEES,
          STATUS_UPDATE_DATE )
        VALUES
        ( V_CUSTOMER_NUMBER,
          V_CUSTOMER_FIRST_NAME,
          V_CUSTOMER_LAST_NAME,
          V_CUSTOMER_ADDRESS,
```

```
          V_CUSTOMER_CITY,
          V_CUSTOMER_STATE,
          V_CUSTOMER_ZIP_CODE,
          V_RENTAL_SALES,
          V_OVERDUE_FEES,
          SYSDATE );
      END IF;
    END LOOP;
END REFRESH_ACTIVITY;
END RENTAL_ACTIVITY;
/
```

Solution

This case study highlights the benefits of importing an existing custom PL/SQL package into Warehouse Builder and using its functionality to automatically maintain, update, and regenerate the PL/SQL code. Warehouse Builder enables you to automatically take advantage of new database features and upgrades by generating code that is optimized for new database versions. For example, if the customer has a PL/SQL package for Oracle 8*i*, then by importing it into Warehouse Builder they can generate code for both Oracle 8*i* and Oracle 9*i*.

Also, by importing a custom package and re-creating its operations through a Warehouse Builder mapping, you can transparently run and monitor the operations. Otherwise, you must manually access the database to verify and update the code. Warehouse Builder also enables you to perform lineage and impact analysis on all ETL operations while the Runtime Audit Browser monitors the running of the code and logs errors.

Case Study

You can migrate the PL/SQL code into Warehouse Builder by taking these steps:

- Step 1: Import the Custom PL/SQL Package
- Step 2: Create a 'Black Box' Mapping by using a custom transformation in a Warehouse Builder mapping
- Step 3: Migrate Custom Code into a Mapping by migrating the legacy PL/SQL code functionality into a new Warehouse Builder mapping and phase out the custom package
- Step 4: Generate Code for Oracle 9i

Follow these steps to handle a custom PL/SQL package in Warehouse Builder.

Step 1: Import the Custom PL/SQL Package

In the Project Explorer, expand the Transformations node under the Oracle module into which you want to import the PL/SQL package `refresh_activity(`*`DATE`*`)`. Use the Import Metadata Wizard to import the package by right-clicking Transformations and selecting **Import**. On the Filter Information page of this wizard, indicate that you are importing a PL/SQL Transformation.

After you finish the import, the package `refresh_activity(`*`DATE`*`)` appears under the Packages node of the Transformations folder.

Step 2: Create a 'Black Box' Mapping

You can use the `refresh_activity(DATE)` procedure directly in a mapping without making any changes to it. In the mapping, you add a Post-Mapping Process operator to the mapping, with the package `refresh_activity(DATE)` selected.

In this example, you can immediately take advantage of the existing custom code. The learning curve and investment on resources is minimal. You may decide to maintain all the existing and developed PL/SQL code in this manner, using Warehouse Builder only to develop new processing units. Warehouse Builder enables you to use mappings that use the legacy code along with the new mappings you create. In such a case, although you can generate code for these mappings in Warehouse Builder, they cannot use Warehouse Builder features to maintain, update, or audit the code.

Because the legacy code is used as a 'black box' that is not transparent to Warehouse Builder, you still need to maintain the legacy code manually. Thus, you cannot take advantage of the Warehouse Builder features, such as runtime audit browser, lineage and impact analysis, and optimized code generation, that rely on infrastructure code and metadata available for Warehouse Builder generated mappings.

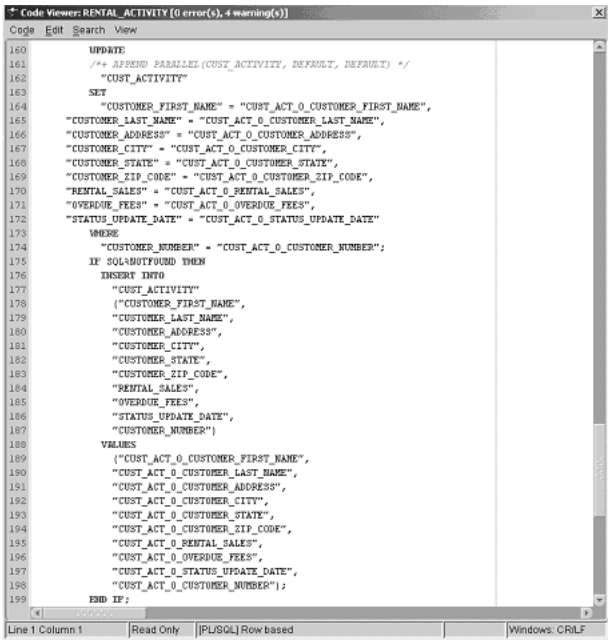
Follow the next steps to take advantage of these features in Warehouse Builder and to automatically maintain, monitor, and generate your PL/SQL code.

Step 3: Migrate Custom Code into a Mapping

To take advantage of the code generation, maintenance, and auditing features, you can gradually migrate the legacy PL/SQL code functionality into a mapping and phase out the custom 'black box' package. The mapping created to provide the PL/SQL code functionality is called `Rental_Activity`.

The recommended method is to test out this new mapping by running it side by side with the 'black box' mapping. If the testing is successful and the new mapping can perform all the operations included in the custom code, the 'black box' mappings can be phased out. Warehouse Builder enables you to maintain, update, and generate code from a mapping without performing manual updates in the database. Figure 19–4 shows a sample of code generated from the `Rental_Activity` mapping that replicates the operations of the custom PL/SQL package for the movie rental company.

Figure 19–4 Sample Code



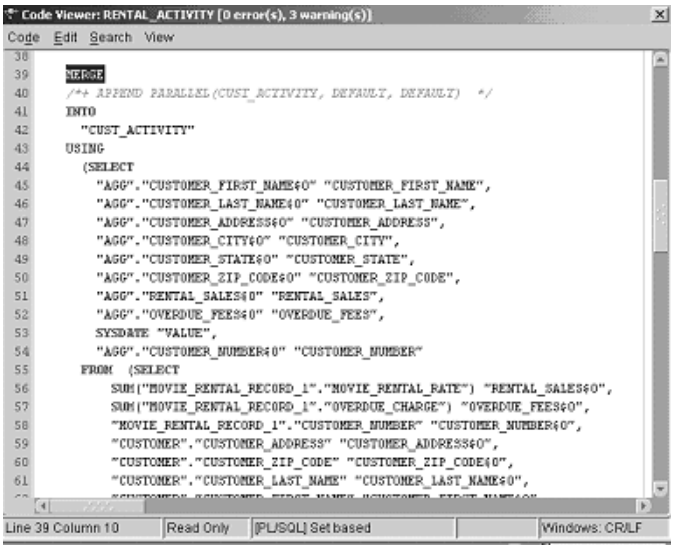
This screenshot displays the sample code that can be seen in the Code Viewer window. At the top of this window is the menu bar containing the following items, ordered from left to right: Code, Edit, Search, and View. The area below the menu bar displays the code, with line numbers on the left for each line of code. Below the code is the status bar.

****

Step 4: Generate Code for Oracle 9*i*

If you upgrade to Oracle 9*i* version of the database, you only need to re-deploy the `Rental_Activity` mapping created in Step 3. Warehouse Builder generates code optimized for the new database version. Figure 19–5 shows the `MERGE` statement from a sample of code generated for the same mapping for Oracle 9*i*.

Figure 19–5 Sample Code for Oracle 9i



This screenshot displays the sample code that can be seen in the Code Viewer window. At the top of this window is the menu bar containing the following items, ordered from left to right: Code, Edit, Search, and View. The area below the menu bar displays the code, with line numbers on the left for each line of code. Below the code is the status bar.

****

No manual steps are required to maintain and generate the new code. Also, you can transparently monitor and maintain their ETL operations. Warehouse Builder enables them to perform lineage and impact analysis on their mappings and the Runtime Audit Browser enables them to track and log errors when running the mappings.

20

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. A Process Flow allows activities to be linked together and to describe constraints between the activities. Constraints can be conditional branches, loops, parallel flows or serial dependencies. Activities can be mappings, transforms or external commands such as email, FTP commands, and operating system executables.

You can use process flows to manage dependencies between mappings. To schedule mappings, process flows, and other executable objects, see "Process for Defining and Using Schedules" on page 26-2.

This chapter contains the following topics:

- About Process Flows
- Instructions for Defining Process Flows
- About the Process Flow Editor
- Adding Activities to Process Flows
- Creating and Using Activity Templates
- About Transitions
- Expressions
- Defining Transition Conditions
- Example: Using Process Flows to Access Flat Files with Variable Names
- Example: Using Process Flows to Transfer Remote Files

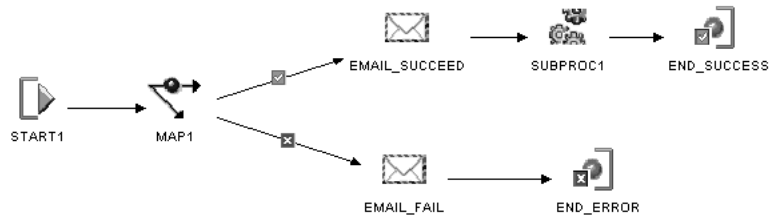
About Process Flows

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. A Process Flow is considered as a type of activity, so a Process Flow can start other process flows.

Figure 20–1 shows an example of a process flow that starts a mapping MAP1. If the mapping completes successfully, then Warehouse Builder sends an email notification EMAIL_SUCCEED and starts another process flow SUBPROC1. If the mapping fails, then Warehouse Builder sends an email EMAIL_FAIL and ends the process flow.

Figure 20–1 Sample Process Flow



This screenshot shows a sample process flow. It has a Start activity, which is linked to a mapping MAP1. The mapping is connected to two conditions ordered from top to bottom, EMAIL_SUCCESS and EMAIL_FAIL.

If MAP1 is successful, an email is sent using the EMAIL_SUCCESS activity and then the subprocess, SUBPROC, that is linked to it is run. After executing the subprocess, the process flow ends with END_SUCCESS activity. There is an arrow from MAP1 to EMAIL_SUCCESS, from EMAIL_SUCCESS to SUBPROC and from SUBPROC to END_SUCCESS.

If the Mapping is not successful, then it will send a failure email, represented by EMIAL_FAIL, and the process will end with an error. This is represented by an arrow from MAP1 to EMAIL_FAIL and from EMAIL_FAIL to END_ERROR.

****

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 to manage and deploy process flows. You can validate, generate, and deploy process flows at either the module or the package level.

You can design a process flow that starts other process flows as long as they are in 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.

For example, Figure 20–1 shows 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.

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. The generated XPDL can be used to integrate with any workflow engine that supports the WfMC standard.

Warehouse Builder provides integration with Oracle Workflow. From the Warehouse Builder Control Center, you can deploy process flow packages or modules to Oracle Workflow.

Instructions for Defining Process Flows

Before You Begin

To enable deployment of process flows, install Oracle Workflow as described in "Enabling Integration with Oracle Workflow" in the *Oracle Warehouse Builder Installation and Administration Guide*.

To define a process flow, refer to the following sections:

1. Creating Process Flow Modules on page 20-3
2. Creating Process Flow Packages on page 20-4
3. Creating Process Flows on page 20-4
4. Creating and Using Activity Templates on page 20-10
5. Adding Activities on page 20-8
6. Connecting Activities on page 20-15
7. Using Activities in Process Flows on page 21-1
8. Using Parameters and Variables on page 20-16
9. Configuring Process Flows Reference on page 22-41
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 day or multiple days as described in "Process for Defining and Using Schedules" on page 26-2.
12. Deploying Process Flows as described in "The Deployment and Execution Process" on page 25-3.

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. Alternatively, select an existing location from the list. 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, inserts its name in the Project Explorer, and prompts you to create a process flow package.

Creating Process Flow Packages

After you create a Process Flow module, you can create a 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 click **New.**

   Warehouse Builder displays the Create Process Flow Package dialog box.

2. Type a name and optional description for the process flow package.

   If you intend to integrate with Oracle Workflow, please note that Oracle Workflow restricts package names to 8 bytes.

3. Click **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 a process flow.

To create a process flow:

1. Right-click a process flow package in the Project Explorer and click **New.**

   Warehouse Builder displays the Create Process Flow dialog box.

2. Type a name and optional description for the process flow.

   Note: If you intend to schedule a process flow, 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 running the process flow.

3. Click **OK.**

   Warehouse Builder runs the Process Flow Editor 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 20-3.

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 can 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 to carry out the activities.

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 Process Flow Editor, the title bar displays the name of the process flow.
- **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 is displayed with a Start activity and an End activity.
- **Palette:** When you first start the Process Flow Editor, Warehouse Builder displays the palette along the left side. The Process Flow Editor contains activity icons that you can drag and drop on to the canvas. You can relocate the palette anywhere on the editor. You can choose to hide or display the palette by clicking the collapse icon on the palette.
- **Indicator Bar:** On the lower panel, under the Bird's Eye View panel and Canvas panel, you can see mode icons, indicators, and descriptions.

  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 a window by placing the mouse pointer on the title bar of the window and then dragging the window to the required position.

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 start the editor, Warehouse Builder displays an Explorer panel 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 start the editor, Warehouse Builder displays the Object Details panel on the left side. This panel 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. If you select an activity parameter in the Explorer, then the object details window displays the properties for that parameter. You can edit properties displayed with a white background but not those with a gray background.

Palette

When you first start 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's Eye View

Use the Bird's Eye View panel to navigate large and complex process flows.

Opening the Process Flow Editor

To open the Process Flow Editor:

1. From the Process Flows node in the Project Explorer, select a process flow module. If no process flow modules are listed, then create a process flow module as described in "Creating Process Flow Modules" on page 20-3.
2. Select a process flow package from a process flow module. If no process flow packages are listed, then create a process flow package as described in "Creating Process Flow Packages" on page 20-4.
3. Select a process flow from the Project Explorer. If no process flows are listed in the process flow package, then right-click the process flow package and select **Create Process Flow.**

   Warehouse Builder prompts you to name the process flow and then starts the editor for you.
4. To open an existing process flow, double-click the process flow in the Project Explorer.

   Alternatively, select a process flow and then from the Edit menu, select **Open Editor.** You can also, select a process flow and press **Ctrl**+**O.** You can also, right-click a process flow, and select **Open Editor**.

   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 will use frequently when designing Process Flows include the following:

Select mode

This screenshot shows the arrow icon for the Select mode operator. It contains an arrow.

****

Use the select mode to select objects on the canvas. When you select an activity, the editor displays a blue border around the activity. 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.

Navigation Edge

This screenshot shows the Navigation Edge icon. It contains a left parenthesis, an arrow pointing to the right side and a right parenthesis.

****

Navigation Edge assists you in navigating complex designs on the canvas. Select the icon from the toolbar and then select an 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 navigation edge icon and then select the last activity in the flow.

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

Adding Activities to Process Flows

You can add activities in a process flow using the Explorer tree in the Warehouse Builder.

About Activities

Activities represent units of work for the process flow such as starting 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 start Warehouse Builder objects such as mappings, transformations, or other process flows. The process flow runs the object and provides a commit statement.

- **Utility Activities:** These activities enable you to perform services such as sending emails and transferring files.
- **Control Activities:** These activities enable you to control the progress and direction of the process flow. For instance, use the Fork activity to run 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 20-10. For email, 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 21-1.

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.

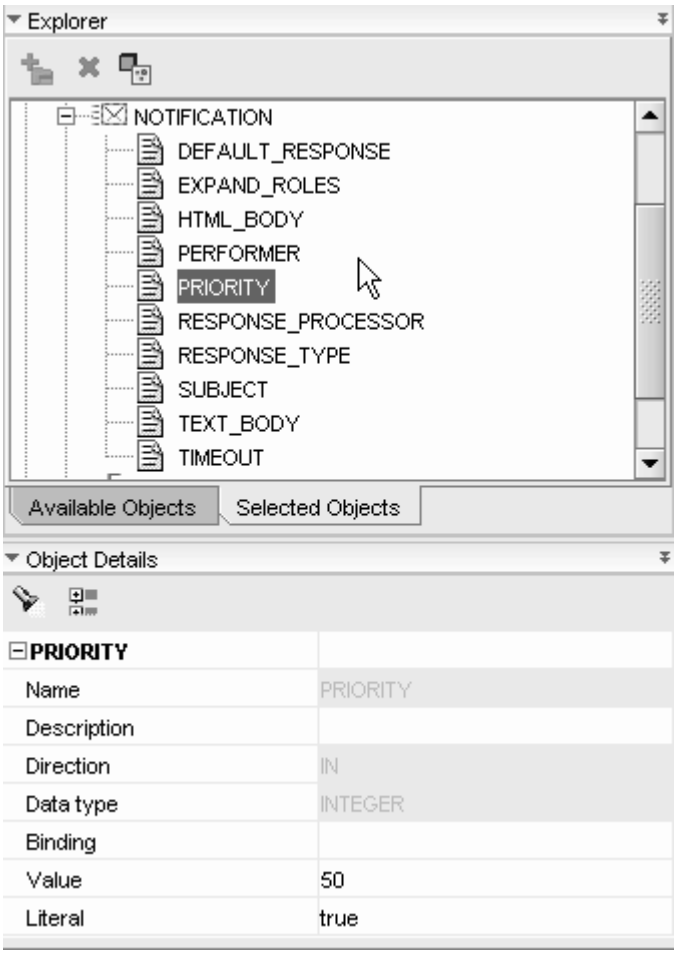
   The editor lists the activity on the explorer pane located at the left side of the editor and in the object details pane along the left side.

4. In Object Details pane**,** enter the parameters for the activity.

   These parameters vary according to the type of activity. 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.

   For example, Figure 20–2 shows the parameters for a notification activity which includes DEFAULT_RESPONSE, EXPANDED_ROLES, HTML_BODY, PERFORMER, PRIORITY, RESPONSE_PROCESSOR, RESPONSE_TYPE, SUBJECT, TEXT_BODY, and TIMEOUT.

Figure 20–2 The Parameters for a Notification Activity



This screenshot shows the parameters for a Notification activity. This screenshot has two panels from top to bottom, Explorer and Object Details. The Explorer panel has two tabs at the bottom, from left to right, Available Objects and Selected Objects. The Selected Objects tab is currently displayed. It contains the NOTIFICATION activity, that is expanded to display the following parameters, from top to bottom: DEFAULT_RESPONSE, EXPAND_ROLES, HTML_BODY, PERFORMER, PRIORITY, RESPONSE_PROCESSOR, RESPONSE_TYPE, SUBJECT, TEXT_BODY, and TIMEOUT.

Below this Explorer panel is the Object Details panel. It contains the object details in a tabular format.

****

Parameters for Activities

Each parameter has the following properties:

Name

This is a name property of the activity parameter. For information about a specific parameter, look up the activity by name under "Using Activities in Process Flows" on page 21-1.

Direction

The direction property 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 data type property 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 the binding property 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,** then it overrides any text you assign to **Value.**

Literal

If you type in a value for the parameter in the field **Value,** then indicate whether the value is a 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 20–1 provides the Literal Value Type, Format, and Example.

Table 20–1 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, 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 is displayed.

   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 20-13 for instructions about 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 3-7. Warehouse Builder maintains a business and a physical name for each object in the workspace. 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. So, when working in the business name mode, if you assign an activity template name that includes mixed cases, special characters, and spaces, then Warehouse Builder creates a default physical name for the objects.

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, an 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 2000 alphanumeric characters and can contain blank spaces. Specifying a description for an activity template is optional.

Activity Templates

The following activity templates are available from the list.

- Assign
- Email
- FTP

- File Exists
- Manual
- Notification
- Set Status
- Sqlplus
- User Defined
- Wait

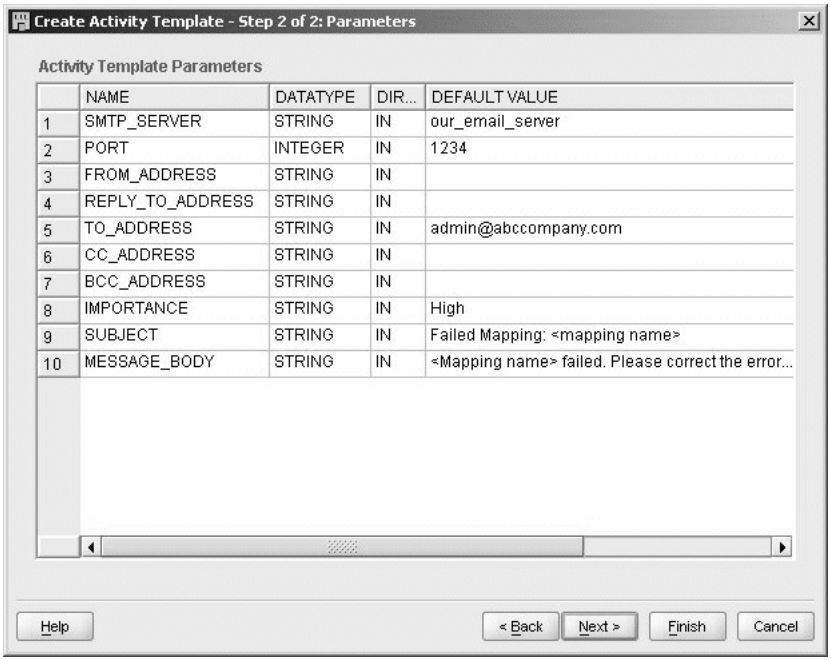
Parameters Page

The wizard displays parameters based on the type of activity you previously selected in the Activity Templates.

Enter default values for the activity. When you use the activity template in a process flow, you can retain or edit the default values.

In Figure 20–3, for example, you could edit the default values for the email subject and message body to contain the name of the mapping.

Figure 20–3 Parameters Page for Email Activity Template



This screenshot shows the parameters page for the Email activity template. The activity template parameters are listed in a table with the following column names, ordered from left to right: NAME, DATATYPE, DIRECTION, and DEFAULT VALUE. The table contains ten rows.

At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page are the following buttons, ordered from left to right: Back, Next, Finish, and Cancel.

****

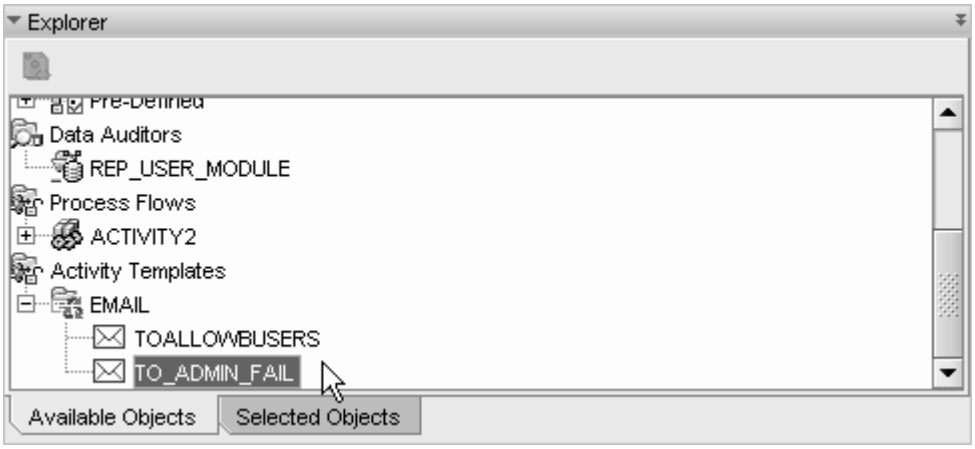
Using Activity Templates

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 Flow Editor, right-click the Process Flow module and select **Open Editor**.
3. In the Process Flow Editor, click the **Available Objects** tab in the Explorer panel and expand **Activity Templates**.

   Figure 20–4 displays the Explorer window with the activity template expanded.

Figure 20–4 Explorer Panel with an Activity Template Selected



This screenshot shows the Explorer panel with an Activity template selected. The Explorer panel consist of two tabs, ordered from left to right, Available Objects and Selected Objects. The TO_ADMIN_FAIL located in the Email node of the Activity Templates is currently selected in the screenshot.

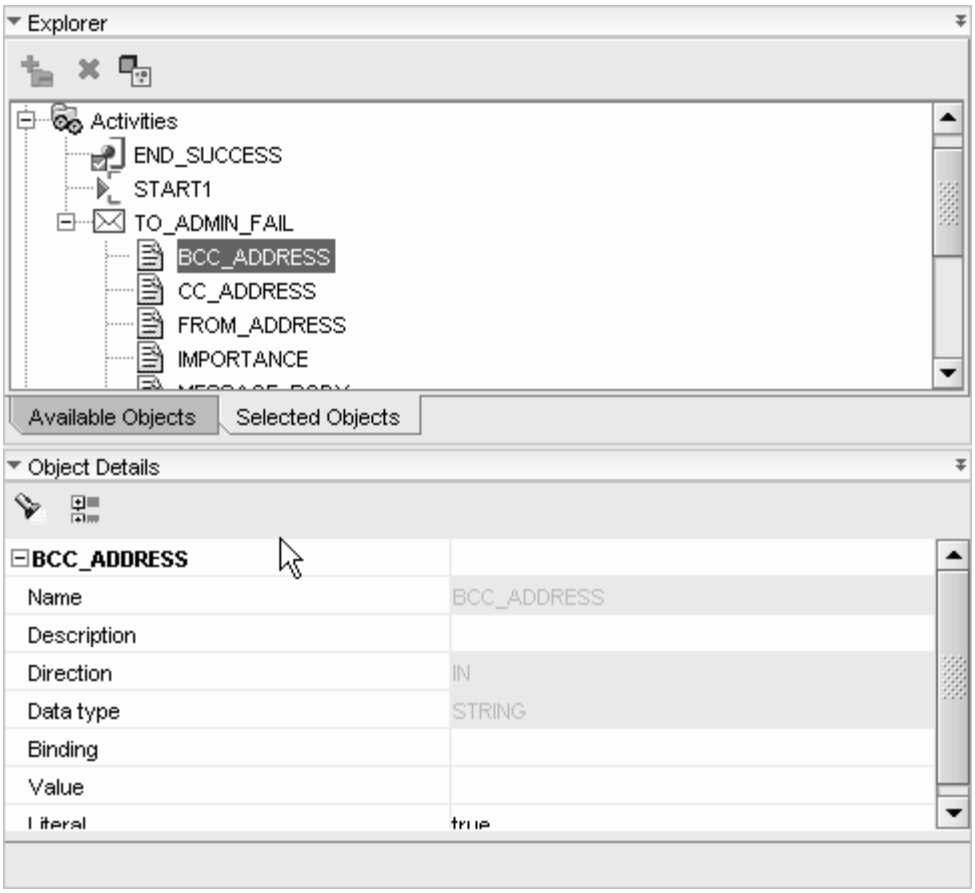
****

4. Drag and drop the activity template onto the canvas.

   Activity templates in a process flow acts 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 panel.

   Figure 20–5 displays the Explorer panel with the BCC_ADDRESS parameter of the EMAIL activity selected.

Figure 20–5 Editing an Activity Template



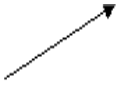
This screenshot shows how to edit an Activity template. This screenshot contains two panels ordered from top to bottom, Explorer panel and the Object Details panel. The Explorer panel consists of two tabs, ordered from left to right, Available Objects and Selected Objects. The BCC_ADDRESS located under the TO_ADMIN_FAIL is currently selected.

Below the Explorer panel is the Object Details panel that displays the properties for BCC_ADDRESS in a tabular format.

****

About Transitions

Use transitions to indicate the sequence and conditions in which activities occur in the process flow. You can use transitions to run an activity based on the completion state of the preceding activity.



This screenshot contains a diagonal arrow extending from the bottom left to the top right.

****

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 FORK, then there may be only one unconditional transition and it is used when no other conditional transition is activated, for example, the final `ELSE` condition 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.
- 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 carried on as a result of the "loop."

Connecting Activities

To create dependencies using transitions:

1. When working in the select mode, place your mouse pointer along the right border of the activity icon along 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 under 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 that 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" on page 20-18. If you select a condition, then 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 require additional configuration. These configuration details for a given activity are listed in "Using Activities in Process Flows" on page 21-1.

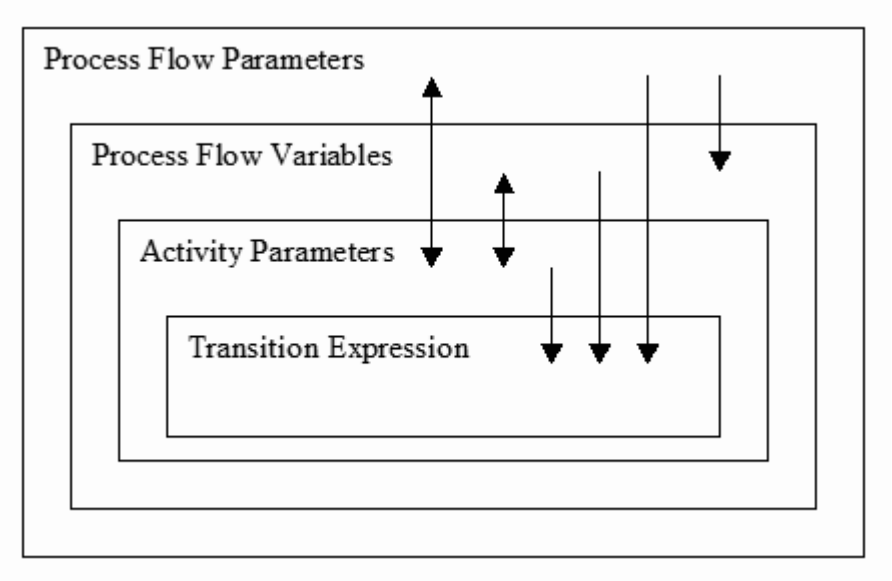
Using Parameters and Variables

Process flows and activities support the PL/SQL parameter passing concept, allowing data to be passed and reused through parameterization. This is accomplished through data stores, which are implemented as either parameters or variables. Process flow allows the data to be passed between data stores.

- Parameters allow 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 running the process flow. Variables are used to pass data between activities.

Figure 20–6 shows the direction in which the data is passed.

Figure 20–6 Relationship between the scope and the direction in which the data is passed



This screenshot shows the relationship between the scope and the direction in which the data is passed. This screenshot contains four rectangles one inside the other. The outermost rectangle represents Process Flow Parameters. The second rectangle represents Process Flow Variables. The third rectangle represents Activity Parameters. The innermost rectangle represents Transition Expression. A double-headed arrow points from top to bottom connecting the Process Flow rectangle and the Activity Parameters rectangle. A double-headed arrow points from top to bottom connecting the Process Flow Variables rectangle and the Activity Parameters rectangle. A

single-headed arrow points from top to bottom connecting the Activity Parameters rectangle and the Transition Expression rectangle. A single-headed arrow points from top to bottom connecting the Process Flow Variables rectangle and the Transition Expression rectangle. A single-headed arrow points from top to bottom connecting the Process Flow Parameters rectangle and the Transition Expression rectangle. A single-headed arrow points from top to bottom connects the Process Flow Parameters rectangle and the Process Flow Variables rectangle.

****

Process flows follow the following rules for allowing the data to be passed between data stores:

1. Process flow variables can be initialized from flow parameters, but the reverse is not allowed.
2. Activity parameters can pass data bidirectionally between process flow variables and process flow parameters.
3. Transition expressions can be evaluated against their source activity parameters, 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, you can reference the hidden data store name. 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.
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.

A variable may not pass data out to a Process Flow parameter. So, 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 use of PL/SQL expressions for the derivation of parameter values and the use of 'complex expression' transitions.

The expression must produce a correctly typed value for data store. Automatic conversion from `VARCHAR` is supported. When the expression is associated with a transition a BOOLEAN result is expected.

During evaluation, an expression will have access to the outer scope which encloses it. So, an expression for an activity parameter will be able to use process flow variables and process flow parameters in its evaluation.

The PL/SQL expression is run in the context of the Control Center user who requested the process of the activity. However, in the case where the Oracle Workflow schema is hosted in a remote database instance, the effective user of the generated database link will be used instead. A different Control Center user may be selected by configuring the process flow and specifying an 'Evaluation Location.' So the expression may reference any PL/SQL function that is accessible to the Control Center user.

Global Expression Values

Warehouse Builder also makes additional data values available to the expression from the current activity and the owning process flow.

Table 20–2 lists these global expression values.

Table 20–2 Global Expression Values

Identifier	Type	Description
NUMBER_OF_ERRORS	NUMBER	Number of errors reported on completion of activity execution
NUMBER_OF_WARNINGS	NUMBER	Number of warnings reported on completion of activity 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
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

Table 20–3 lists the additional constants 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. These are displayed on the canvas as shown in Table 20–3.

Table 20–3 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 preceding notification activity ends with an extended result.

Extended transition is valid only for Notification activities because they are the only activity that return an extended result. The activity acquires this icon when set to an outcome of `#MAIL, #NOMATCH, #TIE, or #TIMEOUT`. Table 20–4 lists the output and the description of the Extended transition.

Table 20–4 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.
#MAIL	A mail error occurred for the notification. Some recipients did not receive an email notification and so it was canceled.
#TIMEOUT	The notification did not receive a response within the configured amount of time.

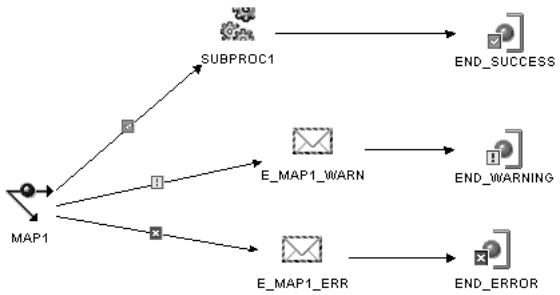
If the activity has only one outgoing activity, then you can specify any of the conditions listed in Table 20–3 or leave the transition as unconditional.

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 is 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 evaluates that more than one outgoing transition is true.

Figure 20–7 shows a portion of a process flow in which different activities are triggered based on the three possible completion states of MAP1. Because 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 20–7 Outgoing Transition Conditions



This screenshot shows a mapping called MAP1 on the left side. In the center, ordered from top to bottom, are a subprocess activity SUBPROC1 and two Email activities E_MAP1_WARN and E_MAP1_ERR. On the right are the following activities, ordered from top to bottom: END_SUCCESS, END1, and END_ERROR.

There are arrows from MAP1 to SUBPROC1, E_MAP_WARN, and E_MAP_ERR. This represents the flow of data from MAP1 to these activities, depending on the result of the mapping execution. If there are any warnings, the data flows from the mapping to the activity E_MAP1_WARN and then to END1. If there are errors in the mapping, the data flows from the mapping to the E_MAP1_ERR and then to the END_ERROR.

****

Example: Using Process Flows to Access Flat Files with Variable Names

Scenario

Your company relies on a legacy system that writes data to a flat file on a daily basis and assigns a unique name to the file based on the date and time of its creation. You would like to create a mapping that uses the generated flat files as a source, and transforms and loads the data to a relational database. However, mappings require files to have permanent names and, in this situation, the name of the source file changes each time the file is created.

Solution

In Warehouse Builder, you can design a process flow that locates the generated file in a specific directory, renames it to a permanent name you designate, and starts a dependent mapping. You can now use the permanent flat file name as the source for your mapping.

Case Study

This case study describes how to create a process flow and a mapping to extract data from a legacy system that generates flat files with variable names. The process flow relies on the use of an external process activity. Assume the following information for the purposes of this case study:

- **Generated Flat File:** The legacy system generates a flat file containing sales data on a daily basis. It saves the file to `c:\staging_files` directory and names the file based on the time and date, such as `sales010520041154.dat`. Every generated file is saved to the same directory and begins with the word `sales`, followed by the timestamp information.

- **Permanent Flat File Name:** You decide to rename the generated file name to `s_data.dat`. This is the name you reference as the flat file source in the mapping.
- **Process Activity:** You design a process flow named `OWF_EXT` to execute batch commands in DOS that copies the generated file, saves it as `s_data.dat`, and deletes the originally generated file.

Your objective is to create logic that ensures the generated flat file is renamed appropriately before it triggers the execution of a mapping.

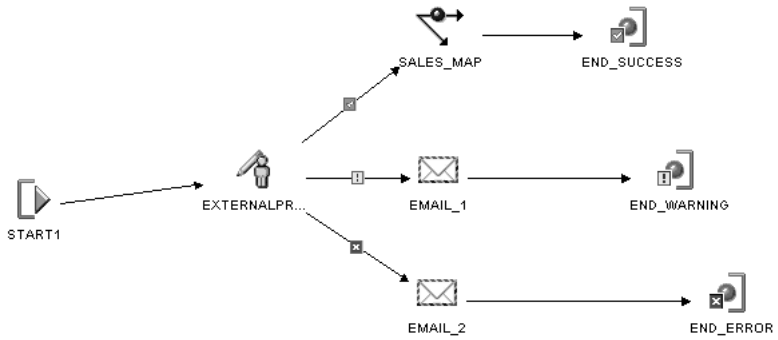
To extract data from a generated flat file with a name that varies with each generation, refer to the following sections:

1. "Creating the Process Flow" on page 20-21
2. "Setting Parameters for the External Process Activity" on page 20-22
3. "Configuring the External Process Activity" on page 20-26
4. "Designing the Mapping" on page 20-27
5. "Deploying and Executing" on page 20-27

Creating the Process Flow

Create a process flow that starts a mapping on the condition that the external process activity completes successfully. Your process flow should resemble Figure 20–8. For more information on creating the process flow, refer to "Instructions for Defining Process Flows" on page 20-3.

Figure 20–8 Process Flow with External Process Transitioning to a Mapping



This screenshot displays the process flow with External Process Transitioning to a mapping. On the left is the START activity, to the right of which is the EXTERNALPROCESS activity. To the right of the EXTERALPROCESS activity, ordered from top to bottom, are the following activities: SALES_MAP, EMAIL_1, and EMAIL_2. To the right of these, ordered from top to bottom, are the following activities: END_SUCCESS, END_WARNNG, and END_ERROR.

The START connects to the EXTERNALPROCESS activity through an arrow. The EXTERNALPROCESS activity had arrows connecting to SALES_MAP, EMAIL_1, and EMAIL_2. Arrows connect SALES_MAP to END_SUCCESS, EMAIL_1 to END_WARNING, and EMAIL_2 to END_ERROR.

If EXTERNALPROCESS is successful, then the data flows to SALES_MAP and the process ends with END_SUCCESS activity. If there are any errors in external process, then an e-mail is sent using the EMAIL_1 activity, and the process ends with END_

WARNING activity. If the external process fails, then an EMAIL_2 is sent and the process ends with END_ERROR.

****

Setting Parameters for the External Process Activity

This section describes how to specify the DOS commands for renaming the generated file. The DOS commands you issue from the external process activity should be similar to the following:

```
copy c:\staging_files\sales*.* c:\staging_files\s_data.dat
del c:\staging_files\sales*.*
```

The first command copies the temporary file into a file with a fixed name `s_data.dat`. The second command deletes the originally generated file.

You can either direct Warehouse Builder to a file containing the script of commands or you can store the commands in the Warehouse Builder user interface. Choose one of the following methods:

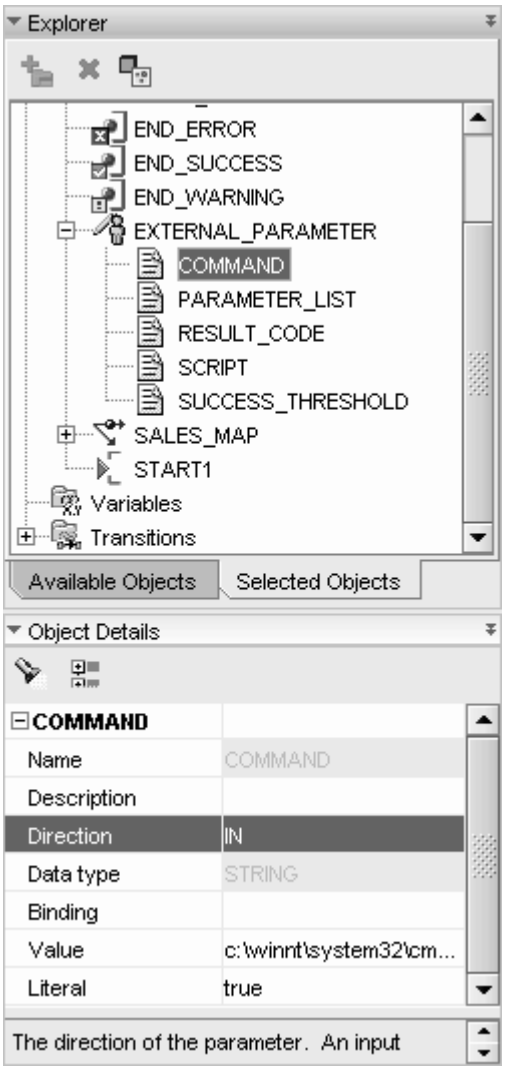
- Method 1: Write a script within Warehouse Builder
- Method 2: Call a script maintained outside of Warehouse Builder

Method 1: Write a script within Warehouse Builder

Choose this method when you want to maintain the script in Warehouse Builder. Consider using this method when the script is small and need not be very flexible.

For this method, write or copy and paste the script into the Value column of the SCRIPT parameter. In the COMMAND parameter, type the path to the DOS shell command such as `c:\winnt\system32\cmd.exe`. Also, type the `${Task.Input}` variable into the Value column of the PARAMETER_LIST parameter. Your Activity View should resemble Figure 20–9.

Figure 20–9 External Process Parameters When Script Maintained in this Product



This screenshot shows the list of parameters available for scripting in an activity. This screenshot shows two panels ordered from top to bottom: Explorer and Object Details. The Explorer panel has two tabs, ordered from left to right: Available Objects and Selected Objects. The Selected Objects tab is currently displayed.

****

Although this case study does not illustrate it, you can use substitution variables in the script when you maintain it in Warehouse Builder. This prevents you from having to update activities when server files, accounts, and passwords change.

Table 20–5 lists the substitute variables you can type for the external process activity. *Working* refers to the computer hosting the Runtime Service, the *local* computer in this case study. Remote refers to a server other than the Runtime Service host. You designate which server is remote and local when you configure the activity as described in "Configuring the External Process Activity" on page 20-26. These values are set when you register the locations at deployment.

Table 20–5 Substitute Variables for the External Process Activity

Variable	Value
${Working.Host}	The host value for the location of the Runtime Service host.
${Working.User}	The user value for the location of the Runtime Service host.
${Working.Password}	The password value for the location of the Runtime Service host.
${Working.RootPath}	The root path value for the location of the Runtime Service host.
${Remote.Host}	The host value for a location other than the Runtime Service host.
${Remote.User}	The user value for a location other than the Runtime Service host.
${Remote.Password}	The password value for a location other than the Runtime Service host.
${Remote.RootPath}	The root path value for a location other than the Runtime Service host.
${Deployment.Location}	The deployment location.

Method 2: Call a script maintained outside of Warehouse Builder

If extra maintenance is not an issue, you can point Warehouse Builder to a file containing a script including the necessary commands. This method is more flexible as it enables you to pass in parameters during execution of the process flow.

The following example shows how to call an external process script outside of Warehouse Builder and illustrates how to pass parameters into the script during execution of the process flow. This example assumes a Windows operating system. For other operating systems, issue the appropriate equivalent commands.

To call a script outside the external process activity:

1. Write the script and save it on the file directory. For example, you can write the following script and save it as `c:\staging_files\rename_file.bat:`

   copy c:\staging_files\%1*.dat c:\staging_files\s_data.dat

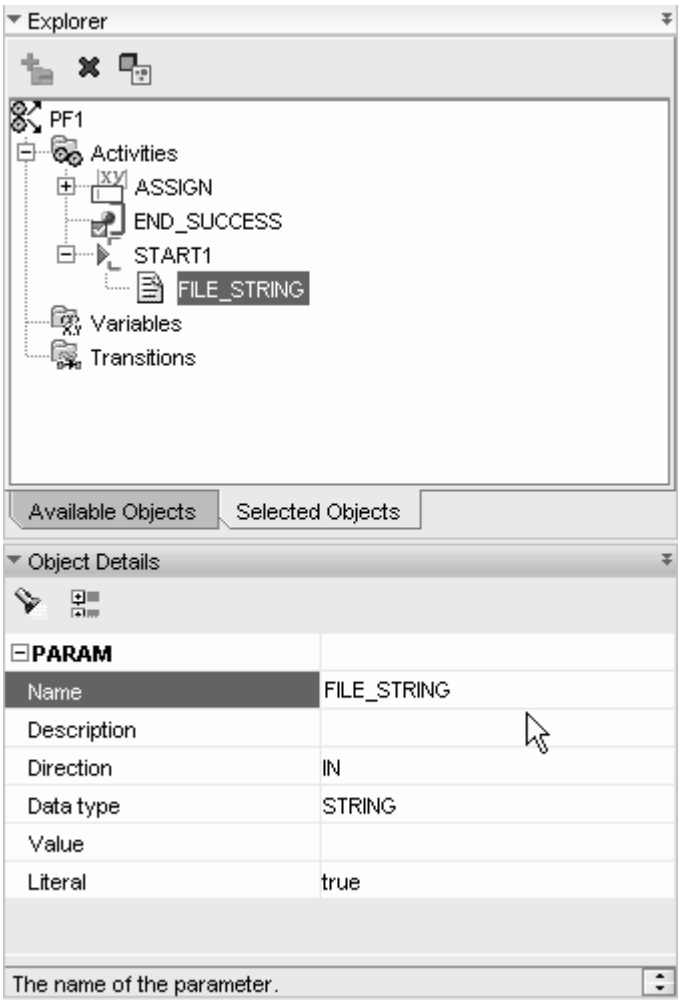
   del c:\staging_files\%1*.dat

   In this sample script, we pass a parameter %1 to the script during the execution of the process flow. This parameter represents a string containing the first characters of the temporary file name, such as `sales010520041154.`

2. Select the start activity on the canvas to view and edit activity parameters in the Available Objects tab of the Explorer panel displayed in the Process Flow Editor.

   To add a start parameter, click **Add** on the upper left corner of the Explorer pane in the Available Objects tab. Create a start parameter named `FILE_STRING` as shown in Figure 20–10. During execution, Warehouse Builder will prompt you to type a value for `FILE_STRING` to pass on to the `%1` parameter in the `rename_file.bat` script.

Figure 20–10 Start Activity in the Activity View



This screenshot shows the Start Activity in the Explorer panel of the Process Flow Editor. This panel contains two tabs Available Objects and Selected Objects, with Available Objects being the active tab. FILE_STRING is currently selected under the START1 activity.

The lower part of the screenshot contains the Object Details panel. The activity details are available in the Object Details panel, with the following rows, ordered from top to bottom: Name, Description, Direction, Data type, Value, and Literal.

****

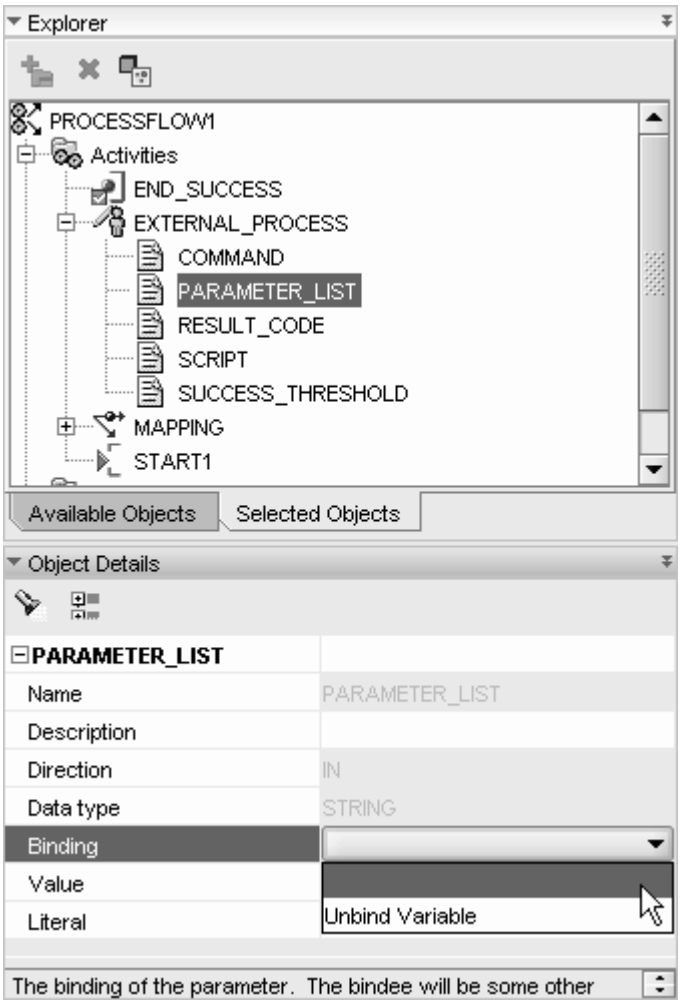
3. Select the external process activity on the canvas and edit its parameters as shown in Figure 20–11.

   For the COMMAND parameter, type the path to the script in the column labeled Value. If necessary, use the scroll bar to scroll down and reveal the column. For this example, type `c:\staging_files\rename_file.bat`.

   For PARAMETER_LIST, click the row labeled Binding and select the parameter you defined for the start activity, FILE_STRING

   Accept the defaults for all other parameters for the external process. Your Activity View for the external process activity should resemble Figure 20–11.

Figure 20–11 External Process Parameters When Calling an Outside Script



This screenshot shows the EXTERNALPROCESS activity. The upper part of the screenshot displays the Explorer panel that contains the Available Objects and Selected Objects tabs. The Available Objects is currently selected.

The External process activity details are available in table, with the following column names, ordered from left to right: ACTIVITY, PARAMETER, DATATYPE, DIRECTION, BINDING, and VALUE. The BINDING column, which is currently selected, contains a list.

****

Configuring the External Process Activity

When you apply conditions to the outgoing transitions of an external process, you must define the meaning of those conditions when you configure the external process activity.

To configure the external process activity:

1. Right-click the process flow on the navigation tree and select **Configure.**
2. Expand the external process activity and the Path Settings. Warehouse Builder displays the configuration settings.

3. Complete this step if you wrote the script in the Warehouse Builder user interface using the substitution variables related to Remote Location, Working Location, and Deployment Location as listed in Table 20–5 on page 20-24. Use the list to select the values.

   Because this case study does not use substitution variables, accept the defaults values.

4. Set the Deployed Location to the computer where you deploy the process flow.

5. Select **Use Return as Status**.

   This ensures that the process flow uses the external process return codes for determining which outgoing transition to activate. For the process flow in this case study, shown in Figure 20–8 on page 20-21, if the external process returns a success value, the process flow continues down the success transition and executes the downstream mapping.

Designing the Mapping

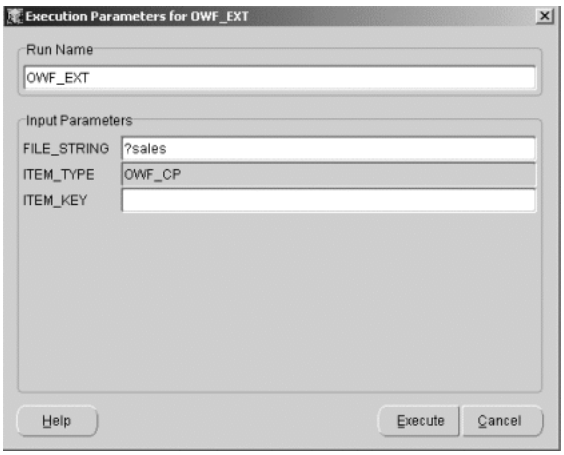
Now you can design a mapping with `s_data.dat` as the source. You can create a PL/SQL mapping or a SQL*Loader mapping. For a PL/SQL, map the flat file source to an external table and design the rest of the mapping with all the operators available for a PL/SQL mapping. For SQL*Loader, map the flat file source to a staging table and limit the mapping to those operators permitted in SQL*Loader mappings.

Deploying and Executing

Deploy the mapping. Also, deploy the process flow package or module containing the process flow `OWF_EXT`.

Execute the process flow manually. When you execute the process flow, Warehouse Builder prompts you to type values for the parameter you created to pass into the script, FILE_STRING For this case study, type `?sales` where the question mark is the separator, as shown in Figure 20–12. The external activity then executes the command `rename_file.bat sales`.

Figure 20–12 External Process Activity in the Activity View



This screenshot shows the Execution Parameters dialog box. At the top is the Run Name field. Below this field is the Input Parameters box containing the following fields, ordered from top to bottom: FILE_STRING, ITEM_TYPE, and ITEM_KEY.

At the lower left-hand corner of the dialog box is the Help button. At the lower right-hand corner of the dialog box are the following buttons, ordered from left to right: Execute and Cancel.

****

Subsequent Steps

After you successfully execute the process flow manually, consider creating a schedule.You can define a daily schedule to execute the process flow and therefore the mapping.

Creating a Schedule

Use schedules to plan when and how often to execute operations such as mappings and process flows that you deploy through Warehouse Builder.

To create a scheduler:

1. Right-click the **Schedules** 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 location list. Click **Edit** to type in the connection information and test the connection.

4. Click **Next.**

   The wizard displays the Summary page. Verify the name and status of the new Scheduler 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 schedule.

Example: Using Process Flows to Transfer Remote Files

Scenario

Developers at your company designed mappings that extract, transform, and load data. The source data for the mapping resides on a server separate from the server that performs the ETL processing. You would like to create logic that transfers the files from the remote computer and triggers the dependent mappings.

Solution

In Warehouse Builder, you can design a process flow that executes file transfer protocol (FTP) commands and then starts a mapping. For the process flow to be valid, the FTP commands must involve transferring data either from or to the server with the Runtime Service installed. To move data between two computers, neither of which host the Runtime Service, first transfer the data to the Runtime Service host computer and then transfer the data to the second computer.

You can design the process flow to start different activities depending upon the success or failure of the FTP commands.

Case Study

This case study describes how to transfer files from one computer to another and start a dependent mapping. The case study provides examples of all the necessary servers, files, and user accounts.

- **Data host computer:** For the computer hosting the source data, you need a user name and password, host name, and the directory containing the data. In this case study, the computer hosting the data is a UNIX server named `salessrv1`. The source data is a flat file named `salesdata.txt` located in the `/usr/stage` directory.
- **Runtime Service host computer:** In this case study, Warehouse Builder and the Runtime Service are installed on a computer called `local` with a Windows operating system. `Local` executes the mapping and the process flow.
- **Mapping:** This case study assumes there is a mapping called `salesresults` that uses a copy of `salesdata.txt` stored on `local` at `c:\temp` as its source.
- **FTP Commands:** This case study illustrates the use of a few basic FTP commands on the Windows operating system.

Your objective is to create logic that ensures the flat file on `salessrv1` is copied to the `local` computer and then trigger the execution of the `salesresults` mapping.

To transfer files and start a dependent mapping, refer to the following sections:

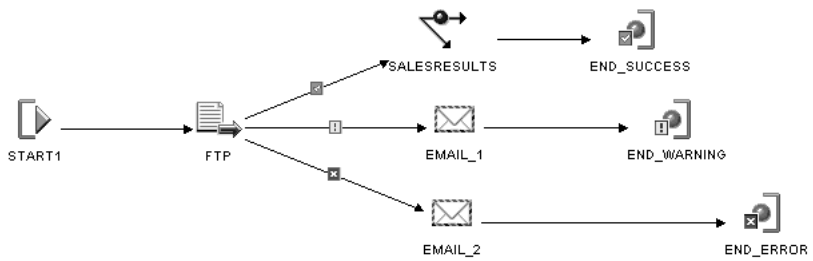
1. "Defining Locations" on page 20-34.
2. "Creating the Process Flow" on page 20-29
3. "Setting Parameters for the FTP Activity" on page 20-30
4. "Configuring the FTP Activity" on page 20-33
5. "Registering the Process Flow for Deployment" on page 20-33

After you complete the instructions in the above sections, you can run the process flow.

Creating the Process Flow

Use the Process Flow Editor to create a process flow with an FTP activity that transitions to the `salesresults` mapping on the condition of success. Your process flow should appear similar to Figure 20–13.

Figure 20–13 Process Flow with FTP Transitioning to a Mapping



This screenshot displays the process flow with FTP Transitioning to a mapping. The following objects are ordered from left to right: Start object and FTP activity. To the right of the FTP activity, ordered from top to bottom, are the following: a mapping activity SALESRESULTS, an Email activity EMAIL_1, and another Email activity EMAIL_2. To the right of these, ordered from top to bottom are the following activities: END_SUCCESS, END_WARNING, and END_ERROR.

The FTP activity splits into three paths, based on the result of the activity. Arrows connect the FTP activity to SALESRESULTS, EMAIL_1, and EMAIL_2. SALESRESULTS is connected to END_SUCCESS, EMAIL_1 to END_WARNING, and EMAIL_2 to END_ERROR.

****

Setting Parameters for the FTP Activity

This section describes how to specify the commands for transferring data from the remote server `salessrv1,` to the `local` computer. You specify the FTP parameters by typing values for the FTP activity parameters on the Activity View as displayed in Figure 20–14.

Warehouse Builder offers you flexibility on how you specify the FTP commands. Choose one of the following methods:

- **Method 1: Write a script in Warehouse Builder:** Choose this method when you want to maintain the script in Warehouse Builder and/or when password security to servers is a requirement.

  For this method, write or copy and paste the script into the Value column of the SCRIPT parameter. In the COMMAND parameter, type the path to the FTP executable such as `c:\winnt\system32\ftp.exe.` Also, type the `Task.Input` variable into the Value column of the PARAMETER_LIST parameter.

- **Method 2: Call a script maintained 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, type the appropriate command in PARAMETER_LIST to direct Warehouse Builder to the file. For a Windows operating system, type 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, type the following: `?"-s:c:\temp\move.ftp"?`

  Leave the SCRIPT parameter blank for this method.

Example: Writing a Script in Warehouse Builder for the FTP Activity

The following example illustrates Method 1 described above. It relies on a script and the use of substitution variables. The script navigates to the correct directory on `salessrv1` and the substitution variables are used for security and convenience.

This example assumes a Windows operating system. For other operating systems, issue the appropriate equivalent commands.

To define a script within the FTP activity:

1. Select the FTP activity on the canvas to view and edit activity parameters in the Available Objects tab of the Explorer panel in the Process Flow Editor.

2. For the COMMAND parameter, type the path to the FTP executable in the column labeled Value. If necessary, use the scroll bar to scroll to the right and reveal the column labeled Value.

   For windows operating systems, the FTP executable is often stored at `c:\winnt\system32\ftp.exe`.

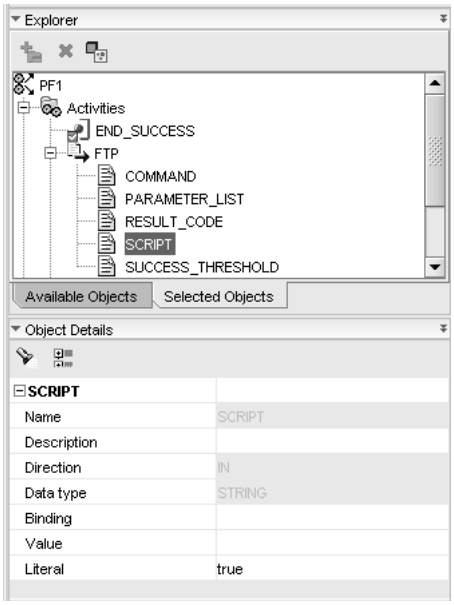
3. For the PARAMETER_LIST parameter, type the `Task.Input` variable.

   When defining a script in Warehouse Builder and using Windows FTP, you must type `?"-s:${Task.Input}"?` into PARAMETER_LIST.

   For UNIX, type `?"${Task.Input}"?`.

4. Navigate and highlight the SCRIPT parameter. Your Available Objects tab should display similar to Figure 20–14.

Figure 20–14 Activity View for FTP Activity Using a Script



This screenshot shows the FTP Activity in the Explorer panel. The SCRIPT parameter is currently selected in the Explorer panel.

Below the Explorer panel is the Object Details panel that contains details of the FTP activity parameters. The Object Details panel contains the following rows for the SCRIPT parameter: Name, Description, Direction, Data type, Binding, Value, and Literal.

****

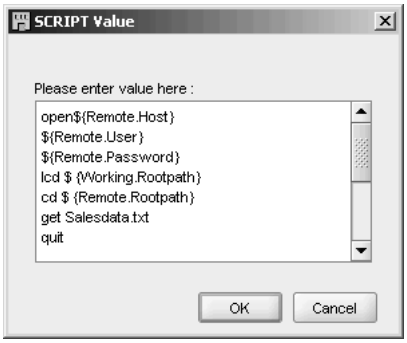
5. Click the Ellipses displayed to the right of the Value field displayed in the Object Details panel.

   Warehouse Builder displays the SCRIPT Value editor. Write or copy and paste FTP commands into the editor.

   Figure 20–14 shows a script that opens a connection to the remote host, changes the directory to the local computer, changes the directory to the remote host, transfers the file, and closes the connection.

Notice that the script in Figure 20–15 includes `${Remote.User}` and `${Remote.Password}`. These are substitution variables. Refer to "Using Substitution Variables" on page 20-32 for more details.

Figure 20–15 SCRIPT Value Editor Using Substitution Variables



This screenshot shows the SCRIPT Value dialog box. In the "Please enter value here:" field some values are displayed. At the lower right-hand corner of the dialog box are two buttons, ordered from left to right, OK and Cancel.

****

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 as described in "Defining Locations" on page 20-34.

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 provided for the associated location.

Table 20–6 lists the substitute variables you can provide for the FTP activity. `Working` refers to the computer hosting the Runtime Service, the *local* computer 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. For more information, see "Configuring the FTP Activity" on page 20-33.

Table 20–6 Substitute Variables for the FTP Activity

Variable	Value
${Working.RootPath}	The root path value for the location of the Runtime Service host.
${Remote.Host}	The host value for the location involved in transferring data to or from the Runtime Service host.
${Remote.User}	The user value for the location involved in transferring data to or from the Runtime Service host.

Table 20–6 (Cont.) Substitute Variables for the FTP Activity

Variable	Value
${Remote.Password}	The password value for the location involved in transferring data to or from the Runtime Service host.
${Remote.RootPath}	The root path value for the location involved in transferring data to or from the Runtime Service host.

Configuring the FTP Activity

As part of configuring the complete process flow, configure the FTP activity.

To configure the FTP Activity:

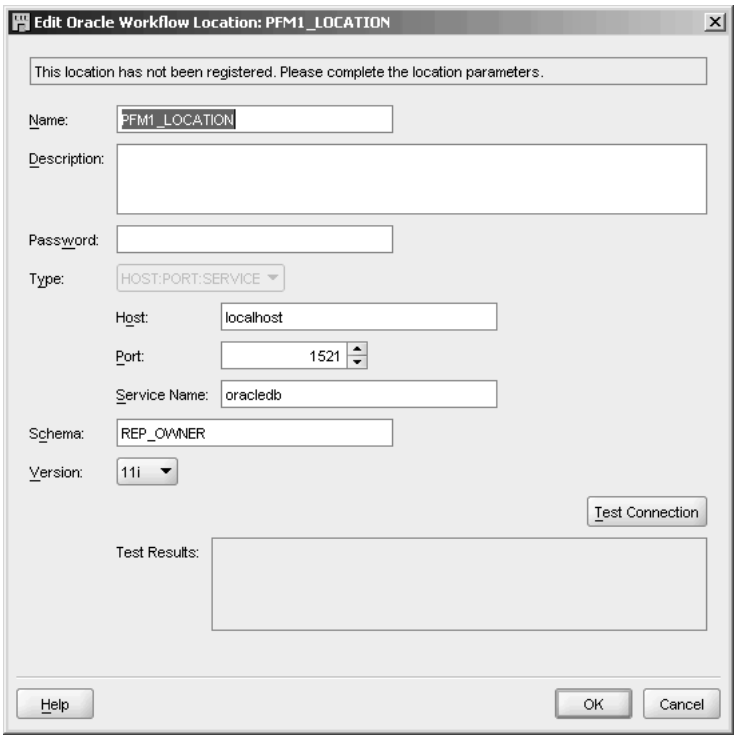
1. Right-click the process flow on the navigation tree and select **Configure.**
2. Expand the FTP activity and the Path Settings. Warehouse Builder displays the configuration settings.
3. Set Remote Location to REMOTE_LOCATION and Working Location to LOCAL_LOCATION.
4. Click to select the **Use Return as Status**. This ensures that the process flow uses the FTP return codes for determining which outgoing transition to activate. For the process flow in this case study, shown in Figure 20–13 on page 20-29, if FTP returns a success value of 1, the process flow continues down the success transition and executes the `salesresults` mapping.

Registering the Process Flow for Deployment

After you complete these instructions, you can deploy and run the process flow. To deploy the process flow, start the Deployment Manager by right-clicking and selecting **Deploy** from either the process flow module or package on the navigation tree. The Deployment Manager prompts you to register the REMOTE_LOCATION and the LOCAL_LOCATION.

Figure 20–16 shows the registration information for the REMOTE_LOCATION. For the LOCAL_FILES, only the root path is required.

Figure 20–16 Example Location Registration Information



This screenshot displays the Edit Oracle Workflow Location dialog box. It contains the following fields, ordered from top to bottom:

- Name
- Description
- Password
- Type

  This contains the fields Host, Port, and Service Name.
- Schema
- Version

Below these fields, on the right-hand side, is the Test Connection button. Below this button is the Test Results area. At the lower left-hand corner of the dialog box is the Help button. At the lower right-hand corner of the window, the following buttons, ordered from left to right: Ok and Cancel.

****

Now you can run the process flow.

Defining Locations

Locations are logical representations of the various data sources and destinations in the warehouse environment. In this scenario, the locations are the logical representations of the host and path name information required to access a flat file. Warehouse Builder requires these definitions for deploying and running the process flow. When you deploy the process flow, Warehouse Builder prompts you to type the host and path name information associated with each location. You must define locations for each computer involved in the data transfer.

To define locations, right-click the appropriate Locations node in the Connection Explorer and select **New**. For `salessrv1`, right-click Files under the Locations node and create a location named `REMOTE_FILES`. Repeat the step for `local` and create the location `LOCAL_FILES`.

21

Activities in Process Flows

After you design mappings that define the operations for moving data from source to target, you can create and define process flows. Use process flows to interrelate mappings and activities external to Warehouse Builder. For example, email, FTP commands, and operating system executables.

Using Activities in Process Flows

Use this section as a reference for all the activities. This section begins by categorizing activities by type. For detailed descriptions of each activity, see the alphabetical listing in the remainder of this section.

Activities that Represent Objects

Table 21–1 lists the activities that represent objects you previously created in Warehouse Builder. 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 21–1 Activities that Represent Objects

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 activities that represent design objects, the process flow evaluates each of these activities as a separate transaction. 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 through the Process Flow Editor" on page 22-13.

Utility Activities

Table 21–2 lists each utility activity and shows the associated icon.

Table 21–2 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.
	Manual	Halts a process flow and requires manual intervention to resume the process flow.
	Notification	Sends 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.
	Wait	Delays the progress of the process flow by a specified amount of time.

Control Activities

Table 21–3 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 21–3 Control Activities

Icon	Activity	Brief Description	Incoming Transitions	Outgoing Transitions
	AND	Specifies the completion of all incoming activities before starting another activity.	Two 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.	One or more allowed	Not allowed
	End (with errors)	Designates a path as ending in errors.	One or more allowed	Not allowed
	End (with warnings)	Designates a path as ending with warnings.	One or more allowed	Not allowed

Table 21–3 (Cont.) Control Activities

Icon	Activity	Brief Description	Incoming Transitions	Outgoing Transitions
	End Loop	Defines the end of a For Loop or While Loop	One or more allowed	One to For Loop or While Loop only
	For Loop	Use this activity with an End Loop to define constructs that repeat.	One from End Loop required plus more from other activities	One Loop condition and one Exit required
	FORK	Starts two or more activities after completing an activity.	One or more allowed	Two or more unconditional transitions only
	OR	Starts an activity after the completion of any of two or more specified activities.	Two or more allowed	One unconditional transition only
	Route	Defines exclusive OR and if-then-else scenarios.		
	While Loop	Run other activities while a condition is true.	One from End Loop required plus more from other activities	One Loop condition and one Exit required

OS Activities

Table 21–4 lists the OS activities that can be initiated by a process flow.

Table 21–4 OS Activities

Icon	Activity	Brief Description
	FTP	Starts a file transfer protocol command during a process flow. For example, use the FTP activity to move data files to the computer where a mapping runs.
	Sqlplus	Runs a SQL*Plus script in a process flow.
	User Defined	Represents an activity that is not predefined and enables you to incorporate it into a process flow.

Because it is not desirable to allow a user have complete control over OS activities, Warehouse Builder allows you to determine which OS activities can be initiated by a process flow. This is primarily achieved by constraining the user's ability to execute operating system commands either by granting or revoking direct execution or by mandating that execution be performed through a third party, as described in Setting a Security Constraint. Further access control can be achieved using a proxy command and parameters, which can be used to secure all executions.

This security feature is controlled by setting properties in the `Runtime.properties` file in the *`$owb_home`*`/owb/bin/admin` directory. This file contains Control Center property values that run the Control Center service. This file is set to read-only at Control Center service startup. If you make changes to the file, then you must restart the Control Center service for the changes to take effect.

Setting a Security Constraint

By default, `security_constraint` for the OS activity commands are set to DISABLED:

```
property.RuntimePlatform.0.NativeExecution.FTP.security_constraint = DISABLED
property.RuntimePlatform.0.NativeExecution.Shell.security_constraint = DISABLED
property.RuntimePlatform.0.NativeExecution.SQLPlus.security_constraint = DISABLED
```

To enable an OS activity, you must set `security_constraint` to `NATIVE_JAVA` or `Scheduler`.

```
property.RuntimePlatform.0.NativeExecution.FTP.security_constraint = NATIVE_JAVA
property.RuntimePlatform.0.NativeExecution.Shell.security_constraint = NATIVE_JAVA
property.RuntimePlatform.0.NativeExecution.SQLPlus.security_constraint = NATIVE_
JAVA
```

`NATIVE_JAVA` allows direct execution by the Control Center service and `SCHEDULER` forces execution through `DBMS_SCHEDULER`.

Setting a Proxy Command and Parameters

For each activity type, `USER DEFINED (Shell)`, `FTP`, and `SQLPlus`, there are two properties. The `proxy_command` property and the `proxy_parameter_list` property (optional).

If a proxy command is specified, then that command is run instead of the user's specified command and parameters. The user specified command and parameters are passed as parameters to the proxy command following the proxy parameters. The proxy command then becomes the context in which the user's command is run.

The `proxy_command` property allows the proxy command to be specified.

To set a proxy command for the activities, set the proxy command as well as the proxy parameter list (optional) using the following command:

```
property.RuntimePlatform.0.NativeExecution.FTP.proxy_command
property.RuntimePlatform.0.NativeExecution.FTP.proxy_parameter_list
property.RuntimePlatform.0.NativeExecution.Shell.proxy_command
property.RuntimePlatform.0.NativeExecution.Shell.proxy_parameter_list
property.RuntimePlatform.0.NativeExecution.SQLPlus.proxy_command
property.RuntimePlatform.0.NativeExecution.SQLPlus.proxy_parameter_list
```

For example, to set a proxy command for Shell:

```
property.RuntimePlatform.0.NativeExecution.Shell.proxy_command = /bin/proxy_sh
property.RuntimePlatform.0.NativeExecution.Shell.proxy_parameter_list = ?-v?-n?
```

> **Note:** Ideally, only the Warehouse Builder administrator must have the rights to modify the `Runtime.properties` file. The users should be granted read-only permission.

AND

Use the AND activity to specify the completion of two or more activities before resuming the process flow.

This image displays the icon for the operator described in the surrounding text.

****

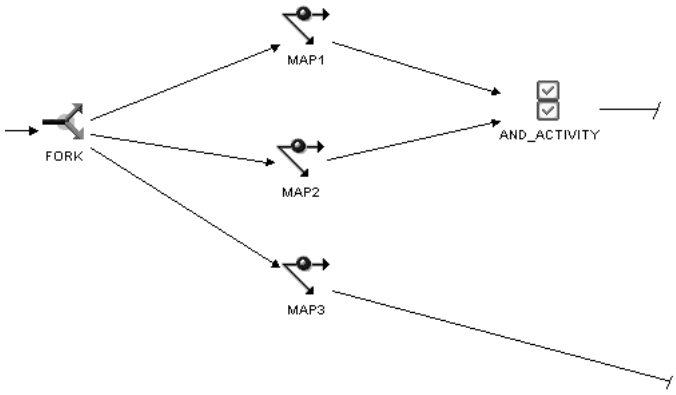
The AND activity can have two or more incoming transitions. To correctly design process flows with an AND activity, you must place a FORK activity upstream of the

AND. Also, the number of transitions going into the AND activity 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 activity returns 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 activity 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 21–1 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 three outgoing transitions while AND_ACTIVITY has two incoming transitions. The process flow would also be valid if the transition and activities associated with MAP3 were deleted.

Figure 21–1 AND Activity in a Process Flow



This image displays the 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

Use the Assign activity to assign a value to a variable. For example, use this activity to initialize a variable back to zero.

This image displays the icon for the operator described in the surrounding text.

****

Table 21–5 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.

This image displays the icon for the operator described in the surrounding text.

****

Table 21–6 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
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 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.

This image displays the icon for the operator described in the surrounding text.

****

Table 21–7 lists the parameters you set for the email activity.

Table 21–7 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 addresses or mailing lists that receive the process flow notification. Use a comma or a semi-colon to separate multiple email addresses.
CC_Address	The email addresses or mailing lists that receive a copy of the process flow notification. Use a comma or a semi-colon to separate multiple email addresses.

Table 21–7 (Cont.) Email Activity Parameters

Parameter	Description
BCC_Address	The email addresses or mailing lists 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, 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@example.com

Jack Emp<jack.emp@example.com>

Jack Emp[jack.emp@example.com]

Jack Emp[jack.emp@example.com],Jill Emp[jill.emp@example.com]

Jack Emp[jack.emp@example.com];Jill Emp[jill.emp@example.com]
```

To execute a process flow with an email activity, you may need to access different host machines and ports. New security measures implemented in Oracle Database 11*g* Release 1 restrict access to hosts and ports. You must explicitly grant access to hosts and ports that the email activity accesses using the `DBMS_NETWORK_ACL_ADMIN` package.

For example, the user `OWBSYS` needs to send an email through the mail server mail.example.com using port 25. The DBA must perform the following steps:

1. Create an Access Control List (ACL) for the user `OWBSYS` using the following command:

   ```
   EXECUTE DBMS_NETWORK_ACL_ADMIN.CREATE_ACL
           ('acl_for_owb_cc.xml','ACL for Control Center','OWBSYS','CONNECT');
   ```

   The ACL has no access control effect unless it is assigned to network target.

2. Assign the Access Control List (ACL) to a network host, and optionally specify a TCP port range. Use the following command:

   ```
   EXECUTE DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL
           ('acl_for_owb_cc.xml','mail.example.com',25)
   ```

3. Commit the changes made using the COMMIT command.

End

Every path in the process flow must terminate in an End activity.

This image displays the icon for the operator described in the surrounding text.

****

When you first create a process flow, a success type End activity is included by default. Use end types to indicate the type of logic contained in a path. Because a given activity such as a mapping has three possible outcomes, the editor includes three ending types, as shown in Table 21–8. You can use these ending types to design error handling logic for the process flow.

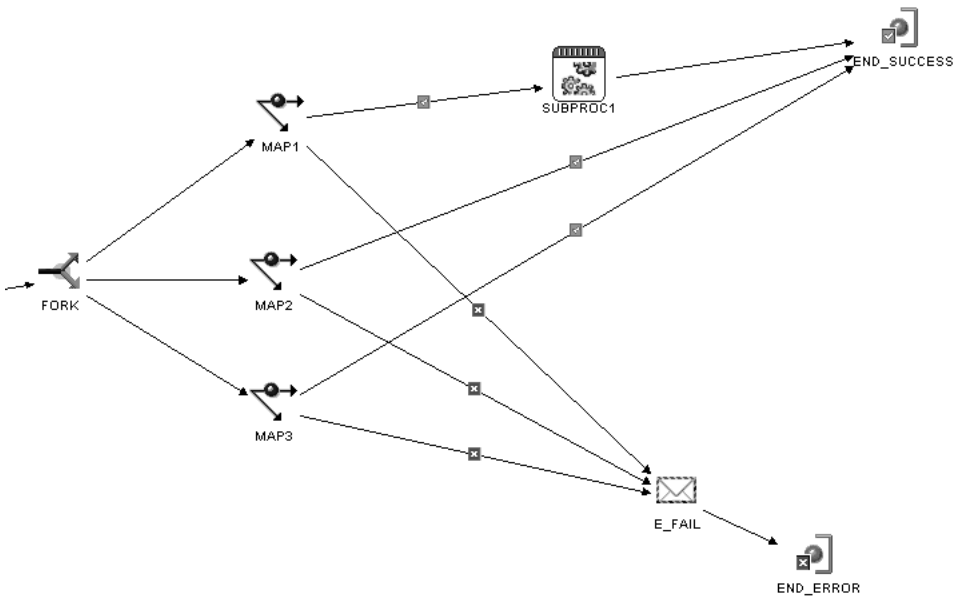
Table 21–8 Types of End Activities

Icon	End Type	Description
	Success	Indicates that the path or paths contain logic dependent on the successful completion of an upstream activity.
	Warning	Indicates that the path or paths contain logic dependent on an upstream activity completing with warnings.
	Error	Indicates that the path or paths contain logic dependent on 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 21–2, END_SUCCESS has three incoming transitions, each dependent on 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 21–2 End Activities in a Process Flow



This image displays the 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 different types of end activity.

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.

   Warehouse Builder does not allow you to select ending types already present in the process flow.

2. Click **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.

This image displays the icon for the operator described in the surrounding text.

****

The End Loop activity must have a single unconditional outgoing transition to its For Loop or While Loop activity. All the flows that are 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 running the next activity. In the Activities panel, type the name of the file.

This image displays the icon for the operator described in the surrounding text.

****

The File Exists activity checks only once. If the file exists, then the process flow proceeds with the success transition. If the file does not exist, then the process flow proceeds 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 a fully qualified file name, a directory name, or a semicolon 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 returns EXISTS. If none of the paths exist, then the activity returns MISSING. If some paths exist, then the activity returns SOME_EXIST.

FORK

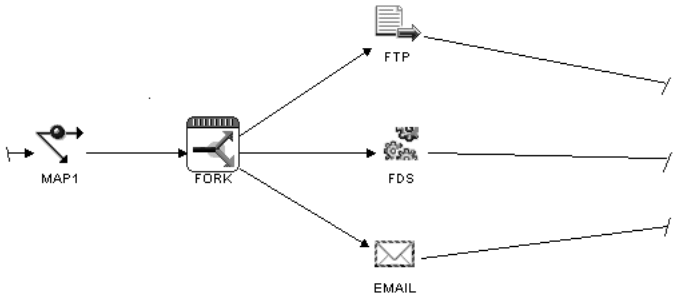
Use the FORK activity to start multiple, concurrent activities after the completion of an activity.

This image displays the icon for the operator described in the surrounding text.

****

You can assign multiple incoming transitions to a FORK activity. The FORK activity is the only activity that enables you to assign multiple unconditional outgoing transitions for parallel process. For example, in Figure 21–3, the process flow carry out the activities named FTP, FDS, and EMAIL in parallel after completing MAP1.

Figure 21–3 FORK Activity Ensures Parallel Process

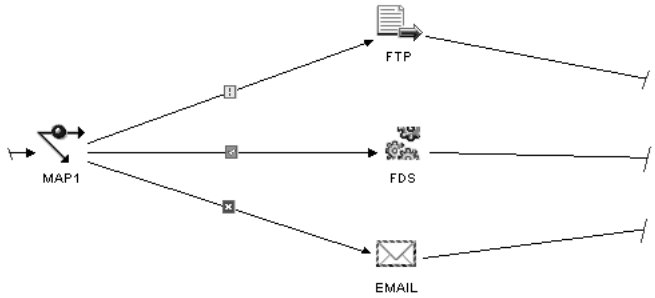


This image displays the FORK Activity.

****

Figure 21–4 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 21–4 Absence of FORK Activity Results in Conditional Process



This image displays the absence of FORK Activity which 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 run activities you include in the loop and then exit and resume the process flow.

This image displays the icon for the operator described in the surrounding text.

****

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 Condition in the object details.

Table 21–9 For Loop Activity Parameters

Parameter	Description
Condition	An expression which when evaluated to true runs the loop transition else the exit transition.
Variable	Bound to a variable or parameter, its value is incremented every iteration.
Initial_Value	The initial value of the variable on 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 computers, neither of which host the Control Center Service, first transfer the data to the Control Center Service host computer and then transfer the data to the second computer.

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 21–10 lists the parameters you set for the FTP activity when writing the script within Warehouse Builder.

Table 21–10 FTP Activity Parameters for Script in Warehouse Builder

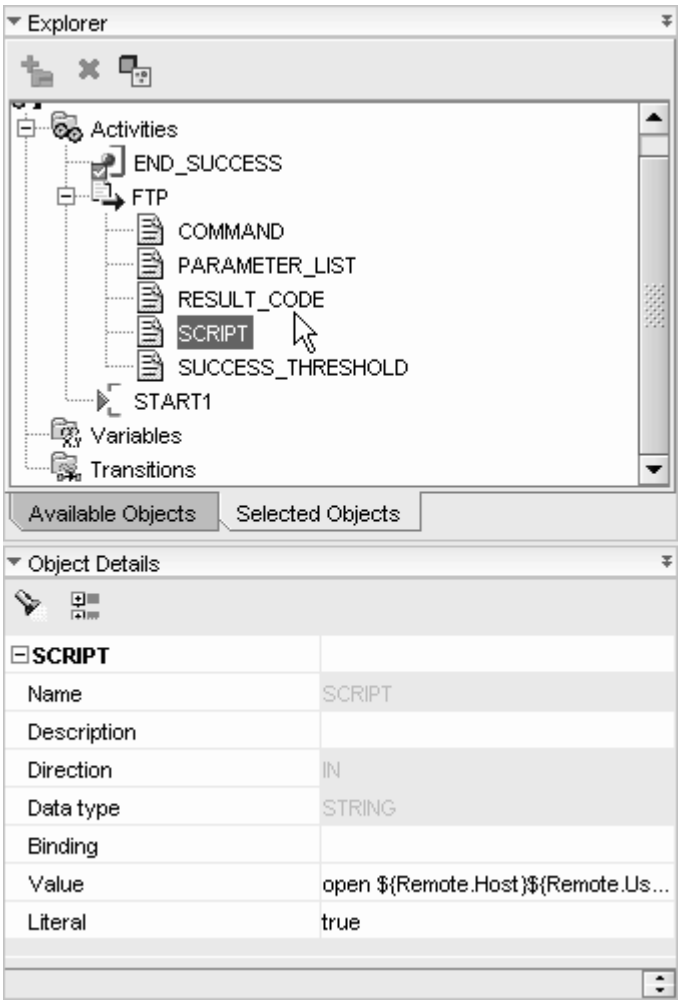
Parameter	Description
COMMAND	Enter the path to file transfer protocol command such as `c:\WINNT\System32\ftp.exe` for Windows operating systems.

Table 21–10 (Cont.) FTP Activity Parameters for Script in Warehouse Builder

Parameter	Description
PARAMETER_ LIST	This is a list of parameters 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 preceded 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 passed on as standard input. Note: The -s parameter is set for the Windows FTP command because it cannot be used 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 21–5 shows an example of the FTP activity parameter settings for calling a script written within Warehouse Builder.

Figure 21–5 Activity View for FTP Activity Using a Script

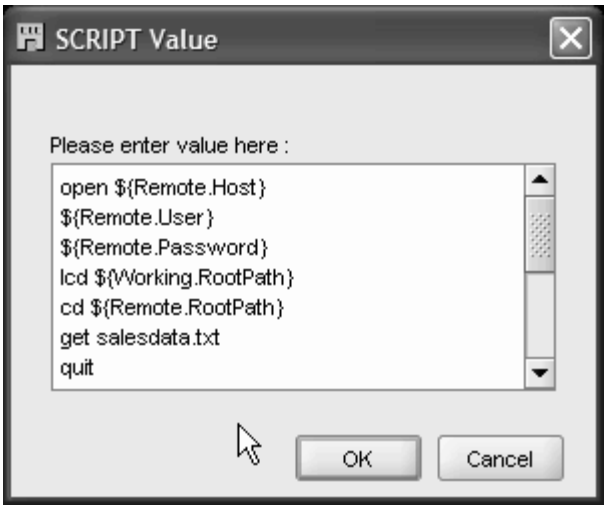


This image displays the activity view for FTP Activity using script.

****

Figure 21–6 shows the first line of the script in the VALUE column of the SCRIPT parameter. To view the entire script, click the Colon button to the right of the **Value** column. 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 21–6.

Figure 21–6 SCRIPT Value Editor Using Substitution Variables



This image displays the script value editor using substitution variables.

****

Notice that the script inFigure 21–6 includes ${Remote.User} and ${Remote.Password}. These are substitution variables. See "Using Substitution Variables" on page 21-14 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 an FTP activity is run with substitution variables for the server passwords, it resolves the variable to the secure password you entered for the associated location.

Table 21–11 lists the substitute variables you can enter for the FTP activity. *Working* refers to the computer hosting the Control Center Service, the *local* computer 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 Reference" on page 22-41.

Table 21–11 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.

Table 21–11 (Cont.) Substitute Variables for the FTP Activity

Variable	Value
${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 OS 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 21–12 lists the parameters you set for the FTP activity when the FTP script resides in a file on your system.

Table 21–12 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 the Manual activity to halt a process flow.

This image displays the icon for the operator described in the surrounding text.

****

Once the process flow halts, a user must intervene via 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 it 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 21–13 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	Select 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.

This image displays the icon for the operator described in the surrounding text.

****

You can assign multiple incoming transitions to a mapping activity. For outgoing transitions, assign one 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 runs in row-based mode. The original mapping retains set-based mode as its operating mode. If you want to change the properties for the underlying mapping, see "Configuring Mappings Reference" on page 22-30.

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 21–14 and Table 21–15 list the different mapping parameters in PL/SQL and SQL*Loader.

Table 21–14 lists the PL/SQL mapping parameters.

Table 21–14 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 21–15 lists the SQL*Loader mapping parameters.

Table 21–15 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 image displays the icon for the operator described in the surrounding text.

****

The Notification activity enables you to design a process to restart or recover ETL processes.This activity works in conjunction with Oracle Workflow. To implement notifications 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 21–16. 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 21–16 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 start an activity based on the completion of one or multiple number of upstream activities. You can assign multiple incoming transitions and only one unconditional outgoing transition to an OR activity.

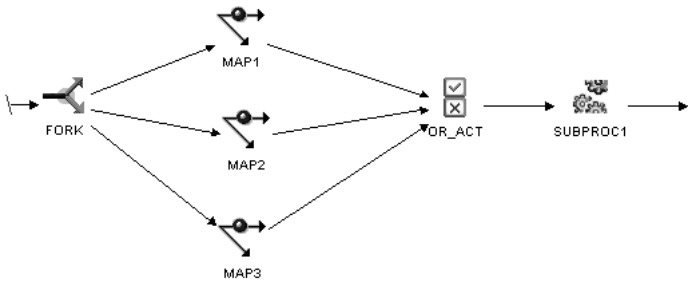
The OR activity has similar semantics to the AND activity except that the OR activity propagates the SUCCESS, WARNING, or ERROR outcome of the first upstream activity that is completed.

This image displays the icon for the operator described in the surrounding text.

****

An OR activity in a process flow ensures that downstream activities are triggered only once for each run of a process flow.

Figure 21–7 The OR activity in an process flow



This image displays the OR activity in a 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 21–7. However, this logic would start SUBPROC1 three times within the same run of a process flow. Avoid this by using an OR activity.

Route

Use the Route 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.

This image displays the icon for the operator described in the surrounding text.

****

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 the Set Status activity to interject a success, warning, or error status.

This image displays the icon for the operator described in the surrounding text.

****

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.

This image displays the icon for the operator described in the surrounding text.

****

To paste or type in a script, select the activity on the canvas, select Script in the editor explorer, and then click **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 through the Process Flow Editor" on page 22-13.

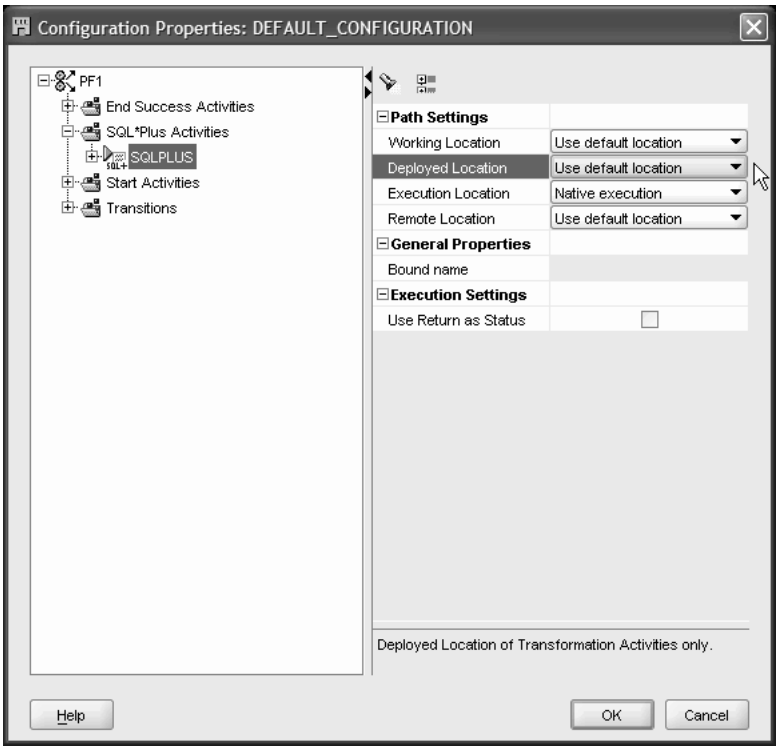
Using Activities in Process Flows

The Process Flow in SQLPlus activity is carried out by the configuration item in the Deployed Location.

To set the location that will run 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 21–8 Configuring the Deployed Location



This image shows the configuration properties for changing the Deployed Location for SQLPlus.

****

6. Under Path Settings, set the **Deployed Location** option to the location that will run 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 21–17 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 21–18 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 home directory or its location is predefined by the workspace 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.

This image displays the icon for the operator described in the surrounding text.

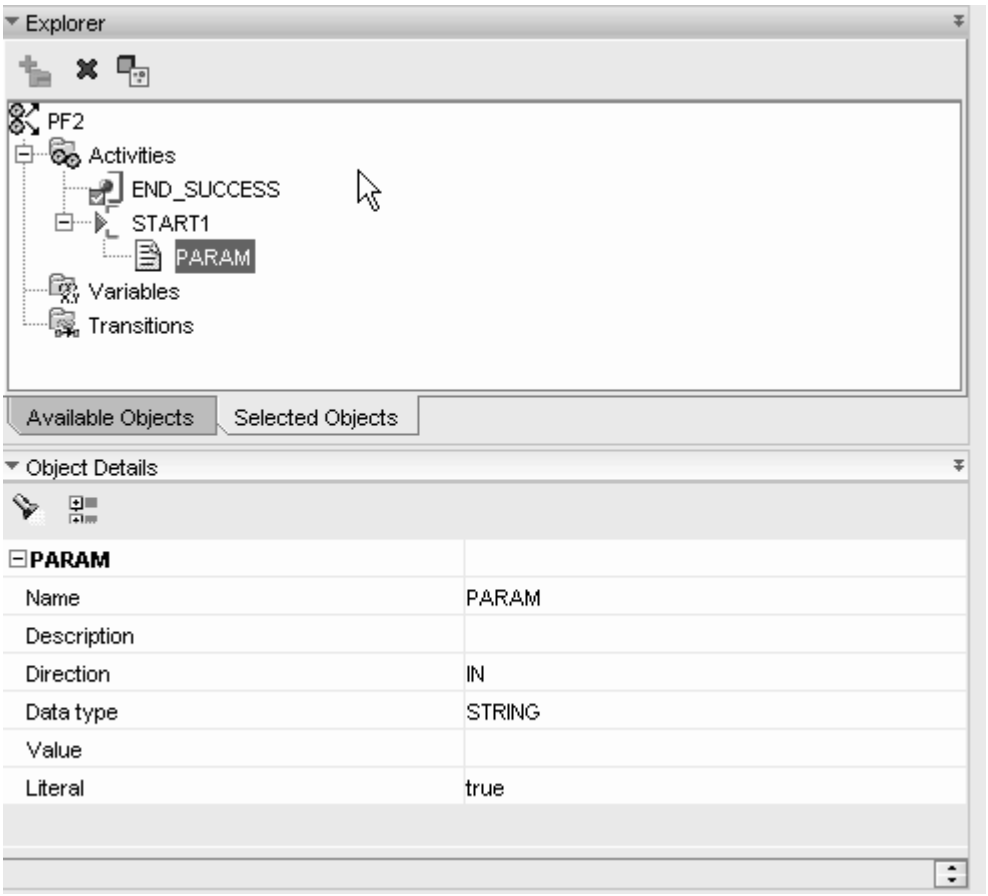
****

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 21–9 Parameter added to a Start activity



This image displays the 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. 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 start a previously created process flow. From one process flow, you can start any other process flow that is contained within the same or any other process flow package.

This image displays the icon for the operator described in the surrounding text.

****

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 one unconditional outgoing transition or up to three 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 box 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 transformation library to a process flow using the Transform activity, the Process Flow Editor displays the parameters for the transformation in the Activity panel.

This image displays the icon for the operator described in the surrounding text.

****

You can specify one or more incoming transitions to start 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 about **Use Return as Status,** see "Configuring Process Flows Reference" on page 22-41.

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.

This image displays the icon for the operator described in the surrounding text.

****

You can specify one or more incoming transitions to start 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 its return value. For more information about **Use Return as Status,** see "Configuring Process Flows Reference" on page 22-41.

Table 21–19 lists the parameters you set for the FTP activity.

Table 21–19 User Defined Process Activity Parameters

Parameter	Description
Command	The command to carry out 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 21–20.

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`.
Script	You can enter a script here or type in a file name for a script. If you type in a file name, use the ${Task.Input} variable in the parameter list to pass the file name. 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 21–20 lists the substitute variables you can enter for the FTP activity.

Table 21–20 Substitute Variables for the User Defined Process Activity

Variable	Value
${Working.Host}	The host value for the location of the Control Center Service host.

Table 21–20 (Cont.) Substitute Variables for the User Defined Process Activity

Variable	Value
${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.

This image displays the icon for the operator described in the surrounding text.

****

Table 21–21 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 run one or more activities only when a condition you define evaluates to being true.

This image displays the icon for the operator described in the surrounding text.

****

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 21–22 While Loop Activity Parameters

Parameter	Description
Condition	.Define with a LOOP or EXIT condition.

22

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
- Best Practices for Designing SQL*Loader Mappings
- Improved Performance through Partition Exchange Loading
- High Performance Data Extraction from Remote Sources
- Configuring ETL Objects
- Configuring Mappings Reference
- Configuring Process Flows Reference

Best Practices for Designing PL/SQL Mappings

This section addresses PL/SQL mapping design and includes:

- Set Based Versus Row Based Operating Modes
- About Committing Data in Warehouse Builder
- Committing Data Based on Mapping Design
- Committing Data Independently of Mapping Design
- Running Multiple Mappings Before Committing Data
- Ensuring Referential Integrity in PL/SQL Mappings

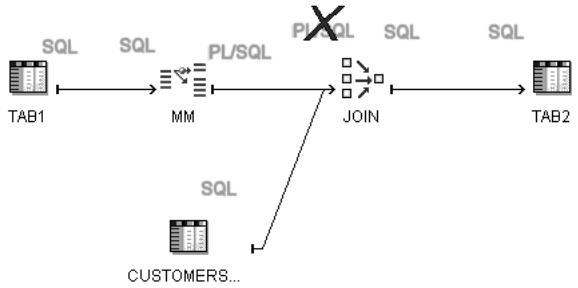
Warehouse Builder generates code for PL/SQL mappings that meet the following criteria:

- 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 22–1 is invalid because the Match-Merge operator `MM` generates PL/SQL output but the subsequent Join operator accepts SQL input only.

Figure 22–1 Mapping Violates Input Requirement for Join Operator



This graphic shows a mapping design that is invalid because the output of one operator violates the input requirement of the next downstream operator. At the top of this image are the following operators, ordered from left to right: TAB1, MM, JOIN, and TAB2. Below these operators is the operator CUSTOMERS.

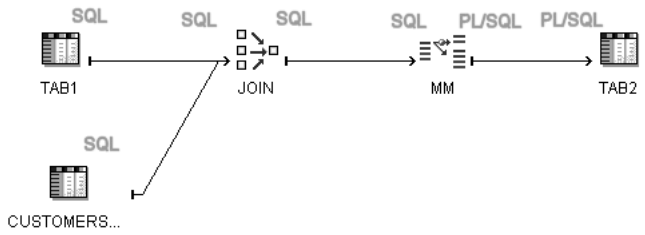
Data originates in TAB1 and flows to MM, representing the Match-Merge operator. This is depicted by an arrow from TAB1 to MM. An arrow connects MM to JOIN, which in turn is connected to TAB2. An arrow connects CUSTOMERS to JOIN.

****

To achieve the desired results for the mapping, consider joining the source tables before performing the Match-Merge or loading the results from the Match-Merge to a staging table before performing the join.

Figure 22–2 displays a mapping in which source tables are joined before the Match-Merge. Figure 22–3 displays a mapping in which the results from the Match-Merge are loaded into a staging table before performing the join.

Figure 22–2 Valid Mapping Design with Sources Joined Before Match-Merge

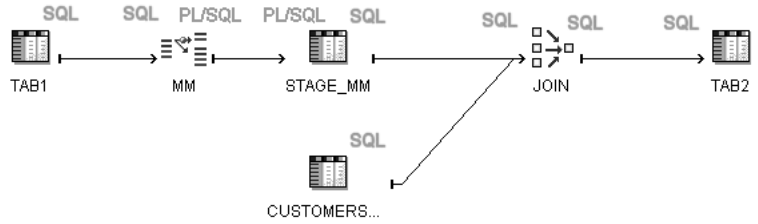


This graphic shows a mapping design that is valid.

At the top of this graphic are the following operators, from left to right: TAB1, JOIN, MM, and TAB2. Below these is the CUSTOMERS operator. There are arrows connecting TAB1 to JOIN, JOIN to MM, and MM to TAB2. There is an arrow connecting CUSTOMERS to JOIN.

****

Figure 22–3 Valid Mapping Design with Staging Table



This graphic shows the valid mapping design with Staging table.

At the top of the graphic are the following operators, ordered from left to right: TAB1, MM, STAGE_MM, JOIN, and TAB2. Below these operators is the CUSTOMERS operator. There are arrows connecting TAB1 to MM, MM to STAGE_MM, STAGE_MM to JOIN, and JOIN to TAB2. There is also an arrow connecting CUSTOMERS to JOIN.

****

Table 22–1 and Table 22–2 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 22–1 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, Dimensions, Cubes, Views, External Tables.	SQL	Yes	Yes	Yes. Part of cursor.
Target Operators: Tables, Dimensions, Cubes, Views,	SQL PL/SQL	Yes, except when loading= UPDATE and database is not 10*g* or higher.	Yes	Yes. Not part of cursor.
Flat File as source	For PL/SQL, create an 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.
Sequence as source	SQL	Yes	Yes	Yes, part of cursor.

Table 22–2 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
Aggregator	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

Table 22–2 (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
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
Match-Merge	SQL input PL/SQL output (PL/SQL input from XREF group only)	No	Yes	Yes. Not part of cursor.
Name and 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 and 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.

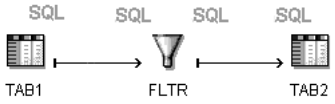
These general rules are explained in the following sections.

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 of the execution error only. With set based mode, you cannot identify the rows that contain errors.

Figure 22–4 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 22–4 Simple Mapping Run in Set Based Mode



This graphic shows the simple mapping run in Set based mode. To the left is a table TAB1. In the center is FLTR, representing a Filter operator. To the right is a table TAB2. The output of TAB1 is SQL and is provided as input to the Filter. The output of the filter is SQL and is provided as an input to a table TAB2. This is represented by an arrow from TAB1 to FLTR and then from FLTR to TAB2.

****

To correctly design a mapping for the set based mode, avoid operators that require row by row processing such as Match-Merge and Name and 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 displays an error when you execute the package in set based mode.

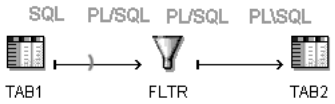
For target operators in a mapping, the loading types INSERT/UPDATE and UPDATE/INSERT are always valid for set based mode. Warehouse Builder supports UPDATE loading in set based mode only when the Oracle Database is 10*g* or higher. Warehouse Builder also supports the loading type DELETE in set based mode. For a complete listing of how Warehouse Builder handles operators in set based mappings, see Table 22–2 on page 22-3.

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.

Figure 22–5 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 22–5 Simple Mapping Run in Row Based Mode



This graphic shows a simple mapping run in a row based mode. To the left is a table TAB1. In the center is FLTR, representing a Filter operator. To the right is a table TAB2. The output of TAB1 is SQL and is provided as input to the Filter. This is represented by an arrow connecting TAB1 to FLTR and another arrow connecting FLTR to TAB2.

During data transfer, the input from TAB1 is converted from SQL to PL/SQL. The output of the filter is PL/SQL and is provided as an input to a table TAB2.

****

If the mapping includes any SQL based operators that 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

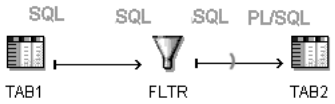
For the preceding operators to be included in the cursor, do not directly precede it by an operator that generates PL/SQL code. In other words, you cannot run the mapping in row-based mode if it contains a Transformation implemented as procedure, a Flat File used as a source, a Match-Merge, or Name and 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 loading the data, which is where errors are likely to occur.

Table 22–6 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 22–6 Simple Mapping Run in Row Based (Target Only) Mode



This graphic shows a simple mapping run in row based (Target only) mode. To the left is a table TAB1. To the center is FLTR, representing a Filter operator. To the right is a table TAB2. Arrows connect TAB1 to FLTR and FLTR to TAB2.

The output of TAB1 is SQL and is provided as input to the Filter. The output of the filter is SQL. During data transfer, the output from the filter is converted from SQL to PL/SQL and is provided as an input to a table TAB2.

****

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 22-8.

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 22-11.

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 22-8.

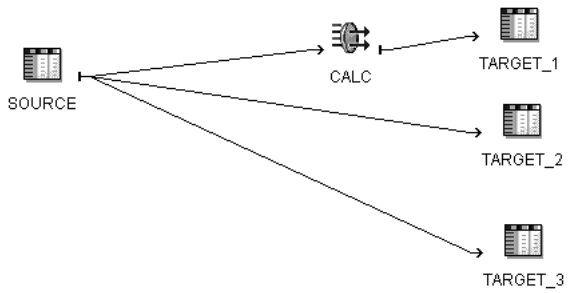
Manual: Select manual commit control for PL/SQL mappings when you want to interject complex business logic, perform validations, or run other mappings before committing data. For examples, see "Embedding Commit Logic into the Mapping" on page 22-10 and "Committing Data Independently of Mapping Design" on page 22-11.

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 22–7 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 affected targets at the same time. If this relationship is not maintained when you run the mapping again, then the data can become inaccurate and possibly unusable.

Figure 22–7 Mapping with Multiple Targets Dependent on One Source



This graphic shows a mapping with multiple targets dependent on one source. On the left is the Table operator SOURCE. In the center, is the Transformation operator CALC. To the right are the following three Table operators, ordered from top to bottom: TARGET_1, ATRGET_2, And TARGET_3. There is an arrow from SOURCE to CALC and from CAL to TARGET_1. There are also separate arrows from SOURCE to TARGET_2 and TARGET_3.

****

If the number of rows from the source table 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 update/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 22–3 shows the valid combinations you can select.

Table 22–3 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 22–7 on page 22-8. 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 22-11.
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-Mapping Process or Post-Mapping Process 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 incorrect 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 starting 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 22–4. Warehouse Builder uses the default setting for any optional parameters that you do not specify.

Table 22–4 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 alone, 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 a separate mapping to load cubes.

   These instructions are not valid for SQL*Loader and ABAP mappings.

2. Configure both mappings with the Commit Control property set to Manual.

   In the Project Explorer, right-click the mapping and select **Configure.** Under the Code Generation Options**,** set the Commit Control property 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 22–4. 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 mapping 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 22–4 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 through the Process Flow Editor" on page 22-13.

Committing Mappings through the Process Flow Editor

For PL/SQL mappings alone, 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 through 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 the Commit Control property set to Manual.

   In the Project Explorer, right-click the mapping and select **Configure.** Under Code Generation Options**,** set the Commit Control property 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.

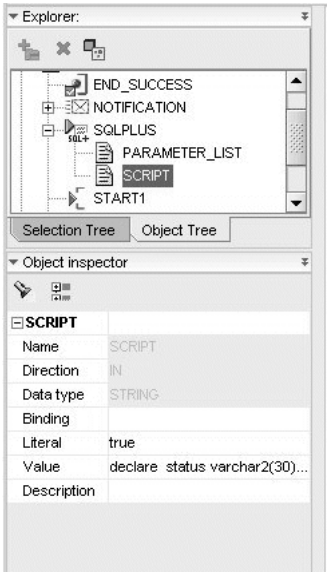
```
declare
    status VARCHAR2(30);
begin
    map1.main(status);
    if status != 'OK' then
       rollback;
    else
       map2.main(status);
       if status != 'OK' then
           rollback;
       else
           commit;
       end if;
    end if;
end;
```

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.

   Figure 22–8 displays the Explorer panel and the Object Inspector panel with SCRIPT selected.

Figure 22–8 Specifying a Script in the Sqlplus Activity



This screenshot shows a SQLPlus activity for specifying a script. To the upper half of the image is the Explorer panel. It contains two tabs, ordered from left to right: Selection Tree and Object Tree. The Object Tree is currently displayed. The SQLPLUS activity is expanded to display the parameter SCRIPT, which is currently selected. Below the Explorer panel is the Object Inspector panel. The information in this panel is in a table with the following rows, ordered from top to bottom: Name, Direction, Data type, Binding, Literal, Value, and Description. This table contains two columns.

****

7. Optionally apply a schedule to the process flow as described in "Process for Defining and Using Schedules" on page 26-2.
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) 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 22-15.
3. In the mapping properties, view the Target Load Order property by clicking the Ellipses button to the right of this 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 box to define the loading order.

For more information, see "Target Load Order" on page 16-21.

4. Ensure that the Use Target Load Ordering configuration property 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
- Using Direct Path Loading to Ensure Referential Integrity in SQL*Loader Mappings

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 22–1, 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 22–1 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 22–1, 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 22–1 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 22–5 and Table 22–6) to maintain the relationship between the master and detail records. Use the Mapping Sequence operator to generate this additional value.

Table 22–5 represents the target table containing the master records from the file in Example 22–1 on page 22-15. 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 22–5 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 22–6 represents the target table containing the detail records from the file in Example 22–1 on page 22-15. 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 22–5.

Table 22–6 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 22-20.

This procedure outlines general steps for building such a mapping. Additional detailed instructions are available at:

- Using the Import Metadata Wizard on page 4-9
- Flat File Operator on page 17-34
- Adding Operators that Bind to Workspace Objects on page 16-8
- Sequence Operator on page 17-27
- "Configuring Mappings Reference" on page 22-30

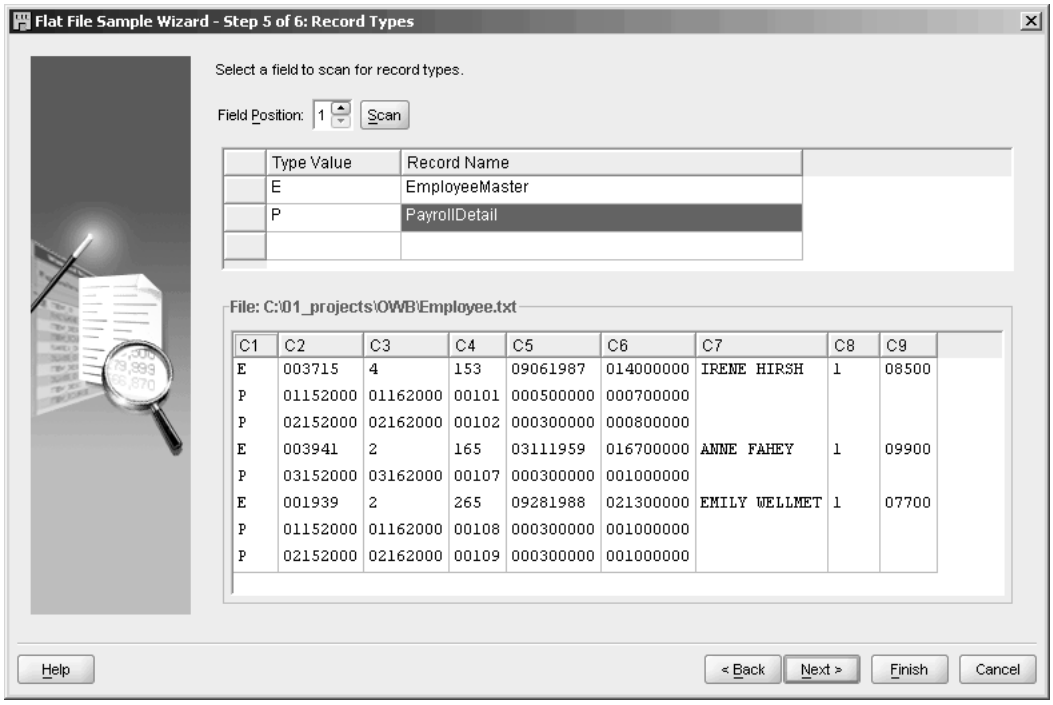
To extract from a master-detail flat file and maintain master-detail relationships, use the following steps:

1. Import and sample the 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. This makes it easier to identify those records in the future.

   Figure 22–9 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 22–9 Naming Flat File Master and Detail Record Types



This screenshot shows the Record Types page of the Flat File Sample wizard for naming flat file master and detail record type. At the top is the field called Field Position with value as 1. To the right of the Field Position field is the Scan button.

Below the Field Position field is a table which contains two columns, ordered from left to right: Type Value and Record Name. This table has two rows. Below the table is the File: box which also has information in a table, with the following column names, ordered from left to right: C1, C2, C3, C4, C5, C6, C7, C8, and C9. At the lower left-hand corner of the page is the Help button. At the lower right-hand corner of the page, are the following buttons, ordered from left to right: Back, Next, Finish, and Cancel.

****

2. Drop a Flat File operator onto the Mapping Editor canvas and specify the master-detail file from which you want to extract data.
3. Drop a Sequence operator onto the mapping canvas.
4. Drop a Table operator for the master records onto the mapping canvas.

   You can either select an existing workspace 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.

5. Drop a Table operator for the detail records onto the mapping canvas.

   You can either select an existing workspace 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.

6. Map all of the necessary flat file master fields to the master table and detail fields to the detail table.

   Figure 22–10 displays the mapping of the fields.

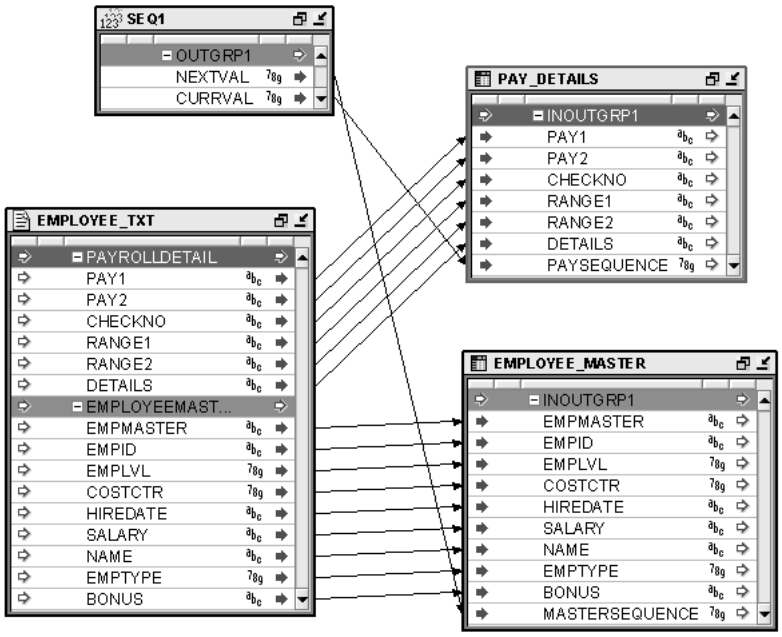
7. Map the Sequence `NEXTVAL` attribute to the additional sequence column in the master table.

   Figure 22–10 displays the mapping from the `NEXTVAL` attribute of the Sequence operator to the master table.

8. Map the Sequence `CURRVAL` attribute to the additional sequence column in the detail table.

   Figure 22–10 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 22–10 Completed Mapping from Master-Detail Flat File to Two Target Tables



This screenshot shows the completed mapping from the master detail flat file into two target tables. To the left, ordered from top to bottom, are the Sequence operator SEQ1 and the Flat File operator EMPLOYEE_TXT. To the right are two Table operators, ordered from top to bottom, PAY_DETAILS and EMPLOYEE_MASTER.

Arrows join columns in EMPLOYEE_TXT to the PAY_DETAILS and EMPLOYEE_ MASTER tables. Arrows connect the NEXTVAL column in SEQ1 to EMPLOYEE_ MASTER and the CURRVAL column in SEQ1 to PAY_DETAILS.

****

9. Configure the mapping that loads the source data into the target tables with the following parameters:

   Direct Mode: Not selected

   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 17-27).
2. Configure a mapping with the following parameters:

   Direct Mode= Not selected

   ROW=1

   ERROR ALLOWED = 0
3. Generate the code and run an SQL*Loader script.

   If the data file has errors, then the loading stops when the first error occurs.
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 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 17-27).

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 tables 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 4-9.
- For additional information on using the Flat File as a source, see "Flat File Operator" on page 17-34.
- For additional information on using Table operators, see "Adding Operators that Bind to Workspace Objects" on page 16-8.

- For additional information on using the Data Generator operator, see "Data Generator Operator" on page 17-12.
- For additional information on using the Constant operator, see "Constant Operator" on page 17-8.
- For additional information on configuring mappings, see "Configuring Mappings Reference" on page 22-30.

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 22–9 on page 22-17. This will make it easier to identify those records in the future.

2. Create a mapping that you will use to load data from the flat file source.
3. Drop a Flat File operator onto the mapping canvas and specify the master-detail file from which you want to extract data.
4. Drop a Data Generator and a Constant operator onto the mapping canvas.
5. Drop a Table operator for the master records onto the mapping canvas.

   You can either select an existing workspace 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.

6. Drop a Table operator for the detail records onto the mapping canvas.

   You can either select an existing workspace 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.

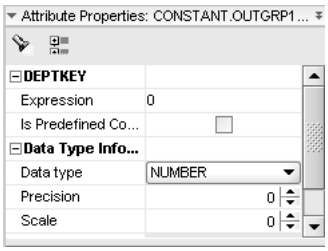
7. Map all of the necessary flat file master fields to the master table and detail fields to the detail table, as shown in Figure 22–12 on page 22-22.
8. Map the Data Generator operator's `RECNUM` attribute to the `RECNUM` columns in the master and detail tables, as shown in Figure 22–12 on page 22-22.
9. 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 22–11 shows the expression property of the constant attribute set to `'0'`. This constant marks all data rows as "just loaded".

Figure 22–11 Constant Operator Properties



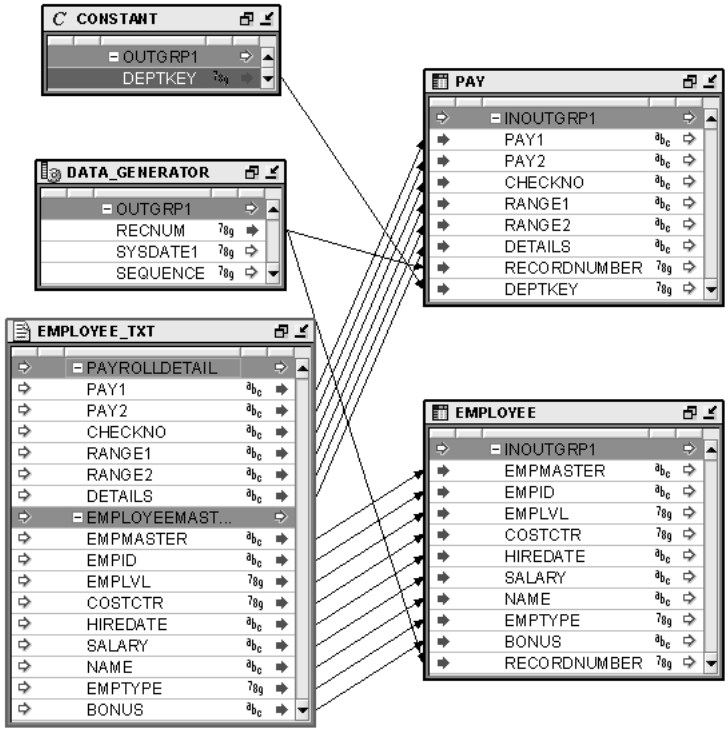
This screenshot shows the Attribute Properties panel. The Attribute Properties are available in the following rows: Expression, Is Predefined Constant, Data type, Precision, and Scale. The value for Expression is set to 0 while the value for Data type is set to NUMBER.

****

10. 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 22–12 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 22–12 Completed Mapping from Master-Detail Flat File with a Direct Path Load



This screenshot shows the completed mapping from master-detail flat file with a direct path load. To the left are a Constant operator CONSTANT, a Data Generator operator DATA_GENERATOR, and a Flat File operator EMPLOYEE_TXT. On the right are two Tables operators, PAY and EMPLOYEE.

An arrow connects CONSTANT to DEPTKEY attribute of PAY. Arrows connect the RECNUM attribute of DATA_GENERATOR to the RECORDNUMBER attribute of PAY and the RECORDNUMBER attribute of EMPLOYEE. The attributes in the group PAYROLLDETAIL of EMPLOYEE_TXT are connected to PAY using arrows. The attributes in the EMPLOYEEMASTER group of EMPLOYEE_TXT are connected to EMPLOYEE using arrows.

****

11. Configure the mapping with the following parameters:

    Direct Mode: True

    Errors Allowed: 0

    Trailing Nullcols: True (for each table)

12. 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.
13. Run the SQL*Loader script.
14. 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
- Configuring a Mapping for PEL
- Direct and Indirect PEL
- Using PEL Effectively
- Configuring Targets in a Mapping
- Restrictions for Using PEL in Warehouse Builder

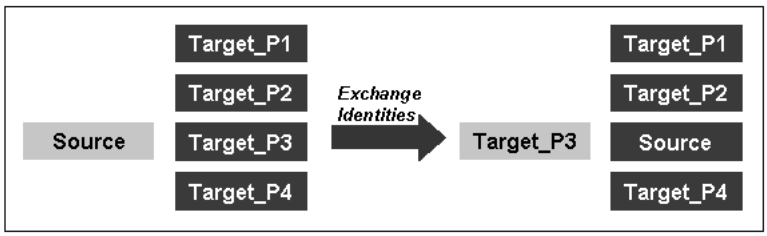
About Partition Exchange Loading

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 22–13 illustrates an example of PEL. 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 Oracle 9*i* completes the loading process without data movement.

Figure 22–13 Overview of Partition Exchange Loading



This screenshot provides an overview of Partition Exchange Loading. To the left is a rectangle named as Source. To the right are four rectangles, ordered from top to bottom: Target_P1, Target_P2, Target_P3, and Target_P4. To the right of these is an arrow, with the label Exchange Identities, pointing from left to right. To the right of the arrow is a rectangle named Target_P3. To the right of TARGET_P3 are four rectangles, ordered from top to bottom: Target_P1, Target_P2, Source, and Target_P4.

****

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 window.

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 22-26.

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.

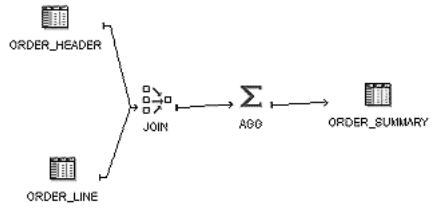
- **Indirect PEL:** By default, Warehouse Builder creates and maintains a temporary table that stages the source data before initiating the partition exchange process. For example, use Indirect PEL when the mapping includes a remote source or a join of multiple sources.
- **Direct PEL:** You design the 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 22–14 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 22–14 Mapping with Multiple Sources



This image represents a mapping that uses multiple sources. On the left are two tables ordered from top to bottom, ORDER_HEADER and ORDER_LINE. To the right of these tables, ordered from left to right, are the following: JOIN, AGG, and ORDER_SUMMARY. There are arrows from ORDER_HEADER and ORDER_LINE to JOIN. Arrows connect JOIN to AGG and then AGG to ORDER_SUMMARY.

****

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 22–14 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.

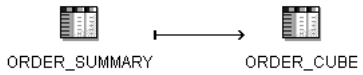
To instantly load data to a target using Direct PEL:

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 22–14 is `ORDER_SUMMARY` and should be of the same structure as the final target, `ORDER_CUBE` in Figure 22–15.

2. Create a second mapping that loads data from the staging table to the final target such as shown in Figure 22–15. Configure this mapping to use Direct PEL.

Figure 22–15 Publish_Sales_Summary Mapping



This screenshot shows the Mapping Editor canvas which contains the `ORDER_SUMMARY` table to the left and the `ORDER_CUBE` table to the right. The `ORDER_SUMMARY` table is linked to the `ORDER_CUBE` table with an arrow pointing horizontally from left to right.

****

3. Use either the Warehouse Builder Process Flow Editor or Oracle Workflow to start 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 13-15.

For example, if you select Month as the frequency of new data collection, you must 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.

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.

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')),
. . .
```

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. Set the following on the Indexes tab of the Data Object Editor:

- Select UNIQUE in the Type column.

- Select LOCAL in the Scope column.

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. In the Project Explorer, right-click the table and select **Configure**. The Configuration Properties dialog box is displayed. Select the primary or unique key in the left panel and select **Using Index** in the right panel.

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 22-28.

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.

> **See Also:** "Moving Large Volumes of Data" in the *Oracle Warehouse Builder Installation and Administration Guide* for instructions on using transportable modules

Configuring ETL Objects

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 user interface.

This chapter includes:

Configuring Mappings Reference

When you configure mappings properly, you can improve the ETL performance. Use this section as a reference for setting configuration parameters that govern how data is loaded and to optimize code for better performance.

This section includes the following topics:

Procedure to Configure Mappings

To configure physical properties for a mapping:

1. Right-click a mapping from the Project Explorer and select **Configure**.

   Warehouse Builder displays the Configuration Properties dialog box for a mapping. You can configure properties for the mapping itself and the operators within the mapping.

2. Select **Deployable** to enable the generation of a set of scripts for mapping entities marked as deployable.

   Deployable is enabled by default. If you disable it, scripts for that mapping are not generated.

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 26, "Scheduling ETL Objects".

5. Expand **Runtime Parameters** to configure your mapping for deployment.

   For a description of each runtime parameter, see "Runtime Parameters" on page 22-31.

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 22-33.

7. Go to the node for each operator in the mapping to set their physical properties.

   For information about configuring sources and targets in a mapping, see "Sources and Targets Reference" on page 22-34. To configure mappings with flat file sources and targets, see "Configuring Flat File Operators" on page 22-39.

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.

If you set the **Default Operating Mode** to row based and deselect Bulk Processing Code**,** then use the **Commit Frequency** parameter to determine the number of rows to be processed before a commit. 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 differ, then Bulk Size overrides Commit Frequency and a commit is implicitly performed for every bulk size.

Maximum Number of Errors

Use **Maximum Number of Errors** to indicate the maximum number of errors allowed while 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 22-5. You can select one of the following operating modes:

- **Set based:** A single SQL statement that inserts all data and performs all operations on the data is generated. This increases the speed of Data Manipulation Language (DML) operations. Set based mode offers optimal performance but minimal auditing details.
- **Row based:** Statements that process data row by row are generated. The select statement is a SQL cursor. All subsequent statements are PL/SQL. Because data is processed row by row, the row based operating mode has the slowest performance but offers exhaustive auditing details.
- **Row based (Target Only):** A cursor select statement is generated and attempts are made 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:** The mapping is executed in set based mode. If an error occurs, the execution fails and the mapping is started 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):** The mapping is first executed 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

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, ANSI SQL syntax is generated. Else, Oracle SQL syntax is generated.

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, data is committed 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 commits 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 22-8.

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 the following 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 22-10.
- 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 22-11.

No Commit: If you set this option, then OWB mapping does not issue a commit while the mapping executes.

Analyze Table Statements

If you select this option, code is generated for analyzing the target table after the target is loaded, if the resultant target table is double or half its original size.

If the target table is not in the same schema as the mapping and you wish to analyze the table, then you would need to grant ANALYZE ANY to the schema owning the mapping.

Enable Parallel DML

If you select this option, Parallel DML is enabled 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, a single SQL statement is generated (`multi_table_insert`) that inserts data into multiple tables based on the same set of source data.

Note that the multiple table insert is performed only if this option 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 does not include active operators, such as an Aggregator or Joiner operator, between the Splitter and the target. In addition, the multiple insert is available only for tables. It is not available for 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 18-37.

Do not select this option for mappings run in row based mode or for mappings executed by Oracle*8i* server. Also, do not select this option when auditing is required.

When this option is selected, one total SELECT and INSERT count is returned 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 code is generated for a mapping, the code for all possible operating modes is generated. 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 DML is performed using LCR APIs if available. If no LCR APIs are available, then the standard DML is used.

Database Link

This parameter is maintained for backward compatibility only.

In previous releases, you could select a database link by name from the 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 that is used 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 releases, 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 about each of these settings and additional information about how to design mappings for PEL, see "Improved Performance through Partition Exchange Loading" on page 22-24.

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 box, 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, commonly used hints such as APPEND and PARALLEL are added. 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 the **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 Ellipsis button next to the **Inline View Hint** option in the Configuration Properties window.

For information about optimizer hints and how to use them, see *Oracle 9i 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, the foreign key constraints on the target table are maintained before loading data. If you do not select this option, the foreign key constraints on target tables are disabled before loading the data and

then re-enabled after loading. Constraint violations found during re-enable are identified in the runtime audit error table and, if specified, loaded into an exceptions table.

When you disable constraints, loading is quicker because a constraint check 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.

The disabling and enabling of constraints happen on the target table. When `Enable Constraints` property is disabled, the constraints on the target table will be disabled prior to the loading of data, and will be re-enabled after loading of data. When the constraints are re-enabled, the entire table is scanned and rows that violate the constraints are logged in the exceptions table and these are reported as constraint violation errors in the audit browser.

Consider a scenario where the target table is empty and the `Enable Constraints` property is disabled. Initially suppose the source table has 10 rows, of which 2 rows violate the constraint on the target table. When the mapping is executed, the constraints on the target table are first disabled, then data is loaded (all 10 rows), and then constraints on the target table are re-enabled. When the constraints are re-enabled, the 2 rows that violate the constraints are logged into the exceptions table. The audit browser reports that there are 2 constraint violation errors.

Later, the mapping is again executed with a new source table containing 20 rows, of which 5 rows violate the constraint on the target table. After the data is loaded into the target table (all 20 rows), the target table has 30 rows. When the constraints on the target table are re-enabled, 7 rows will be logged into the exceptions table and reported as constraint violation errors in the audit browser. These include the 5 rows reported newly as well as the 2 rows reported initially. This is because the Warehouse Builder scans the entire target table, which means that all 30 rows will be checked and therefore the 2 rows with violations from the first data load will still be included. Warehouse Builder cannot identify only the new rows added when the mapping was executed the second time. Therefore, unless you truncate the target table before each data load, you will always see the constraint violations from the previous data loads reported each time.

Deselecting the Enable Constraints option is subject to the following restrictions:

– For set based operating mode, the deselect 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 deselect 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 option is not selected.
  - For the TRUNCATE/INSERT DML type, the deselect 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 the 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 file names 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 box contains additional settings for Mapping Flat File operators, depending on how the operators are used in the mapping.

- **Flat File Operators as a Target:** A PL/SQL deployment code package is generated. For information about configuring the parameters associated with a Mapping Flat File operator used as a target, see "Flat File Operators as a Target" on page 22-39.
- **Flat File Operator as a Source:** SQL*Loader scripts are generated. For information about the parameters associated with a Mapping Flat File operator used as a source, see "Flat File Operator as a Source" on page 22-39.

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**.

   Warehouse Builder displays the Configuration Properties dialog box.

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 box.

3. Select the **Deployable** option to generate a set of scripts for mapping objects marked as deployable. If this option is not selected for a mapping, scripts are not generated 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 22-31.

7. Set the **Code Generation Options** as described in "Code Generation Options" on page 22-33.

8. Set the Sources and Targets Reference as described in "Sources and Targets Reference" on page 22-34.

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 computer where you installed the 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**.
2. Select 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 box.
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, an attempt is made 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 that is generated 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 is generated. This value specifies the maximum number of records to load. If a value is not specified, then 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 is 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 about each SQL*Loader option and clause, see *Oracle Database Utilities*.

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 22-35.

Configuring Process Flows Reference

To configure a process flow module:

1. Right-click the process flow module and select **Configure.**

   Warehouse Builder displays the Configuration Properties dialog box 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 dialog box 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.

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

23

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
- About Data Profiling
- About Data Correction and Augmentation
- About Data Rules
- About Quality Monitoring
- Performing Data Profiling
- Tuning the Data Profiling Process
- Using Data Rules
- Monitoring Data Quality Using Data Auditors
- Setting Data Watch and Repair for Oracle Master Data Management (MDM)

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. Using Warehouse Builder for data management provides the following benefits:

- 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 definitions.
- 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 decisions you make on how to correct data.

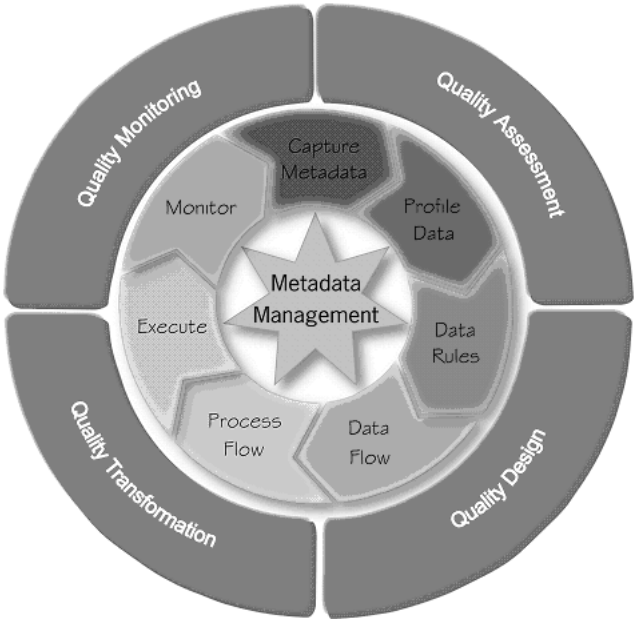
Phases in the Data Quality Lifecycle

Ensuring data quality involves the following phases:

- Quality Assessment
- Quality Design
- Quality Transformation
- Quality Monitoring

Figure 23–1 shows the phases involved in providing high quality information to your business users.

Figure 23–1 Phases Involved in Providing Quality Information



This screenshot shows the Data Quality Process. The outermost circle is divided into four parts. They are:

- Quality Assessment
- Quality Design
- Quality Transformation
- Quality Monitoring

The inner circle is divided into seven parts. They are:

- Capture Metadata
- Profile Data
- Data Rules

- Data Flow
- Process Flow
- Execute
- Monitor

In the Center of the inner circle is a star that represents metadata management.

****

Quality Assessment

In the quality assessment phase, you determine the quality of the source data. The first step in this phase is to import 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.

See Also:

- "About Data Profiling" on page 23-4
- "Performing Data Profiling" on page 23-16

Quality Design

The quality design phase consists of designing your quality processes. You can specify the legal data within a data object or legal relationships between data objects using data rules.

See Also:

- "About Data Rules" on page 23-13
- "Using Data Rules" on page 23-42

You can also correct and augment your data using data quality operators.

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.

See Also:

- "Generate Corrections" on page 23-32 for information about generating corrections based on the results of data profiling
- "About Data Correction and Augmentation" on page 23-11 for information about the data quality operators

Quality Transformation

The quality transformation phase consists of running the correction mappings you designed to correct the source data.

Quality Monitoring

Quality monitoring is the process of examining warehouse data over time and alerting you when the data violates business rules set for the data.

See Also:

- "About Quality Monitoring" on page 23-15
- "Monitoring Data Quality Using Data Auditors" on page 23-45

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 mappings and schema correction scripts. 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

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.

See Also:

- "Performing Data Profiling" on page 23-16
- "Create a Data Profile" on page 23-17

Benefits 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.

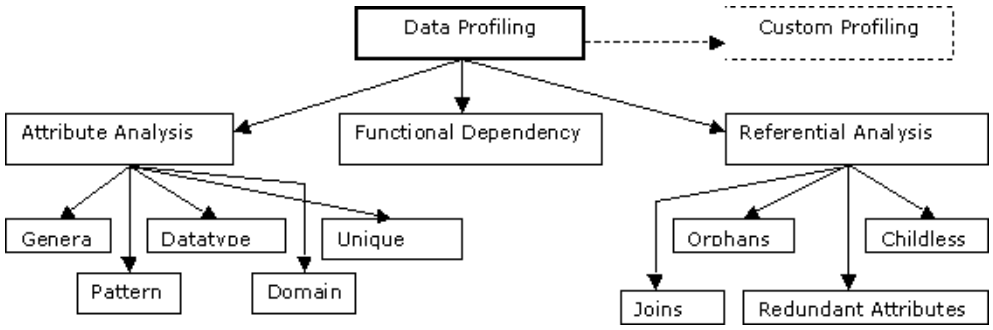
Data profiling offers three main types of analysis:

- Attribute Analysis
- Functional Dependency
- Referential Analysis

In addition to these analysis, you can create custom profiling processes using data rules that allows you to validate custom rules against the actual data and get a score of their accuracy.

Figure 23–2 displays a representation of the types of data profiling and how you can perform each type.

Figure 23–2 Data Profiling Overview



This screenshot contains a graphical representation of the types of analysis performed in each type of data profiling. At the top is a box labeled Data Profiling. Below this box are three boxes labeled, from left to right, Attribute Analysis, Functional Dependency, and Referential Analysis. Arrows connect Data Profiling to each of these boxes.

At the bottom of the image are two rows of boxes. The first row contains the following boxes, from left to right: General, Datatype, Unique, Orphans, and Childless. In the next row are the following boxes, from left to right: Pattern, Domain, Joins, and Redundant Attributes. Arrows connect the box labeled Attribute Analysis to the following boxes: General, Datatype, Unique, Pattern, and Domain. Arrows connect the box labeled Referential Analysis to the following boxes: Orphans, Childless, Joins, and Redundant Analysis.

****

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.

Attribute analysis consists of pattern analysis, domain analysis, data type analysis, and unique key analysis.

Pattern Analysis

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 23–1 shows a sample attribute, Job Code, that could be used for pattern analysis.

Table 23–1 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 23–2 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 23–2 Pattern Analysis Results

Job Code	% Occurred
DDD-X-DD	75%
DD-X-D	25%

Domain Analysis

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.

Datatype Analysis

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

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 23–3 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 23–3 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 23–4 and Table 23–5 show the contents of two tables that are candidates for referential analysis. Table 23–4 is the child object and Table 23–5 is the parent object.

Table 23–4 Employees Table (Child)

ID	Name	Dept. Number	City
10	Alison	17	NY
20	Rochnik	23	SF

Table 23–4 (Cont.) Employees Table (Child)

ID	Name	Dept. Number	City
30	Meijer	23	SF
40	Jones	15	SD

Table 23–5 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 23–6. You can then catch errors such as the one for employee Alison.

Table 23–6 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	

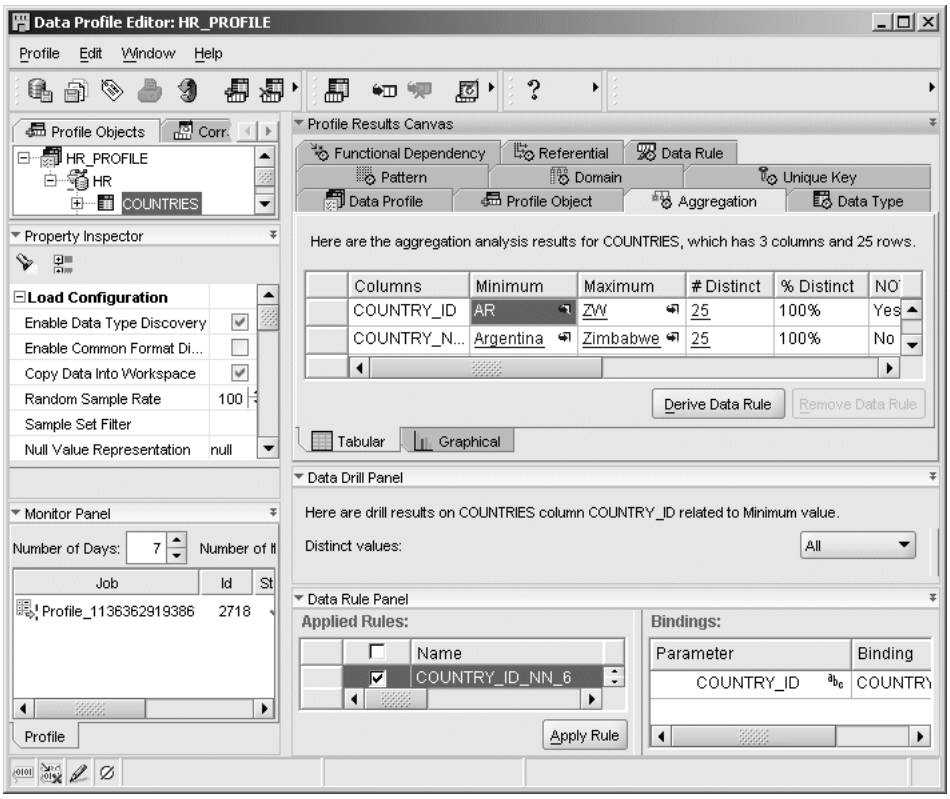
About the Data Profile Editor

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 23–3 displays the Data Profile Editor.

Figure 23–3 Data Profile Editor



This image displays the Data Profile Editor.

****

The Data Profile Editor consists of the following:

- Menu Bar
- Toolbars
- Object Tree
- Property Inspector
- Monitors Panel
- Profile Results Canvas
- Data Drill Panel
- Data Rule Panel

Refer to the online help for detailed information about the contents of each panel.

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 for 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 for 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 Correction and Augmentation

Warehouse Builder enables you to automatically create corrected data objects and 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 to correct and cleanse source data.

To perform data correction and augmentation, use one of the following methods:

- Automatic Data Correction Based on Data Profiling Results
- Data Correction and Augmentation Using Operators

Automatic Data Correction Based on Data Profiling Results

When you perform data profiling, Warehouse Builder generates corrections for the objects that you profiled. You can then decide to create corrected objects based on results of data profiling. The corrections are in the form of data rules that can be bound to the corrected object.

Types of Corrections for Source Data

You can perform the following types of corrections on source data objects:

- **Schema correction**

  Schema correction creates scripts that you can use to create a corrected set of source data objects with data rules applied to them. The corrected data objects adhere to the data rules derived from the results 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`.

- **Data Correction**

  Data correction is the process of creating correction mappings to remove anomalies and inconsistencies in the source data before loading it into the corrected data objects. Correction mappings enforce the data rules defined on the data objects. 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.

  The name of the correction mapping is the object name prefixed with M_. For example, the correction mapping for the `EMPLOYEE` table is called `M_EMPLOYEE`.

About Performing Data Correction

To perform data correction on source data, you must specify the following:

- Data Correction Actions
- Cleansing Strategies for Data Correction

Data Correction Actions

Based on the data profiling results, Warehouse Builder derives a set of data rules that you can use to cleanse the source data. You can automatically generate corrections based on these data rules by performing data correction actions.

For each data rule derived as a result of data profiling, you must choose a correction action that specifies how data values that are not accepted due to data rule enforcement should be handled. The correction actions you can choose are:

- **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. It is similar to the Ignore option, except that a report is created that contains the values that do 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 "Cleansing Strategy" on page 23-35 for details about specifying cleansing strategies.

Cleansing Strategies for Data Correction

When you decide to automatically generate corrected objects based on data profiling results, you must specify how inconsistent data from the source should be cleansed before being stored in the corrected object. To do this, you specify a cleansing strategy for each data rule that is applied to the correction object. Error tables are used to store the records that do not conform to the data rule.

The cleansing strategy you use depends on the type of data rule and the rule configuration. Table 23–7 describes the cleansing strategies and lists the types of data rules for which each strategy is applicable.

Table 23–7 Cleansing Strategies for Data Correction

Cleansing Strategy	Description	Applicable To Data Rule Types
Remove	Does not populate the target table with error records.	All
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.	Domain List Domain Pattern List Domain Range Common Format No Nulls Name and Address Custom

Table 23–7 (Cont.) Cleansing Strategies for Data Correction

Cleansing Strategy	Description	Applicable To Data Rule Types
Set to Min	Sets the attribute value of the error record to the minimum value defined in the data rule.	Domain Range rules that have a minimum value defined
Set to Max	Sets the attribute value of the error record to the maximum value defined in the data rule.	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.	Domain List rules with character data type
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.	Domain List rules with character data type
Merge	Uses the Match-Merge algorithm to merge duplicate records into a single row.	Unique Key
Set to Mode	Uses the mode value to correct the error records if a mode value exists for the functional dependency partition that fails.	Functional Dependency

See Also: "Types of Data Rules" on page 23-14

Data Correction and Augmentation Using Operators

Warehouse Builder provides functionality that enables you to design data correction and augmentation processes for the source data. While transforming the source data, you can use the following operators to ensure data quality:

- Match-Merge Operator
- Name and Address Operator

See Also: Chapter 24, "Data Quality Operators" for information about these operators and their usage

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 workspace. 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 23-42.

Types of Data Rules

Table 23–8 describes the types of data rules.

Table 23–8 Types of Data Rules

Data Rule Type	Description	Example
Domain List	Defines a list of values that an attribute is allowed to have.	The Gender attribute can have 'M' or 'F'.
Domain Pattern List	Defines a list of patterns that an attribute is allowed to conform to. The patterns are defined in the Oracle Database regular expression syntax.	A pattern for telephone number is: (^[[:space:]]*[0-9]{ 3 }[[:punct:]|:space:]]?[0-9]{ 4 }[[:space:]]*$)
Domain Range	Defines a range of values that an attribute is allowed to have.	The value of the Salary attribute can be between 100 and 10000.
Common Format	Defines a known common format that an attribute is allowed to conform to. This rule type has many subtypes: Telephone Number, IP Address, SSN, URL, E-mail Address. Each type has predefined formats listed. You can add more formats to this list.	An E-mail address should be in the following format: ^(mailto:[-_a-z0-9.]+@[-_ a-z0-9.]+$)
No Nulls	Specifies that the attribute cannot have null values.	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.	The Dept_name attribute is dependent on the Dept_no attribute.
Unique Key	Defines whether an attribute or group of attributes are unique in the given data object.	The name of a department must be unique.
Referential	Defines the type of relationship (1:x) a value must have with another value.	The department_id attribute of the Departments table must have a 1:n relationship with the Department_id attribute of the Employees table.
Name and Address	Uses the Name and Address support to evaluate a group of attributes as a name or address.	The department_id attribute of the Departments table should have a 1:n relationship with the department_id attribute of the Employees table.
Custom	Applies an SQL expression that you specify to its input parameters.	A custom rule called VALID_ DATE has two input parameters, START_DATE and END_DATE. A valid expression for this rule is defined as follows: "THIS"."END_DATE" > "THIS"."START_DATE".

Implementation of Data Rules

Warehouse Builder uses different methods of applying data rules to a correction object. The method used depends on the type of data rule you are implementing.

Table 23–9 describes the methods used for object schema correction. It also lists the data rule types for which each correction is used.

Table 23–9 Data Rules Implementation for Schema Correction

Schema Correction Method	Description	Data Rule Types for which Correction Method Can be Used
Create 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 of the Apply Data Rule Wizard.	Custom Domain List Domain Pattern List Domain Range Common Format No Nulls Unique Key
Change 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.	
Create 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.	Functional Dependency
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.	Name and Address

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 must create data auditors. Data auditors ensure that your data complies with the business rules you defined.

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 by auditing and marking how many errors are occurring against the audited data. To monitor and audit a data object, you must first define data rules that specify valid data for that data object.

Data auditors are an important tool in ensuring that data quality levels are up to the standards set by the users of the system. They also help to determine spikes in bad data allowing events to the tied to these spikes.

> **See Also:** "Monitoring Data Quality Using Data Auditors" on page 23-45

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.

> **See Also:** "Auditing Data Objects Using Data Auditors" on page 23-47

Data Auditor Thresholds

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. You can specify a threshold value for each data rule that the data auditor audits. This value is used to determine if the data in the data object is within the limits that you defined. Based on this threshold, the process can choose actions.

For example, you create a data auditor the audit the data in the Employees table. This table contains two data rules, emp_email_unique_rule and emp_sal_min_rule. You specify the threshold value for both rules is 80%. This means that if less than 80% of the data in the Employees table does not comply with the data rules, the auditing for the table fails.

> **Tip:** "Specifying Data Correction Actions" on page 23-46 for information about specifying threshold value

Audit Results for Data Auditors

In addition to setting thresholds for non-compliant records, you can also capture audit results and store them for analysis purposes. When executed, the data auditor sets several output values. One of these values is the audit result. Audit results provide information about the extent of data rule violations that occurred while running the data auditor.

> **See Also:** "Data Auditor Execution Results" on page 23-48

Performing 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 you begin profiling data, first reduce the data set by doing a random sampling. Next 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.

> **See Also:** "Tuning the Data Profiling Process" on page 23-40

Data Profiling Restrictions

- You can only profile data in the default configuration.
- The profiling workspace location should be an Oracle 10*g* database or higher.
- You cannot profile a source table that contains complex data types if the source module and the data profile are located on different database instances.
- You cannot profile a table that contains more than 165 columns.

Steps to Perform Data Profiling

After you have chosen the objects 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
5. Derive Data Rules
6. Generate Corrections
7. Define and Edit Data Rules Manually
8. 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. Step 8 is required when you perform both data profiling and data correction along with 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 must explicitly register the locations before you begin profiling.

Create a Data Profile

After your system is set up, you can create a data profile using the Design Center. A data profile is a metadata object in the workspace. 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).

To create a data profile:

1. From the Project Explorer, expand the project node in which you want to create a data profile.
2. Right-click Data Profiles and select **New**.

The Welcome page of the Create Data Profile Wizard is displayed.

3. On the Name and Description page, enter a name and an optional description for the data profile. Click **Next**.
4. On the Select Objects page, select the objects you want to include in the data profile and use the arrows to move them to the Selected list. Click **Next**.

   To select multiple objects, hold down the **Ctrl** key while selecting objects. You can include tables, views, materialized views, external tables, dimensions, and cubes in your data profile.
5. (Optional) If you selected tables, views, or materialized views that contain attribute sets, the Choose Attribute Set dialog box is displayed. The list at the bottom of this dialog box displays the attribute sets defined on the data object.
   - To profile only the attributes defined in the attribute set, select the attribute set from the list.
   - To profile all columns in the data object, select **<all columns>** from the list.
6. (Optional) If you selected dimensional objects on the Select Objects page, a warning is displayed informing you that the relational objects bound to these dimensional objects will also be added to the profile. Click **Yes** to proceed.
7. On the Summary page, review the choices you made on the previous wizard pages. Click **Back** to change any selected values. Click **Finish** to create the data profile.

   The Warehouse Builder note dialog is displayed. Click **OK** to display the Data Profile Editor for the newly created data profile.

   The new data profile is added to the Data Profiles node in the navigation tree.

Using Attribute Sets to Profile a Subset of Columns from a Data Object

You can use an attribute set to restrict a data profiling operation to a subset of columns from a table, view or materialized view. Reasons to use an attribute set include:

- You can decrease profiling time by excluding columns for which you do not need profiling results.
- Data profiling can only profile up to 165 columns from a table, view or materialized view at a time. You can use an attribute set to select a set of 165 or fewer columns to profile from the object.

Data profiling, using attribute sets, consists of the following high-level steps:

1. Defining Attribute Sets
2. Profiling the Columns in an Attribute Set

Defining Attribute Sets Use the following steps to define an attribute set in a table, view, or materialized view.

1. In the Project Explorer, double-click the table, view, or materialized view.

   The Data Object Editor for the selected object is opened.
2. In the Details panel, select the Attribute Sets tab.
3. In the Attribute Sets section, click a blank area in the Name column and enter the name of the attribute set to create.
4. Close the Data Object Editor.

5. Double-click the data object in which you created an attribute set in step 3.

   The Data Object Editor for the selected object is displayed.

6. On the Attributes Sets tab, select the name of the attribute set created in step 3.

   The Attributes of the selected attribute set section displays the attributes in the data object.

7. Select **Include** for all the attributes that you want included in the attribute set.

8. Save your changes and close the Data Object Editor.

Profiling the Columns in an Attribute Set Use the following steps to profile columns contained in the attribute set.

1. In the Project Explorer, right-click the Data Profiles node and select **New**.

   The Welcome page of the Create Data Profile Wizard is displayed.

2. On the Welcome page, click **Next**.

3. On the Name and Description page, enter a name and an optional description for the data profile. Click **Next**.

4. On the Select Objects page, select the data object that you want to profile and use the shuttle arrows to move the data profile to the Selected list.

5. When the selected data object contains attribute sets, the Choose Attribute Set dialog box is displayed.

6. Select the attribute set that you want to profile and click **OK**.

   The Select Objects page is displayed.

7. On the Select Objects page, click **Next**.

8. On the Summary page, review the options you choose on the previous wizard pages and click **Finish**.

   The data profile is created and added to the Navigator tree.

Profile the Data

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.

Steps to Profile 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. 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 in the Project Explorer, right-click a data profile, and select **Open Editor**.

   The Data Profile Editor opens the selected data profile.

2. To specify the types of analysis to be performed on the data, configure the data profile.

See "Configuring Data Profiles" on page 23-20.

3. From the **Profile** menu, select **Profile**.

   - (Optional) If this is the first time you are profiling data, the Data Profile Setup dialog box is displayed. Enter the details of the profiling workspace. For more information about the information to be entered, click **Help**.

   Warehouse Builder begins preparing metadata for profiling. The progress window containing the name of the object being created to profile the data is displayed. After the metadata preparation is complete, the Profiling Initiated dialog box is displayed informing you that the profiling job has started.

4. On the Profiling Initiated dialog box, click **OK**.

   Once the profiling job starts, the data profiling is asynchronous and you can continue working or even close the client. Your profiling process will continue to run until it is completed.

5. 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.

6. After the profiling is complete, the Retrieve Profile Results dialog box is displayed and you are prompted to refresh the results. Click **OK** to retrieve the data profiling results and display them in the Data Profile Editor.

   You can use this option if you have previously profiled data in the same data profile. It allows 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.

Configuring Data Profiles

You can, and should, configure a data profile before running it if there are specific types of analysis you do, or do not, want to run.

You can configure a data profile at one of the following levels:

- The entire profile (all the objects it contains)
- An individual object in the data profile

  For example, the data profile contains three tables. If you want to perform certain specific types of analysis on one table, configure this table only.

- An attribute within an object

  For example, if you know that 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.

Steps to Configure Data Profiles

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

   The Data Profile Editor for the data profile is displayed.

2. On the Property Inspector panel, set configuration properties at the required level.

To configure the entire data profile:

a. On the Profile Objects tab, select the data profile.

b. On the Property Inspector panel, set the configuration properties for the data profile.

To configure a particular object in the data profile:

a. On the Profile Objects tab, expand the node that represents the data profile.

b. Select the data object.

c. On the Property Inspector panel, set the configuration properties for the object.

To configure an attribute within an object in a data profile:

a. On the Profile Objects tab, expand the node that represents the data profile.

b. Expand the data object that contains the attribute and select the required attribute.

c. On the Property Inspector panel, set the configuration properties for the attribute.

Note: "Reference for Setting Data Profiling Configuration Parameters" on page 23-38 for details about configuration parameters you can set for data profiles

---

.

View Profile Results

After the profile operation is complete, you can open the data profile in the Data Profile Editor to view and analyze the results. 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.

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 profiling results for the selected object are displayed using the following tabs of the Profile Results Canvas.

   - Data Profile

- Profile Object
- Aggregation
- Data Type
- Domain
- Pattern
- Unique Key
- Functional Dependency
- Referential
- Data Rule

You can switch between various objects in the data profile. The tab that you had selected for the previous object remains selected.

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 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.

Aggregation

The Aggregation tab displays all the essential measures for each column, such as minimum and maximum values, number of distinct values, and null values. Some measures are available only for specific data types. These include the average, median, and standard deviation measures. Information can be viewed from either the Tabular or Graphical sub-tabs.

Table 23–10 describes the various measurement results available in the Aggregation tab.

Table 23–10 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
# Nulls	The total number of null values for a specific attribute
%Nulls	The percentage of null values over the entire row set number

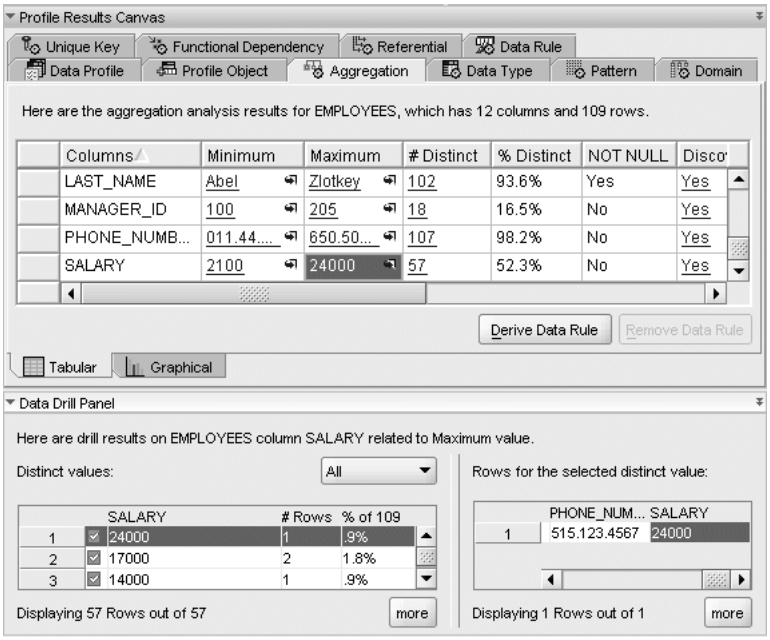
Table 23–10 (Cont.) Aggregation Results

Measurement	Description
Six Sigma	For each column, number of null values (defects) to the total number of rows in the table (opportunities).
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 allows you to analyze the data in the sample that produced this result.

For example, if you scroll to the SALARY column, shown in Figure 23–4, 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 23–4 Aggregation Tabular Results



This image displays the Aggregation tab of the Data Profile Editor. The tab contains the aggregation measures for each column in the source table.

****

The graphical analysis 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 menus to change the displayed data object.

Data Type

The Data Type tab 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.

One example of data type profiling would be finding a column defined as VARCHAR but is actually storing only numeric values. Changing the data type of the column to NUMBER would make storage and processing more efficient.

Table 23–11 describes the various measurement results available in the Data Type tab.

Table 23–11 Data Type Results

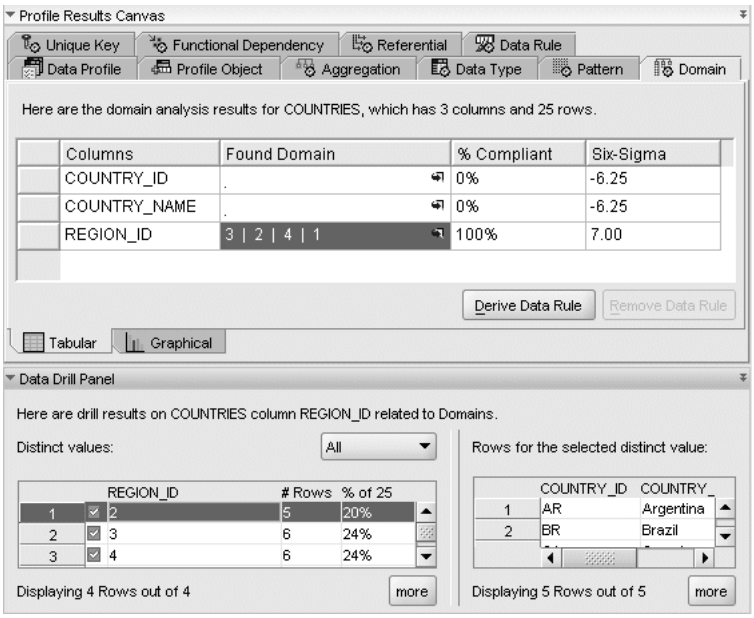
Measurement	Description
Columns	Name of the column
Documented Data Type	Data type of the column in the source object
Dominant Data Type	From analyzing the column values, data profiling determines that this is the dominant (most frequent) data type.
% Dominant Data Type	Percentage of total number of rows where column value has the dominant data type
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 the 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

Domain

The Domain tab displays results about the possible set of values that exist in a certain attribute. Information can be viewed from either the Tabular or Graphical subtabs.

Figure 23–5 displays the Domain tab of the Data Profile Editor.

Figure 23–5 Domain Discovery Results



This image displays the Domain tab of the Data Profile Editor that contains the results of domain profiling.

****

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.

Table 23–12 describes the various measurement results available in the Domain tab.

Table 23–12 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

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 23–13 describes the various measurement results available in the Pattern tab.

Table 23–13 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 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, a phone number is unique in 98% of the records. It can be a unique key candidate, and you can then cleanse the noncompliant records. You can also use the drill-down feature to view the cause of the duplicate phone numbers in the Data Drill panel. Table 23–14 describes the various measurement results available in the Unique Key tab.

Table 23–14 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 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** 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.

Table 23–15 describes the various measurement results available in the Functional Dependency tab.

Table 23–15 Functional Dependency Results

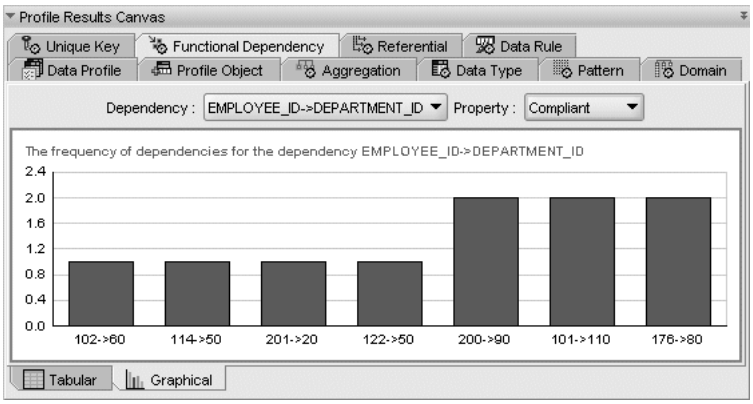
Measurement	Description
Determinant	The name of the attribute that is found to determine the attribute listed in the Dependent
Dependent	The name of the attribute found to be determined by value of another attribute
# Defects	The number of values in the Determinant attribute that were not determined by the Dependent attribute
% Compliant	The percentage of values that are compliant with 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** list, the information shown is limited to absolute dependencies. If you have an attribute that is always dependent on another attribute, it is recommended 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 sub-tab so that you can view information graphically. You can select a dependency and property from the lists to view graphical data.

For example, in Figure 23–6, you select the dependency where DEPARTMENT_ID seems to determine COMMISSION_PCT. (DEPARTMENT_ID->COMMISSION_PCT). In a 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 23–6 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 23–6 Graphical Functional Dependency



This image displays the Graphical sub-tab of the Functional Dependency tab. It provides a graphical representation of the results of functional dependency profiling.

****

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 sub-tabs are available only in the Referential tab: Joins and Redundant Columns.

Table 23–16 describes the various measurement results available in the Referential tab.

Table 23–16 Referential Results

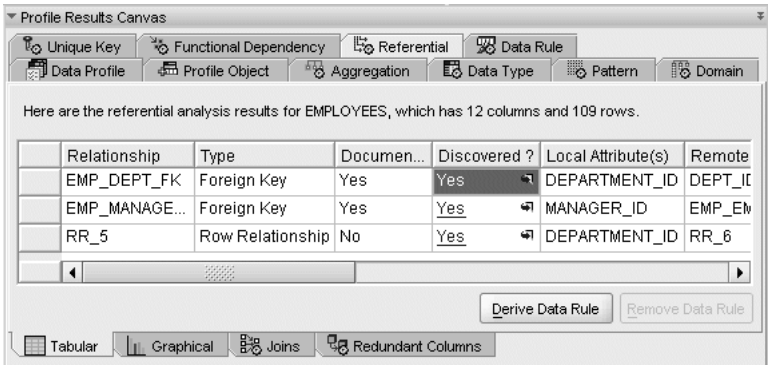
Measurement	Description
Relationship	The name of the relationship.
Type	The type of relationship.
Documented?	Yes or No. Yes indicates that a foreign key on the column exists in the data dictionary. No indicates that the foreign key was discovered as a result of data profiling.
Discovered?	From analyzing 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.
Remote Module	The name of the module that contains the referenced object.

Table 23–16 (Cont.) Referential Results

Measurement	Description
Cardinality Range	The range of the cardinality between two attributes. For example, the EMP table contains 5 rows of employee 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 per 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 are compliant with 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, if you are analyzing two tables for referential relationships: the Employees table and the Departments table. Using the Referential data profiling results shown in Figure 23–7, you discover that the DEPARTMENT_ID column in the Employees table is 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 23–7 Referential Results



This image displays the Referential tab of the Data Profile Editor. The tab displays the results of referential data profiling in a tabular format. Each row represents one column in the source object.

****

You can also select the Graphical subtab to view information graphically. This view is effective for viewing noncompliant records, such as orphans. To use the Graphical sub-tab, make a selection from the **Reference** and **Property** lists.

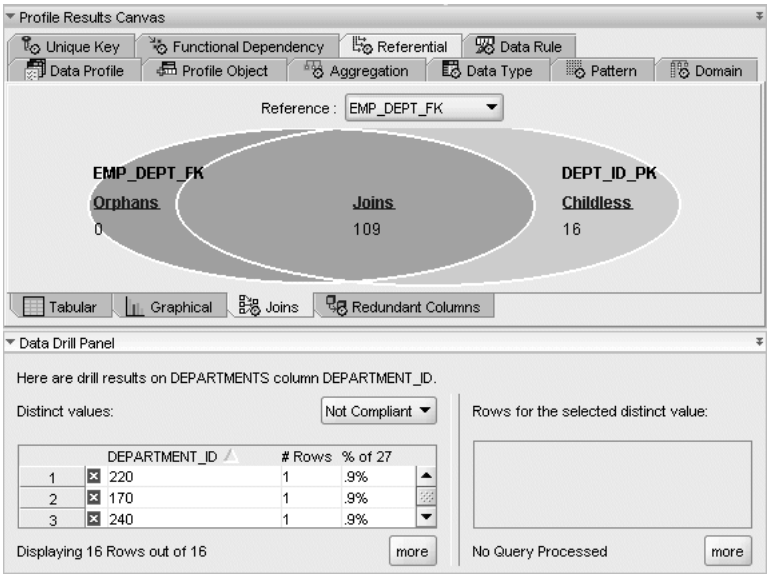
The Joins sub-tab displays a join analysis on the reference selected in the Reference list. The results show the relative size and exact counts of the three possible outcomes for referential relationships: joins, orphans, and childless objects.

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 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.

Figure 23–8 displays the Joins subtab of the Referential tab.

Figure 23–8 Join Results



This image displays the Joins subtab of the Referential tab.

****

The Redundant Columns sub-tab 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, consider two tables `EMP` and `DEPT`, shown in Table 23–17 and Table 23–18, having the following foreign key relationship: EMP.DEPTNO (uk) = DEPT.DEPTNO (fk).

Table 23–17 EMP Table

Employee Number	Dept. No	Location
100	1	CA
200	2	NY
300	3	MN

Table 23–18 DEPT Table

Dept No	Location	Zip
1	CA	94404
3	MN	21122

Table 23–18 (Cont.) DEPT Table

Dept No	Location	Zip
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.

Derive Data Rules

Based on the results of data profiling, you can derive 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. Use data rules 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.

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 example, if 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 a prime candidate for the unique key. The results also show that 85% of the values in the `Gender` column are either 'M' or 'F', making it a good candidate for a domain. You can then derive these rules directly from the Profile Results Canvas.

Steps to Derive Data Rules

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.

   For example, to create a data rule that enforces a unique key rule for the EMPLOYEE_NUMBER column, navigate to the Unique Key tab.

4. Select the cell that contains the results you want derived into a data rule.

5. From the **Profile** menu select **Derive Data Rule**. Or click the **Derive Data Rule** button.

   For example, to create a Unique Key rule on the EMPLOYEE_NUMBER column, select this column and click **Derive Data Rule**.

   The Derive Data Rule Wizard displays.

6. On the Welcome page, click **Next**.

7. On the Name and Description page, the **Name** field displays a default name for the data rule. To specify a new name, select the name, enter the new name, and click **Next**.

8. On the Define Rule page, provide details about the data rule parameters and click **Next**.

   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. On the Summary page, review the options you set in the wizard using this page. Click **Back** if you want to change any of the selected values. Click **Finish** to create the data rule.

   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 the Data Rules node in the Project Explorer. You can reuse this data rule by attaching it to other data objects.

Generate Corrections

After you have derived data rules from the profiling results, you can automate the process of correcting source data based on profiling results. You can create the schema and mapping corrections. The schema correction creates scripts that you can use 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.

Steps to Automate Data Correction

1. Generate correction objects based on data profiling results.

   See "Steps to Create Corrections" on page 23-33.

   You can view the corrected objects as described in "Viewing the Correction Tables and Mappings" on page 23-35.

2. Deploy the generated corrections to create the corrected objects in the target schema.

   See "Steps to Deploy Correction Objects" on page 23-35.

Steps to Create Corrections

Use the Data Profile Editor to create corrections based on the profiling results.

To create corrections:

1. If the Data Profile is not already open, open it by right-clicking the data profile in the Project Explorer and selecting **Open Editor**.
2. From the Profile menu, select **Create Correction**.

   The Create Correction Wizard is displayed.
3. On the Select Target Module page, specify the target module that will contain the corrections and click **Next**. You can create a new module or use an existing module.

   - To store the corrections in an existing target module, choose **Select an existing module** and select the module from the Available list.
   - To store correction in a new target module, select **Create a new target module**. The Create Module Wizard guides you through the steps of creating a new target module.

   You can remove correction objects created as a result of previous corrections by selecting **Remove previous correction objects**.
4. On the Select Objects page, select the objects that you want to correct by moving them to the Selected list and click **Next**.

   The **Filter** 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.
5. On the Select Data Rules and Data Types page, perform schema correction by selecting the corrections that you want to implement for each object. Click **Next**.

   See "Selecting Data Rules and Data Types for Corrected Objects" on page 23-34.
6. (Optional) On the Data Rules Validation page, note down the validation errors, if any, and correct them before proceeding.

   If correction objects from a previous data correction action exist for the objects selected for correction, this page displays a message. Click **Next** to remove previously created correction objects.
7. On the Verify and Accept Corrected Tables page, select the objects that you want to correct and click **Next**.

   Se e "Selecting the Objects to be Corrected" on page 23-34.
8. On the Choose Data Correction Actions page, specify the correction actions to be performed for objects and click **Next**.

   See "Choosing Data Correction and Cleansing Actions" on page 23-35.
9. On the Summary page, click **Finish** to create the correction objects.

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. Depending on the data correction actions specified, correction objects may include transformations used to cleanse and correct data.

Selecting Data Rules and Data Types for Corrected Objects

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.

> **See Also:** "Implementation of Data Rules" on page 23-15.

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.

Selecting the Objects to be Corrected

Use the Verify and Accept corrected Tables page to confirm the objects that you want to correct. This page lists the tables that have been modified with the applied rules and the data type changes.

Perform the following actions on this page:

1. To create a data object in the corrected schema, select **Create** to the left of the object.

   The lower part of the page displays the details of the selected object using tabs. The Columns tab displays the details of columns that will be created in the corrected data object. The Constraints tab displays details of constraints that will be created on the corrected data object. The Data Rules tab displays details of data rules that will be created on the corrected data object.

2. On the Columns tab, perform the following actions:

   - To create a displayed column in the corrected data object, select **Create** to the left of the column name.
   - To remove a displayed column from the corrected object, deselect **Create** to the left of the column name.
   - To modify a displayed column in the corrected object, edit the Data Type, Length, Precision, Seconds Precision, and Scale attributes. However, you cannot modify a column name.

3. On the Constraints tab, perform the following actions:

   - To add additional constraints, click **Add Constraint**.
   - To remove a constraint from the corrected data object, select the constraint and click **Delete**.

4. On the Data Rules tab, perform the following actions:
   - To apply a data rule, that was derived based on data profiling results, to the corrected object, select the check box to the left of the data rule.

     Ensure that the Bindings column contains the column to which the data rule should be applied.
   - To apply a new data rule to the corrected object, click **Apply Rule**. The Apply Data Rule wizard guides you through the process of applying a data rule.

Choosing Data Correction and Cleansing Actions

Use the Choose Data Correction Actions page to select the actions 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 menu in the **Action** column. The settings you select here determine how to handle data values that are not accepted due to data rule enforcement. Select one of the following actions:

- Ignore
- Report
- Cleanse

> **See Also:** "Data Correction Actions" on page 23-12 for a description of each correction action.

Cleansing Strategy Use the **Cleansing Strategy** list to specify a cleansing strategy. This option is enabled only if you select 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.

> **See Also:** "Cleansing Strategies for Data Correction" on page 23-12

Steps to Deploy Correction Objects

To deploy the correction mappings created as part of the data correction process:

1. 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 is granted to `PUBLIC` on the `HR.EMPLOYEES` table.
2. To create the correction objects in your target schema, deploy the correction tables created as a result of data profiling.
3. To cleanse data, deploy and execute the correction mappings.

Viewing the 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.

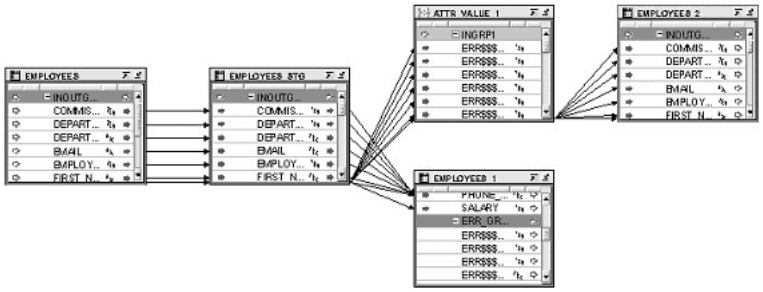
To view the correction mappings:

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 23–9 displays a correction map generated by the Create Correction wizard.

Figure 23–9 Generated Correction Mapping



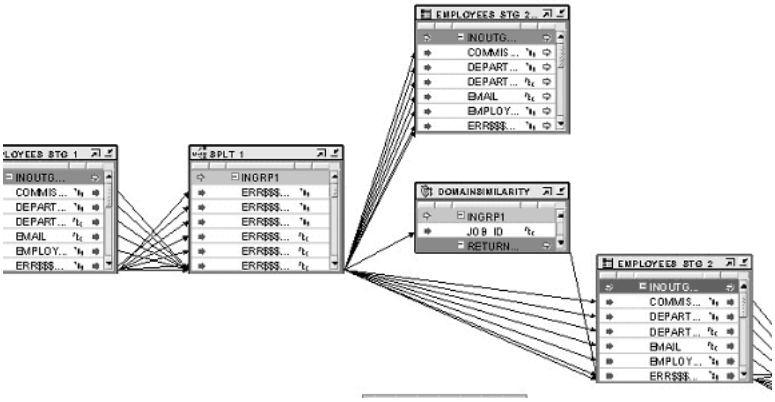
This image displays the correction mapping generated as a result of performing correction action based on data profiling results.

****

3. Select the submapping ATTR_VALUE_1 and click the Visit Child Graph icon from the toolbar to view the submapping.

   Figure 23–10 displays the submapping that is displayed.

Figure 23–10 Correction Submapping



This image displays the submapping of the correction mapping.

****

The submapping is the element in the mapping that performs 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.

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 Rules" on page 23-43.

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.

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 box.

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 box is displayed.

3. Edit any of the following properties of the data profile and click **OK**.

   - To modify the name or description of the data profile, on the Name tab, select the name or description and enter the new value.
   - To add or remove objects, on the Select Objects tab, use the arrows to add objects to or remove objects from the data profile.
   - To change the location that is used as the data profiling staging area, use the Data Locations tab.

     Use the arrows to move the new location to the Selected Locations section. Ensure that you select **New Configuration Default** to set this location as the default profiling location for the data profile.

     Note: If you modify the profiling location after you have performed data profiling, the previous profiling results are lost.

Adding Data Objects to a Data Profile

To add data objects to a data profile, complete 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 box is displayed.

3. On the Add Profile Tables dialog box, select the objects that you want to add to the data profile. Select the objects and use the arrows to move them to the Selected section.

   You can select multiple objects by holding down the Ctrl key and selecting objects.

Reference for Setting Data Profiling Configuration Parameters

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 display the Expression Builder. Use the Expression Builder to define the WHERE clause.
- **Null Value Representation:** This value will be considered as 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, and not for an individual attribute.

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 the Data Profile 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.

> **See Also:** "Reference for Setting Data Profiling Configuration Parameters" on page 23-38

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 is 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 workspace 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 computer 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 computer 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 10 million rows), it is highly recommended to use a multiple processor computer.

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 ensure this by assigning a larger size of the System Global Area. It is recommended that the size of the System Global Area be at least 500 MB. If possible, configure it to 2 GB or 3 GB.

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 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) X 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 and 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* or a higher version and you do not have any volume management software, you can use the Automatic Storage Management feature of the Oracle Database to spread the 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 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.

Use the Design Center to create and edit data rules. Once you create a data rule, 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 that 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 would be enforced, and schemas would be normalized. The second way data rules are used is in a correction mapping that validates 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.

Creating Data Rule Folders

Each data rule belongs to a data rule folder, which is a container object that groups related data rules. To create a data rule, you must first create a data rule folder.

To create a data rule folder, in the navigation tree, right-click **Data Rules** and select **New**. The Create Data Rule Folder dialog box is displayed.

Creating Data Rules

The Data Rules folder in the Project Explorer contains the data rules. Every data rule must belong to a data rule folder. The subfolder DERIVED_DATA_RULES contains the data rules derived as a result of data profiling. You can create additional data rule folders to contain any data rules that you create.

To create a data rule:

1. Right-click the Data Rule folder in which the data rule should be created and select **New**.

   The Welcome page of the Create Data Rule Wizard is displayed.

2. On the Welcome page, click **Next**.

3. On the Name and Description page, specify a name and an optional name for the data rule. Click **Next**.

4. On the Define Rule page, specify the type of data rule to create. Also specify any additional information required to create the data rule. Click **Next**.

   See "Defining the Data Rule" on page 23-43 for information about defining the data rule.

   For example, when you create a Domain Range rule, you must specify the values that represent the valid domain values.

5. On the Summary page, review the selections you made in the wizard. Click **Back** to modify any selected values. Click **Finish** to create the data rule.

   The data rule is added to the data rule folder under which you created the data rule.

Defining the Data Rule

Use the Define Rule page or the Define Rule tab to provide details about the data rule. The top section of the page displays the **Type** list that represents the type of data rule. 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 23-14.

> **Note:** When you are deriving a data rule, the Type field is automatically populated and you cannot edit this value.

The bottom section of this page specifies additional details about the data rule. The number and names of the fields displayed in this section depend on the type of data rule you create.

For example, if you select Custom as the Type, 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 section 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.

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:

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

   The Edit Data Rule dialog box is displayed.

2. On the Name tab, you can perform the following tasks:

   - To rename a data rule, select the name and enter the new name.
   - To edit the description for the data rule, select the description and enter the new description.

3. On the Define tab, edit the properties of the data rule.

   Note: You cannot change the type of data rule. You can only modify the properties related to that type of data rule such as the domain bounds, domain list, number of attributes in a unique key, and so on.

Applying Data Rules to Objects

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 a data object. You can apply precreated data rules or any data rule you created to data objects. The types of data objects to which you can apply data rules are tables, views, materialized views, and external tables.

To apply a data rule to a data object:

1. In the Project Explorer, right-click the object to which you want to apply a data rule select **Open Editor**.

   The Data Object Editor for the data object is displayed.

2. Go to the Data Rules tab.

   Any data rules already bound to the data object are displayed on this tab.

3. Click **Apply Rule**.

   The Apply Data Rule Wizard is displayed.

4. On the Welcome page, click **Next**.

5. On the Select Rule page, select the data rule that you want to apply to the data object and click **Next**.

   Data rules are grouped under the nodes BUILT_IN, DERIVED_DATA_RULES, and any other data rule folders that you create.

   The BUILT_IN node contains the default data rules defined in the workspace. 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.

6. On the Name and Description page, enter a name and an optional description for the applied data rule. Click **Next**.
7. On the Bind Rule Parameters page, use the Binding list to select the column in the data object to which the data rule must be applied. Click **Next**.
8. On the Summary page, review the sections you made on the previous wizard pages. Click **Back** to modify selected values. Click **Finish** to apply the data rule.

   The data rule is bound to the data object and is listed on the Data Rules tab.

Monitoring Data Quality Using Data Auditors

Data auditors are objects that you can use to continuously monitor your source schema to ensure that the data adheres to the defined data rules. You can monitor an object only if you have defined data rules for the object. You can create data auditors for tables, views, materialized views, and external tables.

> **See Also:** "About Data Auditors" on page 23-15

To monitor data quality, perform the following steps:

1. Create a data auditor containing the data objects that you want monitor.

   See "Creating Data Auditors" on page 23-45
2. Run the data auditor to identify the records that do not comply with the data rules defined on the data objects. You can either run data auditors manually or schedule them to run at specified times.

   See "Auditing Data Objects Using Data Auditors" on page 23-47 for information about running data auditors.

> **Note:** You cannot import metadata for data auditors in Merge mode. For more information about import mode options, refer to the *Oracle Warehouse Builder Installation and Administration Guide*.

Creating Data Auditors

Use the Create Data Auditor Wizard to create data auditors. Data auditors are part of an Oracle module in a project.

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.
3. On the Name and Description page, enter a name and an optional description for the data auditor. Click **Next**.
4. On the Select Objects page, select the data objects that you want to audit and click **Next**.

   The Available section lists the objects available for auditing. This contains only objects that have data rules bound to them. The Selected section contains the objects that are selected for auditing. Use the shuttle buttons to move objects to the

Selected section. You can select multiple objects by holding down the Ctrl key while selecting objects.

5. On the Choose Actions page, specify the action to be taken for records that do not comply with data rules bound to the selected objects and click **Next**.

   See "Specifying Data Correction Actions" on page 23-46.

6. On the Summary page, review the selections you made. Click **Back** to modify any selected values or click **Finish** to create the data auditor.

The new data auditor is added to the Data Auditors node. At this stage, only the metadata for the data auditor is stored in your workspace. To use this data auditor to monitor the quality of data in your data objects, you must run the data auditor.

Specifying Data Correction Actions

Use the Choose Actions page of the Create Data Auditor Wizard or the Choose Action tab of the Edit Data Auditor dialog box to specify data correction actions. This page contains two sections: Error threshold mode and Data Rules.

Error threshold mode

Error threshold mode is used to determine the compliance of data to data rules in the objects. Select one of 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 value.
- **Six Sigma:** The data auditor will set the audit result based on the Six Sigma values for the data rules. If the calculated Six Sigma value for any rule is less than the specified Sigma Threshold value, then the data auditor will set the AUDIT RESULT to 2.

Data Rules

The Data Rules section lists the data rules applied to the objects selected on the Select Object page. For each rule, specify the following:

- **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** if you want the data rule to be ignored.
- **Defect Threshold:** The percent of records that should comply with the data rules to ensure successful auditing. Specify a value between 1 and 100. This value is ignored if you select Six Sigma in the Error threshold mode section.
- **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 select Percent in the Error threshold mode section.

Editing Data Auditors

After you create a data auditor, you can edit it and modify any of its properties.

To edit a data auditor:

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

   The Edit Data Auditor dialog box is displayed.

2. On the Name tab, enter a new name or description for the data auditor.

3. On the Select Objects tab, add or remove objects that will be audited as part of the data auditor using the arrows.
4. On the Choose Actions tab, edit the data correction actions you specified.

   See "Specifying Data Correction Actions" on page 23-46.
5. On the Reconcile Objects tab, select the check box to the left of an objects to reconcile its definition with the latest repository definition. Click **Reconcile**.
6. Click **OK** to close the Edit Data Auditor dialog box.

Auditing Data 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

Manually Running Data Auditors

To check if the data in the data object adheres to the data rules defined for the object, you must run the data auditor. 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 window as described in "Data Auditor Execution Results" on page 23-48.

Automatically Running Data Auditors

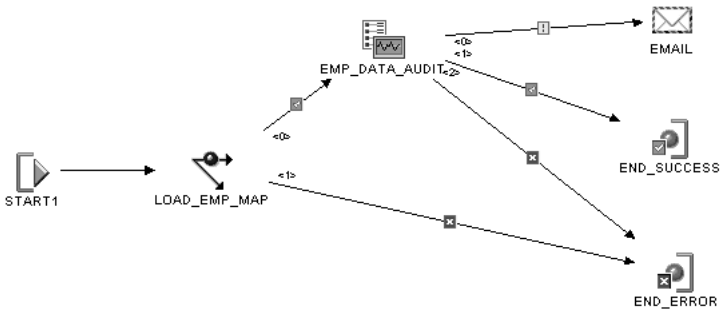
You can automate the process of running a data auditor using the following steps:

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 26-2.

Figure 23–11 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 23–11 Data Auditor Monitor Activity in a Process Flow



The image is described in the surrounding text.

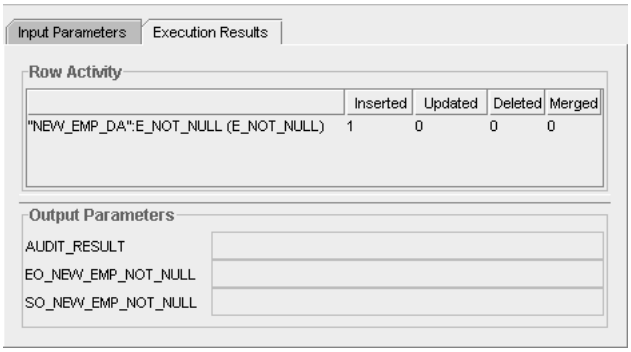
****

Data Auditor Execution Results

After you run a data auditor, the Job Details window displays the details of the execution. The Job Details window contains two tabs: Input Parameters and Execution Results. Note that the Job Details window is displayed only when you set the deployment preference Show Monitor to true. For more information about deployment preferences, see "Deployment Preferences" on page 3-4.

Figure 23–12 displays the Execution Results tab of the Job Details window.

Figure 23–12 Data Auditor Execution Results



This image displays the tabs in the Job Details window. The two tabs are Input Parameters and Execution Results (currently active).

The top portion of this tab contains the Row Activity section. This section contains the following columns: Inserted, Updated, Deleted, and Merged. The bottom portion of this tab contains the Output Parameters section. this section contains the following fields: AUDIT_RESULT, BO_NEW_EMP_NOT_NULL, and SP_NEW_EMP_NOT_NULL.

****

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. 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.

In Figure 23–12, the data rule called E_NOT_NULL inserted one record into the error table.

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.

  2: At least one data rule failed to meet the minimum quality threshold.

  For more information about setting the threshold, see the step on choosing actions in "Creating Data Auditors" on page 23-45.

- **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.

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 box 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 box is displayed.
3. Based on your requirement, configure the parameters listed in Run Time Parameters, Data Auditor Parameters, and Code Generation Options.

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 the execution of this step is terminated.

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 Parameters

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, Oracle SQL syntax is generated.

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. If the target table is not in the same schema as the mapping and you wish to analyze the table, then you would need to grant ANALYZE ANY to the schema owning the mapping.

Optimized Code: Select this option to indicate that optimized code should be generated.

Generation Mode: Select the mode in which optimized code should be generated. The option you can select are: 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.

Viewing Data Auditor Error Tables

When you run a data auditor, either manually or as part of the process flow, records that do not comply with the data rules defined on the objects contained in the data auditor are written to error tables. Each object contained in the data auditor has a corresponding error table that stores non-compliant records for that object.

You can view all non-compliant records that are written to error tables by using the Repository Browser.

To view error tables created as a result of data auditor execution:

1. Grant privileges on the error tables as described in "Granting Privileges on Error Tables" on page 23-51.
2. Use the Repository Browser to view the error tables. Perform the following steps:
   - Open the Repository Browser as described in "Opening the Repository Browser" on page 27-2.
   - View error tables using the Repository Browser as described in "Viewing Error Tables Created as a Result of Data Auditor Execution" on page 27-17.

Granting Privileges on Error Tables

Before you view data stored in error tables using the Repository Browser, you must grant privileges on the error tables to the OWBSYS user. This enables the Repository Browser to access error table data.

To grant privileges on error tables:

1. In SQL*Plus, log in to the schema containing the error tables.

   The error table for an object is stored in the same schema as the object.
2. Run the SQL script *OWB_ORACLE_HOME*`\owb\rtp\sql\grant_error_table_privileges.sql`.
3. When prompted, enter the name of the error table for which you want to grant privileges.

   If you did not specify a name for the error table of an object, using the Error Table Name property, Warehouse Builder provides a default name. For objects that use error tables, the default error table name is the object name suffixed by "_ERR".
4. Repeat steps 2 and 3 for each error table to which you want to grant privileges.

Setting Data Watch and Repair for Oracle Master Data Management (MDM)

Data Watch and Repair (DWR) is a profiling and correction solution designed to assist data governance in Oracle's Master Data Management (MDM) solutions. MDM applications need to provide a single consolidated view of data. To do this, they need to first clean up a system's master data before they can share it with multiple connected entities.

Warehouse Builder provides data profiling and data correction functionality that enables MDM applications to cleanse and consolidate data. You can use DWR for the following MDM applications:

- Customer Data Hub (CDH)
- Product Information Management (PIM)
- Universal Customer Master (UCH)

Overview of Data Watch and Repair (DWR) for MDM

Data Watch and Repair (DWR) enables you to analyze, cleanse, and consolidate data stored in MDM databases using the following:

- Data profiling

  Data profiling is a data analysis method that enables you to detect and measure defects in your source data.

  For more information about data profiling, see "About Data Profiling" on page 23-4.

- Data rules

  Data rules help ensure data quality by determining the legal data and relationships in the source data. You can import MDM-specific data rules, define your own data rules, or derive data rules based on the data profiling results.

  For more information about data rules, see "About Data Rules" on page 23-13.

- Data Correction

  Data correction enables you to correct any inconsistencies, redundancies, and inaccuracies in both the data and metadata. You can automatically create correction mappings to cleanse source data.

  For more information about data correction, see "About Data Correction and Augmentation" on page 23-11.

DWR enables you to measure crucial business rules regularly. As you discover inconsistencies in the data, you can define and apply new data rules to ensure data quality.

Predefined Data Rules for MDM

Warehouse Builder provides a set of data rules that are commonly used in MDM applications. These include the following customer data rules that can be used in both Customer Data Hub (CDH) and Universal Customer Master (UCM) applications:

- Attribute Completeness
- Contact Completeness
- Data Type
- Data Domain
- Restricted Values
- Unique Key Discovery
- Full Name Standardization
- Common Pattern
- Name Capitalization
- Extended Phone Numbers
- International Phone Numbers
- No Access List by Name Only
- No Access List by Name or SSN
- No Email List

For more details about these data rules, refer to the *Oracle Watch and Repair for MDM User's Guide*.

Prerequisites for Performing Data Watch and Repair (DWR)

To use Data Watch and Repair (DWR), you need the following software:

- Oracle Database 11*g* Release 1 (11.1) or later
- One or more of the following Master Data Management (MDM) applications: Customer Data Hub (CDH), Product Information Management (PIM), or Universal Customer Master (UCH).

For MDM applications that run on an Oracle Database, you can directly use DWR. However, for MDM applications that do not run on an Oracle Database, you need to set up a gateway with the third-party database.

Steps to Perform Data Watch and Repair (DWR) Using Warehouse Builder

1. Create a location corresponding to the Master Data Management (MDM) application database.

   Use the Oracle node under the Databases node in the Connection Explorer. Specify the details of the MDM database such as the user name, password, host name, port, service name, and database version.

2. In the Project Explorer, expand the Applications node to display the nodes for the MDM applications.

   The CDH node represents Customer Data Hub application, the PIM node represents the Product Information Management application, and the UCM node represents the Universal Customer Master application.

3. Right-click the node corresponding to the type of MDM application for which you want to perform DWR and select **Create CMI Module**.

   Use the Create Module Wizard to create a module that stores your MDM metadata definitions. Ensure that you select the location you created in step 1 while creating the module.

4. Import metadata from your MDM application into the module created in step 3. Right-click the module and select **Import**.

   The Metadata Import Wizard is displayed. This wizard enables you to import MDM metadata. For more information, see "Using the Import Metadata Wizard" on page 4-9.

5. Import data rules specific to MDM as described in "Importing MDM Data Rules" on page 23-54.

6. Apply data rules to the MDM application tables as described in "Applying Data Rules to Objects" on page 23-44.

   Applying data rules to tables enables you to determine if your table data complies with the business rules specified in the data rules. You can apply data rules you imported in step 5 or other data rules that you created.

   For more information about creating data rules, see "Creating Data Rules" on page 23-43.

7. Create a data profile that contains all tables from the MDM application that you want to profile.

For more information about creating data profiles, see "Create a Data Profile" on page 23-17.

8. Perform data profiling on the MDM application objects as described in "Profile the Data" on page 23-19.
9. View the data profiling results as described in "View Profile Results" on page 23-21.
10. (Optional) Derive data rules based on data profiling results as described in "Derive Data Rules" on page 23-31.

    Data rules derived from data profiling results are automatically applied to the table.
11. Create correction mappings as described in "Steps to Create Corrections" on page 23-33.
12. Correct data and metadata using the correction mappings generated by Warehouse Builder as described in "Steps to Deploy Correction Objects" on page 23-35.
13. Write the corrected data, stored in the correction objects created in step 12, to the MDM application as described in "Writing Corrected Data and Metadata to the MDM Application" on page 23-54.

Importing MDM Data Rules

Data rules required for Customer Data Hub (CDH) and Universal Customer Master (UCM) applications are provided in the *OWB_ORACLE_HOME*/misc/dwr/customer_data_rules.mdl file. To import these data rules, from the Design menu, select **Import**, then **Warehouse Builder Metadata**. In the Metadata Import dialog box, select customer_data_rules.mdl and click **OK**. For more information on using the Metadata Import dialog, click **Help** on this page.

The imported data rules are listed in the Global Explorer, under the MDM Customer Data Rules node of the Public Data Rules node.

Writing Corrected Data and Metadata to the MDM Application

The cleansed and corrected data is contained in the correction objects created as a result of data profiling.

To be more efficient, you can write back only those rows that need to be corrected. You can achieve this by modifying the generated correction mapping. Delete the branch that passes through the compliant rows unchanged (this is the branch that contains the minus filter and the minus set operators). Retain only the corrected rows processing branch in the correction mapping.

Use the following steps to write corrected data to the source MDM application:

1. Create a mapping using the Mapping Editor.
2. Drag and drop the corrected table on to the Mapping Editor. This represents the source table.
3. For UCM, drag and drop the interface table that corresponds to the base table with which you are working.

   Use the MDM application tools and documentation to determine the base table for a particular interface table.
4. Map the columns from the corrected table to the interface table.

5. Deploy and execute the mapping to write corrected data to the source MDM application.
6. Update the base table with changes made to the interface table. You can use Siebel Enterprise Integration Manager (EIM). EIM can be run using the command line or from a Graphical User Interface (GUI).

   For more details about using the EIM, refer to *Siebel Enterprise Integration Manager Administration Guide*.

24

Data Quality Operators

This section discusses the mapping operators that help you achieve data quality. Because reporting on erroneous data wastes time and money, data quality is a key element of Business Intelligence. Include data quality operators in your mappings to load clean, accurate records to your targets.

This section contains the following topics:

- About the Match-Merge Operator
- About the Name and Address Operator
- Using the Match-Merge Operator to Eliminate Duplicate Source Records
- Using the Name and Address Operator to Cleanse Source Data

About the Match-Merge Operator

Duplicate records can obscure your understanding of who your customers and suppliers really are. Eliminating duplicate records is an important activity in the data correction process. The Match-Merge operator enables you to identify matching records and merge them into a single record. You can define the business rules used by the Match-Merge operator to identify records in a table that refer to the same data. 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.

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.

The Match-Merge operator enables you to:

- Use weights to determine matches between records.
- Determine matches using built-in algorithms, including the Jaro-Winkler and edit distance algorithms.
- Cross reference data to track and audit matches.
- Create custom rules combining built-in rules for matching and merging.

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 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 could refer to the same person. You can then 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 24–1 shows records that refer to the same person prior to using the Match-Merge operator.

Table 24–1 Sample Records

Row	First Name	Last Name	SSN	Address	Unit	Zip
1	Jane	Doe	NULL	123 Main Street	NULL	22222
2	Jane	Doe	987-65-4325	NULL	NULL	22222
3	J.	Doe	NULL	123 Main Street	Apt 4	22222
4	NULL	Smith	987-65-4325	123 Main Street	Apt 4	22222
5	Jane	Smith-Doe	987-65-4325	NULL	NULL	22222

Table 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.

First Name	Last Name	SSN	Address	Unit	Zip
Jane	Doe	987-65-4325	123 Main Street	Apt 4	22222

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 :

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 :

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 :

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 :

Row	First Name	Last Name	PHN	SSN
A	John	Doe	650-555-0111	NULL
B	Jonathan	Doe	650-555-0111	987-65-4328
C	John	Dough	650-555-0111	987-65-4328

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 24–2:

Table 24–2 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 24–3:

Table 24–3 Sample Data

Row	First Name	Last Name	PHN	SSN
A	John	Jones	650-555-0110	NULL
B	Jonathan	James	650-555-0110	987-65-4326
C	John	Jamos	650-555-0110	987-65-4326

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).

Restrictions on Using the Match-Merge Operator

- Because the Match-Merge operator only accepts SQL input, you cannot map the output of the Name and Address operator directly to the Match-Merge operator. You must use a staging table.
- Because the Match-Merge generates only PL/SQL, you cannot map the Merge or XREF output groups of the Match-Merge operator to a SQL only operator such as a Sort operator or another Match-Merge operator.

Overview of the Matching and Merging Process

Matching determines which records refer to the same logical data. Warehouse Builder provides a variety of match rules to compare records. Match rules range from an exact match to sophisticated algorithms that can discover and correct common data entry errors.

Merging consolidates matched records into a single record that is free from duplicate records, omissions, misspellings, and unnecessary variations. You can define merge rules to select the preferred data values for use in the consolidated record.

See Also:

- "Match Rules" on page 24-6
- "Merge Rules" on page 24-17

Requirements for Matching and Merging Records

Warehouse Builder uses the following in the matching and merging process.

Match Bins

Match bins are containers for similar records and are used to identify potential matches. The match bin attributes are used to determine how records are grouped into match bins. While performing matching, only records within the same match bin are compared. Match bins limit the number of potential matches in a data set, thus improving performance of the match algorithm.

Match Bin Attributes

Before performing matching, Warehouse Builder divides the source records into smaller groups of similar records. *Match bin attributes* are the source attributes used to determine how records are grouped. Records having the same match bin attributes reside in the same match bin. Match bin attributes also limit match bins to manageable sets.

Select match bin attributes carefully to fulfill the following two conflicting needs:

- Ensure that any records that match reside in the same match bin.
- Keep the size of the match bin as small as possible.

  A small match bin is desirable for efficiency.

Match Record Sets

A match record set consists of one or more similar records. After matching records, a match record set is created for each match bin. You can define the match rules that determine if two records are similar.

Merged Records

A merged record contains data that is merged using multiple records in the match record set. Each match record set generates its own merged record.

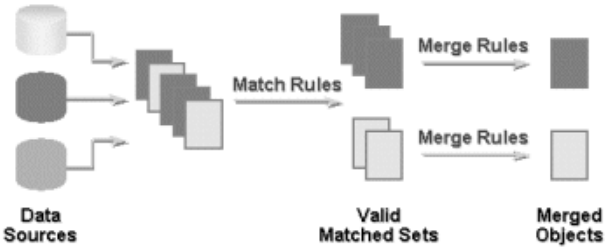
Process for Matching and Merging Records

You use the Match-Merge operator to match and merge records. This operator accepts records from an input source, determines the records that are logically the same, and constructs a new merged record from the matched records.

Figure 24–1 represents high-level tasks involved in the matching and merging process. These include the following:

- Constructing Match Bins
- Constructing Match Record Sets
- Constructing Merge Records

Figure 24–1 Match-Merge Process



This image depicts the Match-Merge process. On the left are three icons that represent the data sources. To the right of the data sources is a set of overlapping rectangles, which are connected to the sources by an arrow. To the right of this are two rectangles, one below the other, representing the valid matched sets. An arrow, with Match Rules written on it, connects to the valid matched sets. To the right of the valid matched sets are two rectangles, one below the other, that represent the merged objects. Arrows connect the rectangle at the top of the valid matched sets to the rectangle at the top of the merged objects and the rectangle at the bottom of the valid matched sets the one at the bottom of the merged objects.

****

Constructing Match Bins

The match bin is constructed using the match bin attributes. Records with the same match bin attribute values will reside in the same match bin. A small match bin is desirable for efficiency.

Constructing Match Record Sets

Match rules are applied to all the records in each match bin to generate one or more match record sets. Match rules determine if two records match. A match rule is an *n* X *n* algorithm where all records in the match bin are compared.

One important point of this algorithm is the transitive matching. Consider three records A, B, and C. If record A is equal to record B and record B is equal to record C, this means that record A is equal to record C.

> **See Also:** "Match Rules" on page 24-6 for information about the types of match rules and how to create them

Constructing Merge Records

A single merge record is constructed from each match record set. You can create specific rules to define merge attributes by using merge rules

> **See Also:** "Merge Rules" on page 24-17 for more information about the types of merge rules

Match Rules

Match rules are used to determine if two records are logically similar. Warehouse Builder enables you to use different types of rules to match source records. You can define match rules using the MatchMerge Wizard or the MatchMerge Editor. Use the editor to edit existing match rules or add new rules.

Match rules can be active or passive. Active rules are generated and executed in the order specified. Passive rules are generated but not executed.

Table 24–4 describes the types of match rules.

Table 24–4 Types of Match Rules

Match Rule	Description
All Match	Matches all rows within a match bin.
None Match	Turns off matching. No rows match within the match bin.
Conditional	Matches rows based on the algorithm you set. For more information about Conditional match rules and how to create one, see "Conditional Match Rules" on page 24-6.
Weight	Matches row based on scores that you assign to the attributes. For more information about Weight match rules and how to create one, see "Weight Match Rules" on page 24-9.
Person	Matches records based on the names of people. For more information about Person match rules and how to create one, see "Person Match Rules" on page 24-10.
Firm	Matches records based on the name of the organization or firm. For more information about Firm match rules and how to create one, see "Firm Match Rules" on page 24-12.
Address	Matches records based on postal addresses. For more information about Address match rules and how to create one, see "Address Match Rules" on page 24-14.
Custom	Matches records based on a custom comparison algorithm that you define. For more information about Custom match rules and how to create one, see "Custom Match Rules" on page 24-16.

Conditional Match Rules

Conditional match rules specify the conditions under which records match.

A conditional match rule allows you to combine multiple attribute comparisons into one composite rule. When more than one attribute is involved in a rule, two records are considered to be a match only if all comparisons are true. Warehouse Builder displays an AND icon in the left-most column of subsequent conditions.

You can specify how attributes are compared using comparison algorithms.

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 24–5 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.

Comparison Algorithms

Each attribute in a conditional match rule is assigned a comparison algorithm, which specifies how the attribute values are compared. Multiple attributes may be compared in one rule with a separate comparison algorithm selected for each.

Table 24–5 describes the types of comparisons.

Table 24–5 Types of Comparison Algorithms for Conditional Match Rules

Algorithm	Description
Exact	Attributes match if their values are exactly the same. For example, "Dog" and "dog!" would not match, because the second string is not capitalized and contains an extra character. For data types other than `STRING`, this is the only type of comparison allowed.
Standardized Exact	Standardizes the values of the attributes before comparing for an exact match. With standardization, the comparison ignores case, spaces, and non-alphanumeric characters. Using this algorithm, "Dog" and "dog!" would match.
Soundex	Converts the data to a Soundex representation and then compares the text strings. If the Soundex representations match, then the two attribute values are considered matched.
Edit Distance	A "similarity score" in the range 0-100 is entered. If the similarity of the two attributes is equal or greater to the specified value, the attribute values are considered matched. The similarity algorithm computes the edit distance between two strings. A value of 100 indicates that the two values 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	Standardizes the values of the attribute before using the Similarity algorithm to determine a match. With standardization, the comparison ignores case, spaces, and non-alphanumeric characters.
Partial Name	The values of a string attribute are considered a match if the value of one entire attribute is contained within the other, starting with the first word. For example, "Midtown Power" would match "Midtown Power and Light", but would not match "Northern Midtown Power". The comparison ignores case and non-alphanumeric characters.

Table 24–5 (Cont.) Types of Comparison Algorithms for Conditional Match Rules

Algorithm	Description
Abbreviation	The values of a string attribute are considered a match if one string contains words that are abbreviations of corresponding words in the other. Before attempting to find an abbreviation, this algorithm performs a Std Exact comparison on the entire string. The comparison ignores case and non-alphanumeric character. For each word, the match rule will look for abbreviations, as follows. If the larger of the words being compared 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" would match "International Bus Prd".
Acronym	The values of a string attribute are considered a match if one string is an acronym for the other. Before attempting to identify an acronym, this algorithm performs a Std Exact 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 are considered a match. For example, "Chase Manhattan Bank NA" matches "CMB North America". The comparison ignores case and non-alphanumeric characters.
Jaro-Wrinkler	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.
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.

Creating Conditional Match Rules

To define a conditional match rule, complete the following steps:

1. On the top portion of the Match Rules tab or the Match Rules page, select **Conditional** in the Rule Type column.

   A Details section is displayed.

2. Click **Add** to add a new row.

3. Select an attribute in the Attribute column.
4. In the Algorithm column, select a comparison algorithm. See Table 24–5 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.

Weight Match Rules

A weighted match rule allows you to assign an integer weight to each attribute included in the rule. You must also specify a threshold. For each attribute, the Match-Merge operator multiplies the weight by the similarity score, and sums the scores. If the sum equals or exceeds the threshold, the two records being compared are considered a match.

Weight match rules are most useful when you need to compare a large number of attributes, without having a single attribute that is different causing a non-match, as can happen with conditional rules.

Weight rules implicitly invoke the similarity algorithm to compare two attribute values. This algorithm returns an integer, percentage value in the range 0-100, which represents the degree to which two values are alike. A value of 100 indicates that the two values are identical; a value of zero indicates no similarity whatsoever.

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.

Example of Weight Match Rules

Table 24–6 displays the attribute values contained in two separate records that are read in the following order.

Table 24–6 Example of Weight Match Rule

Record Number	First Name	Middle Name	Last Name
Record 1	Robert	Steve	Paul
Record 2		Steven	Paul

You define a match rule that uses the Edit Distance similarity algorithm. The Required Score to Match is 120. The attributes for first name and middle name are defined with a Maximum Score of 50 and Score When Blank of 20. The attribute for last name has a Maximum Score of 80 and a Score When Blank of 0.

Consider an example of the comparison of Record 1 and Record 2 using the weight match rule.

- Since first name is blank for Record 2, the Blank Score = 20.
- The similarity of middle name in the two records is 0.83. Since the weight assigned to this attribute is 50, the similarity score for this attribute is 43 (0.83 X 50).
- Since the last name attributes are the same, the similarity score for the last name is 1. The weighted score is 80 (1 X 80).

The total score for this comparison is 143 (20+43+80). Since this is more than the value defined for Required Score to Match, the records are considered a match.

Creating Weight Match Rules

To use the Weight match rule, complete the following steps:

1. On the Match Rules tab or the Match Rules page, 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 using the Attribute column.
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 **Score When Blank**, assign a value to be used when the attribute is blank in one of the records.
6. 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 value specified in the Required score to match parameter.

Person Match Rules

Built-in Person rules provide an easy and convenient way for matching names of individuals. Person match rules are most effective when the data has first been corrected using the Name and Address operator.

When you use Person match rules, you must specify which data within the record represents the name of the person. The data can come from multiple columns. Each column must be assigned an input role that specifies what the data represents.

To define a Person match rule, you must define the Person Attributes that are part of the rule. For example, you can create a Person match rule that uses the Person Attributes first name and last name for comparison. For each Person Attribute, you

must define the Person Role that the attribute uses. Next you define the rule options used for the comparison. For example, while comparing last names, you can specify that hyphenated last names should be considered a match.

Person Roles

Table 24–7 describes the roles for different parts of a name that are used for matching. On the Match Rules page or Match Rules tab, use the Roles column on the Person Attributes tab to define person details.

Table 24–7 Name Roles for Person Match Rules

Role	Description
Prename	Prenames are compared only if the following are true: ■ The Last_name and, if present, the middle name (Middle_name_std, Middle_name_2_std, and Middle_name_3_std roles) in both records match. ■ The "Mrs. Match" option is selected. ■ Either record has a missing First_name_std.
First Name Standardized	Compares the first names. By default, the first names must match exactly, but you can specify other comparison options as well. 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. If a Last_name role has not been assigned, a role of First_name_std must be assigned.
Middle Name Standardized, Middle Name 2 Standardized, Middle Name 3 Standardized	Compares the middle names. By default, the middle names must match exactly, but other comparison options can be specified. If more than one middle name role is assigned, attributes assigned to the different roles are cross-compared. For example, values for Middle_name_std will be compared not only against other Middle_name_std values, but also against Middle_name_2_std, if that role is also assigned. Middle names match if either or both are blank. If any of the middle name roles are assigned, the First_name_std role must also be assigned.
Last Name	Compares the last names. By default, the last names must match exactly, but you can specify other comparison options. The last names match if both are blank, but not if only one is blank.
Maturity Post Name	Compares the post name, such as "Jr.", "III," and so on. The post names match if the values are exactly the same, or if either value is blank.

Person Details

Table 24–8 describes the options that determine a match for person match rules. Use the Details tab of the Match Rules tab or the Match Rules page to define person details.

Table 24–8 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.

Table 24–8 (Cont.) Options for Person Match Rule

Option	Description
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	Records are considered a match if the similarity is greater than or equal to score. For example, "Susan" will match "Susen" if the score is less than or equal to 80. Uses a similarity score to determine a match, as calculated by the Edit Distance or Jaro-Winkler algorithms. 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	Matches hyphenated names to unhyphenated names such as "Reese-Jones" to "Reese". You can select this option for the last name role.
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.

Creating Person Match Rules

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 right arrow to move them to Name Roles section.

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 24–7 for the types of roles you can assign.

4. Select the Details tab and select the applicable options as listed in Table 24–8.

Firm Match Rules

Built-in Firm match rules provide an easy and convenient way for matching business names. Firm match rules are most effective when the data has first been corrected using the Name and Address operator. Similar to the Person rule, this rule requires users to set what data within the record represents the name of the firm. The data can come from multiple columns and each column specified must be assigned an input role that indicates what the data represents.

Note that you need not assign a firm role to every attribute, and not every role needs to be assigned to an attribute. The attributes assigned to firm roles are used in the match rule to compare the records. The attributes are compared based on the role they have been assigned and other comparison options have you set. For a complete list of

firm roles and how each role is treated in a firm match rule, see "Firm Roles" on page 24-13.

Firm Roles

Firm roles define the parts of a firm name that are used for matching. The options you can select for firm role are Firm1 or Firm2. If you select one attribute, for firm name, select Firm1 as the role. If you selected two attributes, designate one of them as Firm1 and the other as Firm2.

- **Firm1:** If this role is assigned, the business names represented by Firm1 are compared. Firm1 names will not be compared against Firm2 names unless if the Cross-match firm1 and firm2 box is checked. By default, the firm names must match exactly; but other comparison options can also be specified. Firm1 names do not match if either or both names are blank.
- **Firm2:** If this role is assigned, the values of the attribute assigned to Firm2 will be compared. Firm2 names will not be compared against Firm1 names unless if the Cross-match firm1 and firm2 box is checked. By default, the firm names must match exactly; but other comparison options can also be specified. Firm2 names do not match if either or both names are blank. If a Firm1 role is not assigned, a Firm2 roles must be assigned.

Firm Details

Table 24–9 describes the rule options you can set for each component of the firm name to determine a match.

Table 24–9 Options for Firm Rules

Option	Description
Strip noise words	Removes the following words from Firm1 and Firm2 before matching: THE, AND, CORP, CORPORATION, CO, COMPANY, INC, INCORPORATED, LTD, TO, OF, and BY.
Cross-match firm 1 and firm 2	When comparing two records for matching, in addition to matching firm1 to firm1 and firm2 to firm2 of the respective records, match firm1 against firm2 for the records.
Match on partial firm name	Uses the Partial Name algorithm to determine a match. For example, match "Midtown Power" to "Midtown Power and Light".
Match on abbreviations	Uses the Abbreviation algorithm to determine a match. For example, match "International Business Machines" to "IBM".
Match on acronyms	Uses the Acronym algorithm to determine a match. For example, match "CMB, North America" to "Chase Manhattan Bank, NA".
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. Two records are considered as a match if the similarity is greater than or equal to the value of similarity score.

Creating Firm Match Rules

To define a Firm match rule, complete the following steps:

1. On the Match Rules tab or the Match Rules page, 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 are moved to the Firm Roles box.

3. For each attribute, click **Roles**. 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. For more details, see "Firm Details" on page 24-13.

Address Match Rules

Address Match rules provide a method of matching records based on postal addresses. Address match rules are most effective when the data has first been corrected using a Name and Address operator.

Address Rules work differently depending on whether the address being processed has been corrected, using the Name and Address operator, or not. Generally, corrected addresses have already been identified in a postal matching database, and are therefore syntactically correct, legal, and existing addresses according to the Postal Service of the country containing the address. Corrected addresses can be processed more quickly, since the match rule can make certain assumptions about their format.

Uncorrected addresses may be syntactically correct, but have not been found in a postal matching database. Addresses may have not been found because they are not in the database, or because there is no postal matching database installed for the country containing the address. Address match rules determine whether an address has been corrected based on the Is_found role. If Is_found role is not assigned, then the match rule performs the comparisons for both the corrected and uncorrected addresses.

To create an Address match rule, assign address roles to the various attributes. The attributes assigned to address roles are used in the match rule to compare the records. Attributes are compared depending on which role they have been assigned, and what other comparison options have been set.

Address Roles

Table 24–10 describes the address roles you can select for each part of an address.

Table 24–10 Address Roles

Role	Description
Primary Address	Compares the primary addresses. Primary addresses can be, for example, street addresses ("100 Main Street") or PO boxes ("PO Box 100"). By default, the primary addresses must match exactly, but a similarity option can also be specified. The Primary_address role must be assigned.
Unit Number	Unit numbers (such as suite numbers, floor numbers, or apartment numbers) are compared if the primary addresses match. The unit numbers match if both are blank, but not if one is blank, unless the Match on blank secondary address option is set. If the Allow differing secondary address is set, the unit numbers are ignored.
PO Box	Compares the Post Office Boxes. The PO Box is just the number portion of the PO Box ("100"), and is a subset of the primary address, when the primary address represents a PO Box ("PO Box 100"). If the primary address represents a street address, the PO Box will be blank.

Table 24–10 (Cont.) Address Roles

Role	Description
Dual Primary Address	The Dual_primary_address is compared against the other record's Dual_primary_address and Primary_address to determine a match.
Dual Unit Number	Compares the Dual_unit_number address with the Dual_unit_number and Unit_number of the other record. The unit numbers will match if one or both are blank. To assign the Dual_unit_number role, the Dual_primary_address role must also be assigned.
Dual PO Box	Dual_PO_Box address of a record is compared with the Dual_PO_Box and the PO_Box of the other record. To assign the Dual_PO_Box role, the Dual_primary_address role must also be assigned.
City	Compares the cities for uncorrected addresses. For corrected addresses, the cities are only compared if the postal codes do not match. If both City and State roles match, then the address line roles, such as Primary_address, can be compared. By default, the cities must match exactly. But you may specify a last line similarity option. The cities match if both are blank, but not if only one is blank. If the City role is assigned, then the State role must also be assigned.
State	Assign this role only when also assigning the City role. The states are compared for uncorrected addresses. For corrected addresses, the states are only compared if the postal codes do not match. If both State and City roles match, then the address line roles, such as Primary_address, can be compared. By default, the states must match exactly, but a last line similarity option may be specified. The states match if both are blank, but not if only one is blank. If the State role is assigned, then the City role must also be assigned.
Postal Code	For uncorrected address data, the operator does not use Postal Code. The postal codes are compared for corrected addresses. For uncorrected addresses, the Postal_code role is not used. To match, the postal codes must be exactly the same. The postal codes are not considered a match if one or both are blank. If the postal codes match, then the address line roles, such as Primary_address, can be compared. If the postal codes do not match, City and State roles are compared to determine whether the address line roles should be compared.
Is Found	The Is_found_flag attributes are not compared, but instead are used to determine whether an address has been found in a postal matching database, and therefore represents a legal address according to the postal service of the country containing the address. This determination is important because the type of comparison done during matching depends on whether the address has been found in the postal database or not.

Address Details

Table 24–11 describes the options for determining a match for an address rule.

Table 24–11 Options for Address Roles

Option	Description
Allow differing secondary address	Allow addresses to match even if the unit numbers are not null and are different.
Match on blank secondary address	Allow addresses to match even if exactly one unit number is null.
Match on either street or post office box	Matches records if either the street address or the post office box match.

Table 24–11 (Cont.) Options for Address Roles

Option	Description
Address line similarity	Match if address line similarity >= the score. All spaces and non-alpanumeric characters are removed before the similarity is calculated.
Last line similarity	Match is the last line similarity >= score. The last line consists of city and state. All spaces and non-alphanumeric characters are removed before the similarity is calculated.

Creating Address Match Rules

To define an Address match rule, complete the following steps:

1. On the Match Rules tab or the Match Rules page, 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 designate one attribute as the primary address. 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 24–10 for the types of roles you can assign.

5. Select the Details tab and select the applicable options as listed in Table 24–11.

Custom Match Rules

Custom match rules enable you to write your own comparison algorithms to match records. You can use any input attributes or match functions within this comparison. You can use an active custom rule to control the execution of passive rules.

Consider the following three passive built-in rules:

- `NAME_MATCH`: built-in name rule.
- `ADDRESS_MATCH`: built-in address rule.
- `TN_MATCH`: built-in conditional rule.

You can create a custom rule to specify that two records can be considered a match if any two of these rules are satisfied. Example 5–1 describes the PL/SQL code used to create the custom match rule that implements this example.

Example 24–1 Creating a Custom Rule Using Existing Passive Rules

```
BEGIN
   RETURN(
       (NAME_MATCH(THIS_,THAT_) AND ADDRESS_MATCH(THIS_,THAT_))
     OR
       (NAME_MATCH(THIS_,THAT_) AND TN_MATCH(THIS_,THAT_))
     OR
       (ADDRESS_MATCH(THIS_,THAT_) AND TN_MATCH(THIS_,THAT_))
         );
END;
```

Creating Custom Match Rules

To define a Custom match rule, complete the following steps:

1. On the Match Rules tab or the Match Rules page, 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.

   For more information about using the editor, select **Help Topic** from the Help menu.

3. To enter PL/SQL code, use any combination of the following:

   - To read in a file, select **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, select **Validate** from the Test menu.

   The validation results appear on the Messages tab.

5. To save your code, select **Save** from the Code menu.

6. To close the Custom Match Rules Editor, select **Close** from the Code menu.

Merge Rules

Matching produces a set of records that are logically the same. Merging is the process of creating one record from the set of matched records. A Merge rule is applied to attributes in the matched record set to obtain a single value for the attribute in the merged record.

You can define one Merge rule for all the attributes in the Merge 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 describes the types of merge rules.

Merge Rule	Description
Any	Uses the first non-blank value.
Match ID	Merges records that have already been output from another Match-Merge operator.
Rank	Ranks the records from the match set. The associated attribute from the highest ranked record will be used to populate the merge attribute value.
Sequence	Specify a database sequence for this rule. The next value of the sequence will be used for the value.
Min Max	Specify an attribute and a relation to choose the record to be used as a source for the merge attribute.

Merge Rule	Description
Copy	Choose a value from a different previously merged value.
Custom	Create a PL/SQL package function to select the merge value. The operator will provide the signature of this function. The user is responsible for the implementation of the rule from "BEGIN" to "END;" The matched records and merge record are parameters for this function.
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, see "Using Two Match-Merge Operators" on page 24-41.

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.

About the Name and Address Operator

After matching and merging records, you can further validate information about your customers and suppliers, and discover additional errors and inconsistencies. Warehouse Builder parses the names and addresses, and uses methods specific to this type of data, such as matching common nicknames and abbreviations. You can compare the input data to the data libraries supplied by third-party name and address cleansing software vendors, which can augment your records with information such as postal routes and geographic coordinates.

Successful delivery and lower postage rates are not the only reasons to cleanse name and address data. You will get better results from data analysis when the results are not skewed by duplicate records and incomplete information.

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.

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, postal code, 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 24-24.

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 24–12.

Table 24–12 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 Table 24–12.
   - A Name and Address operator. This action starts 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 18-35.
- 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.

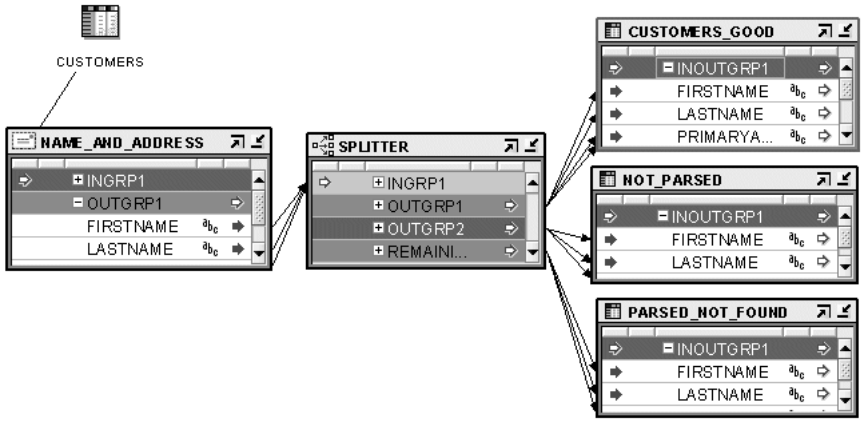
2. Map the attributes from the `CUSTOMERS` table to the Name and Address operator ingroup. Map the attributes from the Name and Address operator outgroup to the Splitter operator ingroup.

   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 24–2 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 24–2 Name and Address Operator Used with a Splitter Operator in a Mapping



This screenshot shows a mapping that contains operators. At the top is the CUSTOMERS table. Below this table, on the left of the image, is the NAME_AND_ ADDRESS operator. There is an arrow from CUSTOMERS to NAME_AND_ ADDRESS. To the right of NAME_AND_ADDRESS is the SPLITTER operator that contains the following: INGRP1, OUTGRP1, OUTGRP2, and REMAINING_ROWS. There is an arrow from the columns FIRSTNAME and LASTNAME in the NAME_ AND_ADDRESS operator to the INDRP1 of the SPLITTER operator. To the right of the SPLITTER operator, are three operators, CUSTOMERS_GOOD, NOT_PARSED, and PARSED_NOT_FOUND. There is an arrow from OUTGRP1 in SPLITTER to CUSOMERS_GOOD, OUTGRP2 in SPLITTER to NOT_PARSED, and REMAINING_ ROWS to PARSED_NOT_FOUND.

****

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 24–13. Compare it with the input record from Table 24–12 on page 24-21.

Table 24–13 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.

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 enables 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.

Input Role Descriptions

For each attribute you select for Name or Address cleansing, you must specify an input role that indicates the type of data that is stored in the source attribute. Warehouse Builder provides a set of predefined input roles from which you can select the most suitable one for your data.

For example, the Employees table contains the columns last_name and city. You can select the Last Name and City respectively for these columns.

Table 24–14 describes the input roles for the Name and Address Operator.

Table 24–14 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.
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.

Table 24–14 (Cont.) Name and Address Operator Input Roles

Input Role	Description
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.
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.

Descriptions of Output Components

Use output components to define attributes that will store data cleansed by the Name and Address operator. Any attributes with an input role of Pass Through are automatically displayed as output components. You can define additional output components to store cleansed data.

Categories 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 24–15 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 24–15 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.

Table 24–15 (Cont.) Name Output Components

Subfolder	Output Component	Description
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.
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 24–16 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 24–16 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.
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.

Table 24–16 (Cont.) Address Output Components

Subfolder	Output Component	Description
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.
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.

Table 24–16 (Cont.) Address Output Components

Subfolder	Output Component	Description
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.
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 box 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.

Table 24–16 (Cont.) Address Output Components

Subfolder	Output Component	Description
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 24–17 describes the Error Status output components. Refer to "Handling Errors in Name and Address Data" on page 24-37 for usages notes on the Error Status components.

Table 24–17 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.

Table 24–17 (Cont.) Error Status Output Components

Subfolders	Output Component	Description
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.
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.

Table 24–17 (Cont.) Error Status Output Components

Subfolders	Output Component	Description
Address Only: Is Found	Non-ambiguous 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.
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.

Table 24–17 (Cont.) Error Status Output Components

Subfolders	Output Component	Description
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
Address Only	Parsing Country	Country parser that was used for the final parse of the record.

Country-Specific

Table 24–18 describes the output components that are specific to a particular country.

Table 24–18 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.

Table 24–18 (Cont.) Country-Specific Output Components

Subfolder	Output Component	Description
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.
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`

Table 24–18 (Cont.) Country-Specific Output Components

Subfolder	Output Component	Description

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`

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.

Using the Match-Merge Operator to Eliminate Duplicate Source Records

Use the Match-Merge operator to identify matching records in a data source and merge them into a single record.

The Match-Merge operator has one input group and two output groups, Merge and Xref. The source data is mapped to the input group. The Merge group contains records that have been merged after the matching process is complete. The Xref group provides a record of the merge process. Every record in the input group will have a

corresponding record in the Xref group. This record may contain the original attribute values and the merged attributes.

The Match-Merge operator uses an ordered record stream as input. From this stream, it constructs the match bins. From each match bin, matched sets are constructed. From each matched set, a merged record is created. The initial query will contain an `ORDER BY` clause consisting of the match bin attributes.

Steps to Use a Match-Merge Operator

To match and merge source data using the Match-Merge operator:

1. Drag and drop the operators representing the source data and the operator representing the merged data onto the mapping editor canvas:

   For example, if your source data is stored in a table, and the merged data will be stored in another table, drag and drop two Table operators that are bound to the tables onto the canvas.

2. Drag and drop a Match-Merge operator onto the mapping editor canvas.

   The MatchMerge wizard is displayed.

3. On the Name and Address page, the Name field contains a default name for the operator. You can change this name or accept the default name.

   You can enter an optional description for the operator.

4. On the Groups page, you can rename groups or provide descriptions for them.

   This page contains the following three groups:

   - **INGRP1:** Contains input attributes.
   - **MERGE:** Contains the merged records (usually this means fewer records than INGRP1).
   - **XREF:** Contains the link between the original and merged data sets. This is the tracking mechanism used when a merge is performed.

5. On the Input Connections page, move the attributes that you want to match and merge form the Available section to the Mapped Attributes section. Click **Next**.

   The Available Attributes section of this page displays nodes for each operator on the canvas. Expand a node to display the attributes contained in the operator, select the attributes, and use the shuttle arrows to move selected attributes to the Mapped Attributes section.

   Note: The Match-Merge operator requires an ordered input data set. If you have source data from more than one operators, use a Set Operation operator to combine the data and obtain an ordered data set.

6. On the Input Attributes page, review the attribute data types and lengths.

   In general, if you go through the wizard, you need not change any of these values. Warehouse Builder populates them based on the output attributes.

7. On the Merge Output page, select the attributes to be merged from the input attributes.

These attributes appear in the Merge output group (the cleansed group). The attributes in this group retain the name and properties of the input attributes.

8. On the Cross Reference Output page, select attributes for the XREF output group.

   The Source Attributes section contains all the input attributes and the Merge attributes you selected on the Merge Output page. The attributes from the Merge group are prefixed with MM. The other attributes define the unmodified input attribute values. Ensure that you select at least one attribute from the Merge group that will provide a link between the input and Merge groups.

9. On the Match Bins page, specify the match bin attributes. These attributes are used to group source data into match bins.

   After the first deployment, you can choose whether to match and merge all records or only new records. To match and merge only the new records, select **Match New Records Only**.

   You must designate a condition that identifies new records. The match-merge operator treats the new records in the following way:

   - No matching is performed for any records in a match bin unless the match bin contains new record.
   - Old records will not be compared with each other.
   - A matched record set will not be presented to the merge processing unless the matched record set contains a new record.
   - An old record will not be presented to the Xref output unless the record is matched to a new record.

   For more information about match bin attributes and match bins, see "Overview of the Matching and Merging Process" on page 24-4.

10. On the Define Match Rules page, define the match rules that will be used to match the source data.

    Match rules can be active or passive. A passive match rule is generated but not automatically invoked. You must define at least one active match rule.

    For more information about the match rules, the types of match rules you can define, and the steps used to define them, see "Match Rules" on page 24-6.

11. On the Merge Rules page, define the rules that will be used to merge the sets of matched records created from the source data.

    You can define Merge rules for each attribute in a record or for the entire record. Warehouse Builder provides different types of Merge rules.

    For more information about the type of Merge rules and the steps to create Merge rules, see "Merge Rules" on page 24-17.

12. On the Summary page, review your selections. Click **Back** to modify any selection you made. Click **Next** to complete creating the Match-Merge operator.

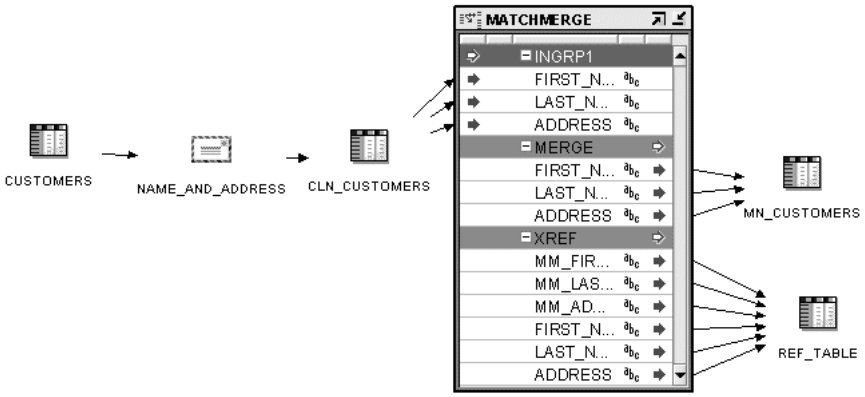
13. Map the Merge group of the Match-Merge operator to the input group of the operator that stores the merged data.

Designing Mappings with a Match-Merge Operator

Figure 24–3 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 starting time consuming match and merge operations

Figure 24–3 Match-Merge Operator in a Mapping



The Customers table provides input to the Name and Address Operator, which stores its output in the CLN_CUSTOMERS table. The CLN_CUSTOMERS table provides FIRST, LAST, and ADDRESS input to the Match-Merge Operator. The Match-Merge operator provides FIRST, LAST, and ADDRESS input to the MM_CUSTOMERS table. It provides FIRST, LAST, ADDRESS, MM_FIRST, MM_LAST, and MM_ADDRESS input to the REF_TABLE table.

****

Whether you include a Name and Address operator or not, be aware of the following considerations as you design your mapping:

- **Operating modes:** 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.
- **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 24-41. 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" on page 24-41.

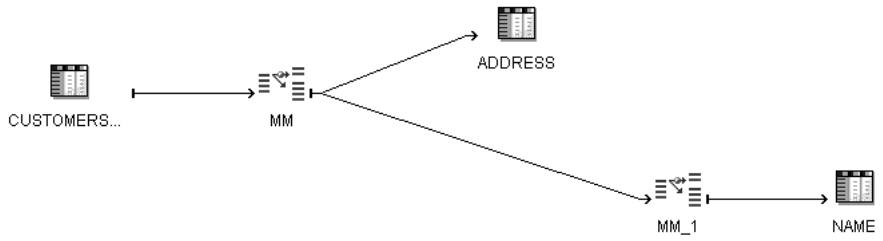
Using Two Match-Merge Operators

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 24–4 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 24–4 Householding Data: XREF Group Merged to Second Match-Merge Operator



This image displays a mapping that uses two Match-Merge operators. The XREF group of the first Match-Merge operator is mapped to a second Match-Merge operator.

****

Using the Name and Address Operator to Cleanse Source Data

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, see "Managing the Name and Address Server" on page 24-45.

Creating a Mapping with a Name and Address Operator

The Name and Address operator has one input group and one output group.

To create a mapping with a Name and Address operator:

1. Drag and drop the operators representing the source data and the operator representing the cleansed data onto the mapping editor canvas:

   For example, if your source data is stored in a table, and the cleansed data will be stored in another table, drag and drop two Table operators that are bound to the tables onto the canvas.

2. Drag and drop a Name and Address operator onto the mapping editor canvas.

   The Name and Address wizard is displayed.

3. On the Name page, specify a name and an optional description for the Name and Address operator.

   Alternately, you can choose to retain the default name displayed in the Name field.

4. On the Definitions page, select values that define the type of source data.

   See "Specifying Source Data Details and Setting Parsing Type" on page 24-43.

5. On the Groups page, optionally rename the input and output groups.

   The Name and Address operator has one input group, INGRP1, and one output group, OUTGRP1. You cannot edit, add, or delete groups. If the input data requires multiple groups, create a separate Name and Address operator for each group.

6. On the Input Connections page, select attributes from any operator in your mapping that you want to copy and map to the Name and Address operator.

   To complete the Input Connections page for an operator:

   a. 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 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.

      Note: If you have not created any operators for the source data, the Available Attributes section is empty.

      ---

   b. Use the right arrow button between the two panels to move your selections to the Mapped Attributes panel.

      The Mapped Attributes section lists the attributes that will be processed by the Name and Address operator.

7. On the Input Attributes page, assign input roles to each attribute you selected on the Input Attributes page.

   Input roles indicate the type of name and address information that resides in a line of data. Whenever possible, choose discrete roles (such as City, state, and Postal Code) rather than non-discrete ones (such as Last Line). Discrete roles help in better parsing.

See Also: "Input Role Descriptions" on page 24-25.

For attributes that have the input role set to Pass Through, specify the data type details using the Data Type, Length, Precision, Scale, and Seconds Precision fields.

8. On the Output Attributes page, define output attributes that determine how the Name and Address operator handles parsed data. The output attribute properties characterize the data extracted from the parser output.

   Any attributes that have the Pass Through input role assigned are automatically listed as output attributes. You can add additional output attributes.

   Note: The attributes for output components with the Pass Through role cannot be changed

   To add output attributes:

   a. Click **Add**.

      A row is added for the new output attribute with a default name. You can rename the output attribute by selecting the name and typing the new name.

   b. Click the Ellipses on the Output Component field to select an output component for the attribute.

      See Also: "Descriptions of Output Components" on page 24-27 for the descriptions of output components

      Ensure that you add error handling flags such as Is Parsed, Is Good Name, and Is Good Address. You can use these flags with the Splitter operator to separate good records from the records with errors and load them into different targets.

   c. Specify the data type details for the output attribute using the Data Type, Length, Precision, Scale, and Seconds Precision fields.

9. For countries that support address correction and postal matching, use the Postal Report page to specify the details for the postal report.

   See "Specifying Postal Report Details" on page 24-44.

Specifying Source Data Details and Setting Parsing Type

Use the Definitions page or the Definitions tab to provide information about your source data and to specify the type of parsing to be performed on the source data. Set the following values: Parsing Type, Primary Country, and Dual Address Assignment.

Parsing Type Select one of the following parsing types:

- **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.

> **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.

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.

Specifying Postal Report Details

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 24-24.

To specify postal reporting, select Yes in the Postal Report files and then provide values for the fields:

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.

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 you specified when installing the server components. 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`.

25

Deploying to Target Schemas and Executing ETL Logic

Oracle Warehouse Builder provides functionality that supports a single logical model and multiple physical models. This enables you to design your data warehouse once and implement this design on multiple target systems. In addition to this, Warehouse Builder also supports multiple physically different implementations of the same object definitions.

This chapter describes the implementation environment in Warehouse Builder. It also describes how to create and use schedules to automate the execution of ETL logic.

This chapter contains the following topics:

- About Deployment and Execution in Warehouse Builder
- The Deployment and Execution Process
- Example: Updating a Target Schema

About Deployment and Execution in Warehouse Builder

After you design your data warehouse, you must implement this design in the target schema by deploying and executing design objects. The Control Center Manager offers a comprehensive deployment console that enables you to view and manage all aspects of deployment and execution. It provides access to the information stored in the active Control Center.

About Deployment

Deployment is the process of creating physical objects in a target location from the logical objects in a Warehouse Builder workspace.

The data objects created when you designed the target schema are logical definitions. Warehouse Builder stores the metadata for these data objects in the workspace. To create these objects physically on the target schema, you must deploy these objects. For example, when you create a table using the Design Center, the metadata for this table is stored in the workspace. To physically create this table in the target schema, you must deploy this table to the target schema. Use the Design Center or the Control Center Manager to deploy objects.

> **Note:** Whenever you deploy an object, Warehouse Builder automatically saves all changes to all design objects to the workspace. You can choose to display a warning message by selecting **Prompt for commit** on the Preferences dialog box.

Deploying a mapping or a process flow includes these steps:

- Generate the PL/SQL, SQL*Loader, or ABAP script, if necessary.
- Register the required locations and deploy any required connectors. This ensures that the details of the physical locations and their connectors are available at runtime.
- Transfer the PL/SQL, XPDL, SQL*Loader, or ABAP scripts from the Design Center to the Control Center.

After deploying a mapping or a process flow, you must explicitly start the scripts, as described in "Starting ETL Jobs" on page 25-8.

You can deploy only those objects for which you have the `COMPILE` privilege. By default, you have this privilege on all objects in the workspace. However, the workspace 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.

Deployment Actions

As soon as you define a new object in the Design Center, the object is listed in the Control Center Manager under its deployment location. Each object has a default deployment action, which you can display. The default deployment action for an object is based on a comparison of its current design status to its current deployment status. For example, a table that has not been previously deployed will have a default deployment action of Create. A table that was previously deployed will have a default action of Upgrade. You can override the default by choosing a different deployment action in the Control Center Manager.

The default is set by the previous action and varies depending on the type of object.

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. For example, this may happen if the object has not been previously deployed from Warehouse Builder.
- **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 re-creates the object. This action is quicker than Upgrade, but it deletes all data.

Deployment Status

After you deploy an object, Warehouse Builder assigns a deployment status to it. The status represents the result of the deployment. You can view the deployment status in the Control Center Manager.

The deployment status can be one of the following:

- **Not Deployed:** Indicates that the object has not yet been deployed to the target schema.
- **Success:** Indicates that the object has been successfully deployed to the target schema.
- **Warning:** Indicates that some warnings were generated during the deployment of the object. Double-click the status to view details about the warning.
- **Failed:** Indicates that deployment of the object failed. Double-click the status to view detailed information about why the deployment failed.

About Execution

For objects that contain ETL logic such as mappings, process flows, and transformations, there is an additional step of execution. Execution is the process of executing the ETL logic defined in the deployed objects.

For example, you define a mapping that sources data from a table, performs transformations on the source data, and loads it into the target table. When you deploy this mapping, the PL/SQL code generated for this mapping is stored in the target schema. When you execute this mapping, the ETL logic is executed and the data is picked up from the source table, transformed, and loaded into the target table.

The Deployment and Execution Process

During the lifecycle of a data system, you typically will take these steps in the deployment process to create your system and the execution process to move data into your system:

1. Select a named 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.

   For information about deploying objects, see "Deploying Objects" on page 25-4.
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.

   For information about starting the ETL process, see "Starting ETL Jobs" on page 25-8.
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 workspace before deployment.

Deploying Objects

Deployment is the process of creating physical objects in a target location from the metadata using your generated code. As part of the deployment process, Warehouse Builder validates and generates the scripts for the object, transfers the scripts to the Control Center, and then invokes the scripts against the deployment action associated with the object. You can deploy an object from the Project Explorer or using the Control Center Manager.

Deployment from the Project Explorer is restricted to the default action, which may be set to Create, Replace, Drop, or Update. The default action is determined by changes to the object design since it was last deployed. To override the default action, use the Control Center Manager, which provides full control over the deployment process. The Control Center Manager also validates and generates 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 from the Project Explorer:

Select the object and click the Deploy icon on the toolbar. You can also select the object, and then choose **Deploy** from the Design menu.

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 deploy from the Control Center Manager:

1. Open a project.
2. Select **Control Center Manager** from the Tools menu.

   The Control Center Manager that provides access to the control center for the active configuration of the project is displayed. If this menu choice is not available, then check that the appropriate named configuration and Control Center are active.

   See Also: "Creating Additional Configurations" in the *Oracle Warehouse Builder Installation and Administration Guide*.

3. In the Control Center Manager navigation tree, expand the location node containing the object to be deployed. Select the objects to be deployed.

   You can select multiple objects by holding down the Ctrl key while selecting the objects.

4. Set the deployment action for the selected objects in the Object Details panel.
   - 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.

5. Click the Deploy icon.

Warehouse Builder must be open during generation of the job, but not during deployment. The Control Center Manager can be closed at any stage.

Deploying Business Definitions to Oracle Discoverer

After you create your business definitions, you can deploy them to Oracle Discoverer. The method used to deploy business definitions depends on the version of Oracle Discoverer to which business definitions are deployed and the licensing option you use.

Note: This feature requires one or more additional options as described in *Oracle Database Licensing Information*.

Note: To deploy business definitions to an Oracle Discoverer location, the EUL owner must have the `CREATE DATABASE LINK` privilege.

Table 25–1 summarizes the combinations possible when you deploy business definitions to Oracle Discoverer using the different licensing options.

Table 25–1 Different Methods of Deploying Business Definitions

	Warehouse Builder Core Functionality	Warehouse Builder Enterprise ETL Option
Versions Lower than Oracle Discoverer 10*g* Release 2	Generate scripts for the business definitions, copy these scripts to an .eex file, and import the .eex file into Oracle Discoverer. See "Deploying Business Definitions Using the Core Functionality" on page 25-6.	Use the Control Center to create an .eex file and then import the .eex file into Oracle Discoverer. See "Deploying Business Definitions to Earlier Versions of Oracle Discoverer" on page 25-6.
Oracle Discoverer 10*g* Release 2	Generate scripts for the business definitions, copy these scripts to an .eex file, and import the .eex file into Oracle Discoverer. See "Deploying Business Definitions Using the Core Functionality" on page 25-6.	Use the Control Center to directly deploy to Oracle Discoverer. See "Deploying Business Definitions Directly to Oracle Discoverer" on page 25-5.

Deploying Business Definitions Directly to Oracle Discoverer

You can directly deploy business definitions to Oracle Discoverer, just like you deploy other data objects, using the Control Center or Project Explorer. 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 11-4.

When you deploy business definitions directly to Oracle Discoverer 10*g* Release 2, the following steps are performed:

1. Creates an .eex file that contains the definitions of the business definitions.
2. Connects to the EUL specified in the Discoverer location associated with the Business Definition module containing the business definitions.

   Note: If the EUL is in a different database from your object definitions, a connector is created. This connector is deployed when you deploy the business definitions.

3. 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 Earlier Versions of Oracle Discoverer

You cannot directly deploy business definitions to versions of Oracle Discoverer earlier than 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 is associated with a version of Discoverer lower than 10*g* Release 2, the deployment will fail. But an .eex file that contains the business definitions is created. This .eex file is assigned a default name, for example, *2022.eex*, and is stored in the *OWB_ORACLE_HOME*\deployed_scripts directory. You can connect to the EUL using Oracle Discoverer and import this .eex file.

Deploying Business Definitions Using the Core Functionality

When you use the core functionality of Warehouse Builder, you cannot directly deploy business definitions to Oracle Discoverer. You also cannot use the Control Center to create an .eex file as described in "Deploying Business Definitions to Earlier Versions of Oracle Discoverer" on page 25-6. However, you can save your business definitions to Discoverer using the steps described in the following section.

Use the following steps to save business definitions to Discoverer:

1. Associate a valid location with the Business Definition Module that contains the business definitions.

   Although you cannot use this location to deploy business definitions, defining the location ensures that the credentials of the EUL user are included in the generated code. When you define the location, a check is performed to determine if the relational schema that the intelligence objects reference is in the same database as the objects. If they are in different databases:

   - A connector is created to the Discoverer location.

- The name of the database link used by the connector is included in the generated code.

The connector is created under the Discoverer location node associated with the Business Definition module. Ensure that you deploy this connector to create a database link.

2. Right-click the business definition module that contains the business definitions that you want to deploy to Discoverer and select **Generate**.

   The Generation Results window is displayed.

3. Navigate to the Scripts tab of the Generation Results window.

   This tab lists all the business definitions with the names of the corresponding files that store the scripts generated for these definitions.

4. Select all the objects that you want to save to Oracle Discoverer.

   You can select multiple files by pressing down the Ctrl key.

5. Click the **Save As** button.

   The Save As dialog box is displayed.

6. Select the directory in which you want to save the generated scripts. Ensure that you save all the files in a single directory.

   For example, you save all the generated scripts in the directory `c:\sales\generated_scripts`.

7. Copy all the generated scripts to a single .eex file.

   Use the operating system commands to concatenate the generated scripts into a single file. For example, in Windows, you open a Command Prompt window and execute the following steps:

   ```
   c:> CD c:\sales\generated_scripts
   c:\sales\generated_scripts> COPY *.xml sales_scripts.eex
   ```

   This copies all the generated .xml files to an .eex file called `sales_scripts.eex`.

8. Edit the .eex file created in the previous step using any text editor and perform the following steps:

   a. Add the following lines at the beginning of the file:

      ```
      <?xml version="1.0" encoding="UTF-8"?>
      <EndUserLayerExport SourceEULId="20030730144738"
      SourceEULVersion="4.1.9.0.0" MinimumCodeVersion="4.1.0"
      ExportFileVersion="4.1.0">
      ```

   b. Add the following lines at the end of the file:

      ```
      </EndUserLayerExport>
      ```

9. Open Oracle Discoverer Administrator and connect to the EUL into which you want to import the business definitions.

10. From the File menu, select **Import**.

    The Import Wizard is displayed.

11. Import the .eex file you created into Discoverer Administrator.

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, select **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.

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 create and use schedules. For more information about schedules, see "Process for Defining and Using Schedules" on page 26-2.

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.

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.

To start ETL from the Project Explorer:

Select a mapping or a process flow, then select **Start** from the Design menu.

To start ETL from the Control Center Manager:

Select the mapping or process flow, then click the Start icon in the toolbar.

Alternatively, you can select the mapping or process flow, then select **Start** from the File menu.

> **See Also:** *Oracle Warehouse Builder API and Scripting Reference* for information about using SQL*Plus to start ETL jobs.

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.

Scheduling ETL Jobs

You can use any of the following methods to schedule ETL:

- Use the scheduler

  See "Process for Defining and Using Schedules" on page 26-2.

- Use a third-party scheduling tool

Deploying to Additional Locations

A workspace 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 workspace and a user for the target schema.
2. In the local Design Center, create a new configuration and a new Control Center for the workspace on the target system. Make this the default configuration.
3. Define a target location for the remote schema.

4. Deploy as usual.

Runtime Preferences

You can alter optional runtime platform preferences using the file *OWB_ORACLE_HOME*/owb/bin/admin/Runtime.properties. This section provides a description of the properties you can alter.

SQL Loader Runtime Preferences

- **property.RuntimePlatform.0.NativeExecution.SQLLoader.oracle_home_10g**

  Defines the location of the Oracle 10*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, the oracle_home_default property is used.

- **property.RuntimePlatform.0.NativeExecution.SQLLoader.oracle_home_9i**

  Defines the location of the Oracle 9*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, the oracle_home_default property is used.

- **property.RuntimePlatform.0.NativeExecution.SQLLoader.oracle_home_8i**

  Defines the location of the Oracle 8*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, the oracle_home_default property is used.

- **property.RuntimePlatform.0.NativeExecution.SQLLoader.oracle_home_default**

  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, 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**

  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.

  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**

  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, the sqlldr_exe_default property is used.

  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**

  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, the sqlldr_exe_default property is used.

  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**

  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, 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.

SQL*Plus Runtime Preferences

- **property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_10g**

  Defines the location of the Oracle 10*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, the `oracle_home_default` property is used.

- **property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_8i**

  Defines the location of the Oracle 8*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, the `oracle_home_default` property is used.

- **property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_9i**

  Defines the location of the Oracle 9*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, the `oracle_home_default` property is used.

- **property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_default**

  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, the `ORACLE_HOME` environment variable of the Control Center is used, else the Control Center's own home is used.

- **property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_10g**

  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, the `sqlldr_exe_default` property is used.

  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**

  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, the `sqlldr_exe_default` property is used.

  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**

  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, the `sqlldr_exe_default` property is used.

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**

  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, 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.

Other Runtime Preferences

- **property.RuntimePlatform.0.delete_all_job_on_complete**

  Represents the action to be taken for audit records of successful completed jobs. Set this property to true to automatically delete audit-execution record and any map-audit records for all jobs that complete successfully. Set this property to false to retain audit records. The default setting is false.

- **property.RuntimePlatform.0.logfile_max_size**

  Represents the maximum number of trace messages that will be logged into the Control Center Service log file. 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 log file will only be created when the Control Center Service is started.

- **property.RuntimePlatform.0.max_number_logfiles**

  Represents the number of log files that will be automatically retained in the *`OWB_ORACLE_HOME`*`/log` subdirectory. This property must be set to at least 2. the default value is 10. When a new log file is created, the system will ensure that at most this number of log files will be retained in the `log` subdirectory.

- **property.RuntimePlatform.0.purge_delay_hours**

  Represents when the audit details must be purged. Audit details of any deployment or execution jobs with start time older than the specified number of hours may be purged, regardless of job status. The default value is 24 hours. The minimum value you can set is 1 hour.

- **property.RuntimePlatform.0.purge_minimum_minutes**

  Audit details of any Deployment or Execution with start time under the specified number of minutes old may not be purged, regardless of job status. The default value is 10 minutes and the minimum value is 1 minute.

- **property.RuntimePlatform.0.recovery**

  Controls whether deployments and executions are recovered when the Control Center Service starts up. The default value is true. This means that any deployments or executions that are still busy will be restarted when the service starts up. A value of false means that such deployments or executions are not recovered and are deactivated.

- **property.RuntimePlatform.0.platform_net_host_name**

  Defines the host name that should be used when making inward connections to the Control Center. On RAC installations, this would normally identify the entire

cluster, not a node within the cluster. This 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, the host name of the node running the Control Center service is used.

- **property.RuntimePlatform.0.platform_net_service_name**

  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, the Control Center service connection information is used.

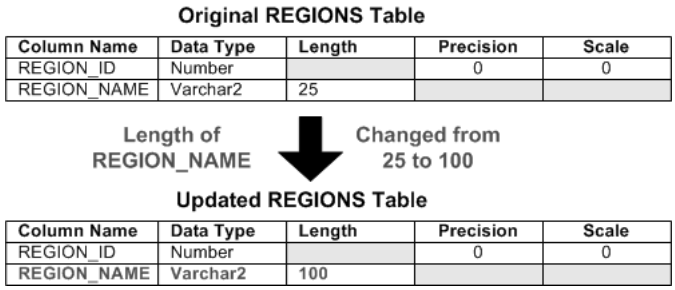
Example: Updating a Target Schema

Scenario

You are in charge of managing a data warehouse that has been in production for a few months. The data warehouse was originally created using two source schemas, Human Resources (HR) and Order Entry (OE) and was loaded into the Warehouse (WH) target schema. Recently you were made aware of two changes to tables in the HR and OE schemas. The WH schema must be updated to reflect these changes.

- Change #1: The first change was made to the HR schema as show in Figure 25–1. The length of the REGION_NAME column in the REGIONS table was extended to 100 characters.

Figure 25–1 Changed REGIONS Table



This image contains the Original Regions table at the top, an arrow pointing downwards, and the Updated Regions table below the arrow.

At the top, is the table containing the original REGIONS table. This table has five column names, ordered from left to right: Column Name, Data Type, Length, Precision, and Scale. This table also has two rows. The first row contains REGION_ID in the Column Name column and Number in the Data Type column. The second row contains REGION_NAME in the Column Name column, Varchar2 in the Data Type column, and 25 in the Length column.

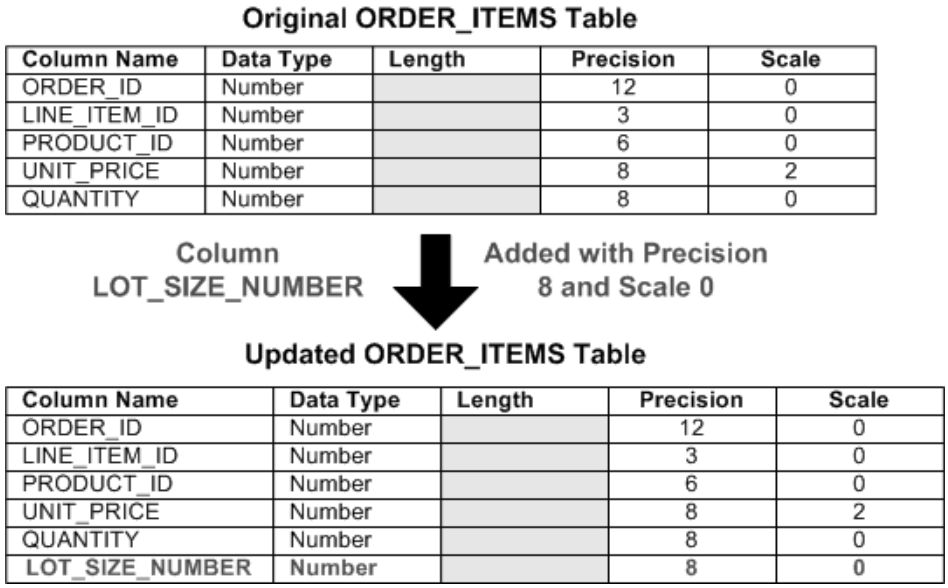
At the bottom of the image is the table containing the updated REGIONS table. This table has five column names, ordered from left to right: Column Name, Data Type Number, Length, Precision, and Scale. This table contains tow rows. The first row contains REGION_ID in the Column Name column and Number in the Data Type

column. The second row contains REGION_NAME in the Column Name column, Varchar2 in the Data Type column, and 100 in the Length column.

****

- Change #2: The second change was made to the OE schema as shown in Figure 25–2. A row called LOT_SIZE_NUMBER was added to the ORDER_ITEMS table with a precision of 8 and scale of 0.

Figure 25–2 Changed ORDER_ITEMS Table



This image displays the Original ORDER_ITEMS table at the top, and arrow from this table pointing downwards. Below the arrow is the Updated ORDER_ITEMS table.

At the top, is the table containing the Original ORDER_ITEMS table. This table has five column names, ordered from left to right: Column Name, Data Type, Length, Precision, and Scale. This table has five rows. In the center is an arrow pointing downwards from the Original ORDER_ITEMS table to the Updated ORDER_ITEMS table.

At the bottom is the table containing the Updated ORDER_ITEMS table. This table has five column names, ordered from left to right: Column Name, Data Type, Length, Precision, and Scale. An additional row for the LOT_SIZE_NUMBER column has been added with Precision 8 and Scale 0.

****

Solution

In order to update the WH schema, you must first determine the impact of these changes and then create and execute a plan for updating the target schema. The following steps provide an outline for what you need to do:

Step 1: Identify Changed Source Objects

Step 2: Determine the Impact of the Changes

Step 3: Reimport Changed Objects

Step 4: Update Objects in the Data Flow

Step 5: Redesign your Target Schema

Step 6: Re-Deploy Scripts

Step 7: Test the New ETL Logic

Step 8: Update Your Discoverer EUL

Step 9: Execute the ETL Logic

Case Study

Step 1: Identify Changed Source Objects

The first step in rolling out changes to your data warehouse is to identify the changes in source objects. In order to do this, you must have a procedure or system in place that can notify you when changes are made to source objects.

In our scenario, you were made aware by the group managing the HR and OE schemas that some objects had been changed. There were two changes, the first was made to the HR schema. The REGION_NAME column was extended from 25 to 100 characters to accommodate longer data. The second change was made to the OE schema. The LOT_SIZE_NUMBER column was added and needs to be integrated into the WH schema.

Step 2: Determine the Impact of the Changes

After you have identified the changes, you must determine their impact on your target schema.

For Change #1, made to the HR schema, you need to update any dependent objects. This entails reimporting the REGIONS table and then updating any objects that use the REGION_NAME column. To identify dependent objects, you can use the Impact Analysis Diagram. You also need to update any mappings that use this table.

For Change #2, made to the OE schema, in addition to reimporting the table and updating mappings, you need to find a way to integrate the new column into the WH schema. Since the column was added to keep track of the number of parts or items in one unit of sales, add a measure called NUMBER_OF_IND_UNITS to the SALES cube in the WH schema and have this measure for each order. Then you need to connect this new column to the SALES cube.

Step 3: Reimport Changed Objects

Since two source objects have changed, you must start by reimporting their metadata definitions into your workspace. Select both the REGIONS table in the HR schema and the ORDER_ITEMS table in the OE schema from the navigation tree and use the Metadata Import Wizard to reimport their definitions.

Warehouse Builder automatically detects that this is an update and proceeds by only updating changed definitions. The Import Results dialog box that displays at the end of the import process displays the details of the synchronization. Click **OK** to continue the import and commit your changes to the workspace. If you do not want to continue with the import, click Undo.

Step 4: Update Objects in the Data Flow

If the change in the source object altered only existing objects and attributes, such as Change #1 in the HR schema, use Impact Analysis diagrams to identify objects that need to be reconciled.

In our scenario, we need to reconcile the column length in all objects that depend on the REGIONS table to ensure that the data continues to load properly.

To update objects in the data flow:

1. Select the REGIONS table in the HR schema from the navigation tree. Select **View** and then click **Impact**.

   The Metadata Dependency Manager opens and the Impact Analysis diagram reveals that the CUSTOMER dimension in the WH schema is the only object impacted by the REGIONS table.

   This step requires that you have already set up the Repository Browser. For more information on setting this up, see *Oracle Warehouse Builder Installation and Administration Guide*.

2. Open the CUSTOMER dimension in the Data Object Editor and update the Region Name level attribute to 100 character length.

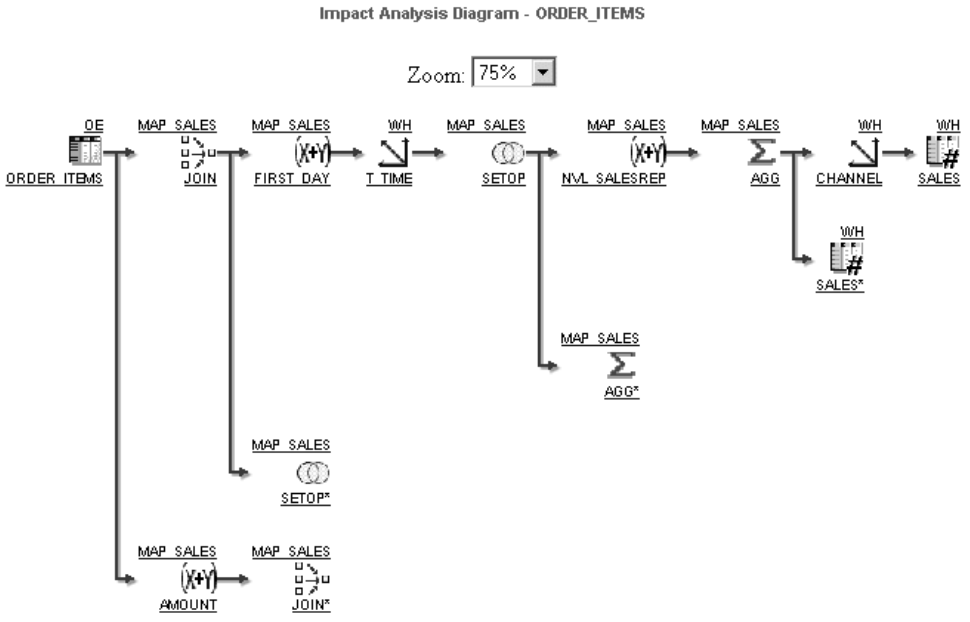
3. Open the MAP_CUSTOMER mapping that connects the source to the target. For both the REGIONS table operator and the CUSTOMER dimension operator, perform an inbound synchronization from data object to mapping operator.

   The mapping operators must be synchronized with the mapping objects they represent in order to generate code based on the updated objects.

You have now completed updating the metadata associated with Change #1.

For Change #2, since it introduced a new column, you do not need to update the data flow the same way you did for Change #1. Make sure you perform an inbound synchronization on all the mappings that use an ORDER_ITEMS table operator. From the Impact Analysis Diagram for the ORDER_ITEMS table shown in Figure 25–3, we can see that this is only the mapping MAP_SALES.

Figure 25–3 Impact Analysis Diagram for ORDER_ITEMS



This screenshot displays the Impact Analysis Diagram. At the top center is a Zoom list (currently set at 75%). Below the list, ordered from left to right, are the following: ORDER ITEMS, JOIN, FIRST_DAY, T_TIME, SETOP, NVL_SALESREP, AGG, CHANNEL, and SALES. Arrows connect these objects in the same order in which they are displayed.

At the bottom, ordered from left to right are the following: AMOUNT, JOIN, SETOP, AGG, SALES. There is an arrow from ORDER_ITEMS to AMOUNT and then from AMOUNT to JOIN (in the bottom row). There is an arrow from JOIN in the top row to SETOP in the bottom row. An arrow connects SETOP in the top row to AGG in the bottom row. An arrow also connects AGG in the top row to SALES in the bottom row.

****

Step 5: Redesign your Target Schema

Since Change #2 introduced the new LOT_SIZE_NUMBER column to the ORDER_ITEMS table, you need to redesign your WH target schema to incorporate this new data into your cube. You can do this by adding a new measure called NUMBER_OF_IND_UNITS to your SALES cube.

To redesign the target schema:

1. Add the measure NUMBER_OF_IND_UNITS with the NUMBER data type, precision of 8, and scale of 0 to the SALES cube.
2. View the lineage diagram for the SALES cube to determine which mappings contain the SALES cube. Perform an inbound synchronization on all SALES cube mapping operators.
3. Open the mapping MAP_SALES and ensure that the table ORDER_ITEMS is synchronized inbound.
4. Connect the LOT_SIZE_NUMBER column in the ORDER_ITEMS table to the JOIN, and then to the SETOP, and then add it to the AGG operators. Ensure that you are doing a sum operation in the AGG operator.
5. Finally, connect the LOT_SIZE_NUMBER output attribute of the AGG operator to the NUMBER_OF_IND_UNITS input attribute of the SALES cube.

Step 6: Re-Deploy Scripts

After the mappings have been debugged, use the Design Center to regenerate and re-deploy scripts. Use the Control Center Manager to discover the default deployment action. Warehouse Builder detects the type of deployment to run.

Step 7: Test the New ETL Logic

After you have reconciled all objects and ensured that the WH target schema has been updated to reflect all changes, test the ETL logic that is be generated from the mappings. Use the Mapping Debugger to complete this task. If you find any errors, resolve them and re-deploy the scripts.

Step 8: Update Your Discoverer EUL

If you are using Discoverer as your reporting tool, proceed by updating your EUL.

To update your Discoverer EUL:

1. Identify the objects that need to be updated in the EUL because of changes made to their structure or data. In this case, the changed objects are the REGIONS and SALES_ITEMS tables and the SALES cube.
2. In the Project Explorer, select all the objects identified in step 1, right-click and select **Derive**.

   The Perform Derivation Wizard displays and updates these object definitions in the Business Definition Module that contains these objects.

3. Expand the Item Folders node in the Business Definition Module that contains these changed objects.
4. Select the objects identified in Step 1, right-click and select **Deploy**.

   The changes to the objects are updated in the Discover EUL.

Step 9: Execute the ETL Logic

After the mappings have been deployed, execute and load data to the target.

26

Scheduling ETL Objects

This chapter contains the following topics:

- About Schedules
- Process for Defining and Using Schedules
- Editing a Schedule

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.

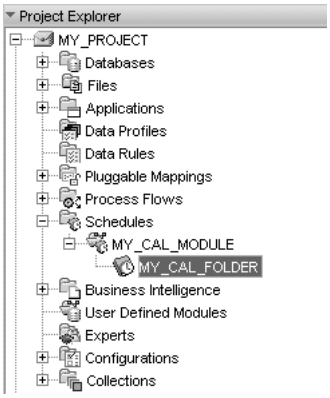
When you are in the development phase of using Warehouse Builder, you may not want to schedule mappings and process flows but rather start and stop them immediately from a Control Center as described in "Deploying Objects" on page 25-4.

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, which can be deployed to a scheduler such as Oracle 10*g* Scheduler or Applications Concurrent Manager.

Schedules are defined in the context of projects and contained in schedule modules under the **Schedules** node on the Project Explorer.

Figure 26–1 displays schedules on the Project Explorer.

Figure 26–1 Schedules on the Project Explorer



This screenshot shows the Project Explorer. The node MY_PROJECT is expanded in the navigation tree. The Schedules node under MY_PROJECT is expanded to display

the MY_CAL_MODULE node, which in turn is expanded to display MY_CAL_FOLDER. The schedule MY_CAL_FOLDER is currently selected.

****

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*. Scheduled jobs should be deployed to a standard database location, not to a Workflow Location. Only Process Flow packages should be deployed to Oracle Workflow.

Scheduled jobs may reference an executable object such as a process flow or a mapping. If a job references a process flow, then you must deploy the process flow to Oracle Workflow and deploy the scheduled job to either a database location or a Concurrent Manager location.

For remote Oracle Workflow locations and remote Warehouse Builder 10*g* locations to which schedules are deployed, 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 displays 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.

   > **See Also:** "Example Schedules" on page 26-8 for examples of schedules
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, click 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.

Editing a Schedule

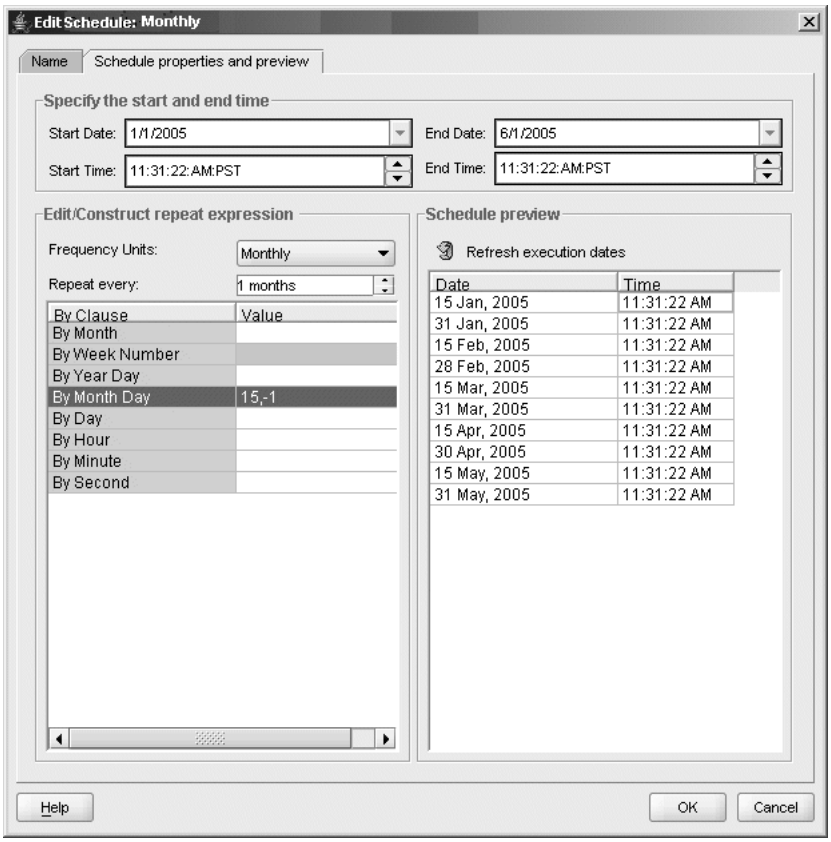
Use the schedule editor to edit a schedule.

The repeat expression appears in the lower left panel of the editor. 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, see Example Schedules on page 26-8.

Figure 26–2 The Schedule Editor



This image displays 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 in the schedule editor, the start date and time become the defaults for the By Clauses in the Repeat Expression. The execution time in Schedule Preview corresponds to the Start Time.

Defining Schedules To Repeat

The repeat expression determines how often the schedule is executed. 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 By Clauses are not available. After you complete the wizard, you can open the schedule and set the By Clauses 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.

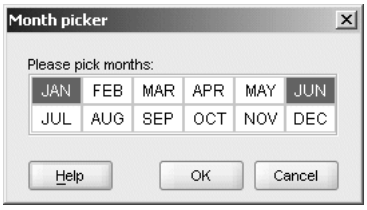
Repeat Every

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, the schedule is evaluated 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, the schedule is evaluated 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 box. If your goal is to know how to quickly type in values, first use the selector dialog box to learn what values are valid and also refer to Example Schedules on page 26-8.

Figure 26–3 Selector dialog box for Picking Months in a Year

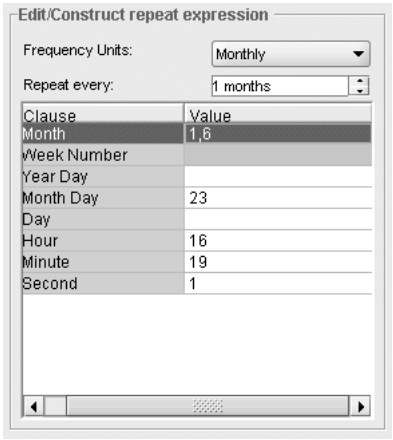


This image displays the selector dialog box for picking months in a year.

****

When you use the selector dialog box and select **OK,** the results are displayed in the schedule editor. In this way, you can use the selector dialog box to learn what values are valid.

Figure 26–4 Month Clause for January and June



This image displays the 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

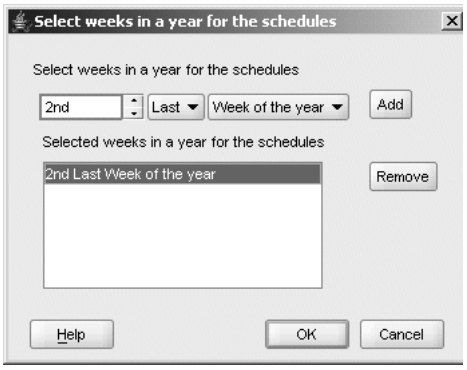
This specifies in which month or months the schedule is valid. If you type in values, use numbers such as 1 for January and 3 for March, or use three-letter abbreviations such as FEB for February and JUL for July.

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 box. 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 box.

Figure 26–5 By Week Number Clause Set to Second To Last Week of the Year



This image displays the By Week Number Clause set to second 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 clause 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 version is 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 per frequency. The supported frequencies are MONTHLY and YEARLY.

Example Schedules

Use Table 26–1 as a guide for defining schedules.

Table 26–1 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

Editing a Schedule

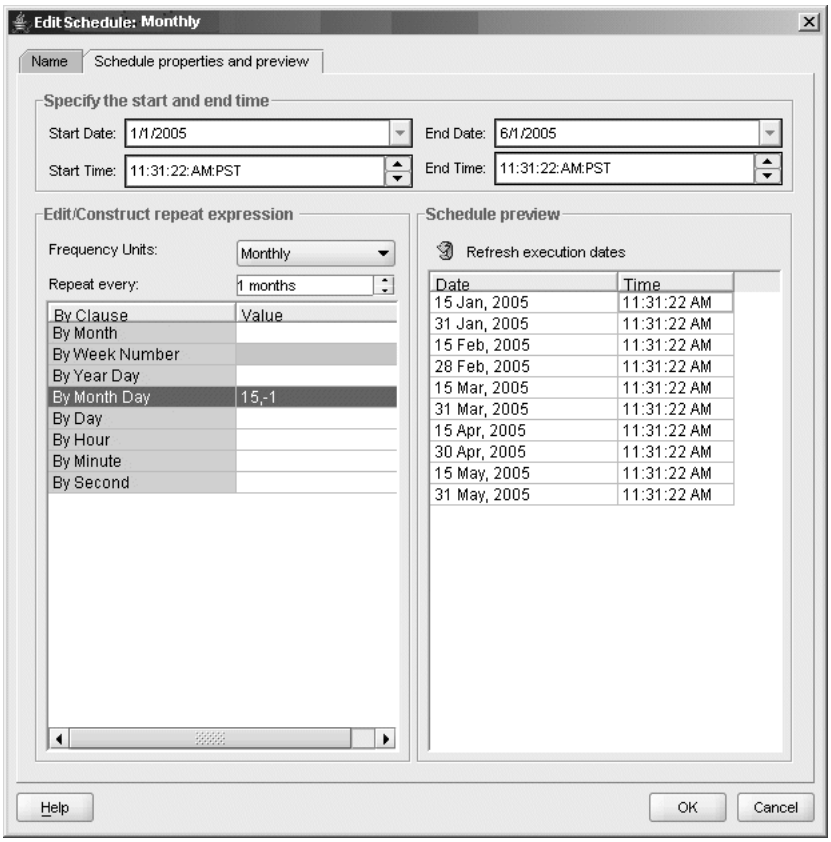
Use the schedule editor to edit a schedule.

The repeat expression appears in the lower left panel of the editor. 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, see Example Schedules on page 26-8.

Figure 26–6 The Schedule Editor



This image displays 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 in the schedule editor, the start date and time become the defaults for the By Clauses in the Repeat Expression. The execution time in Schedule Preview corresponds to the Start Time.

Defining Schedules To Repeat

The repeat expression determines how often the schedule is executed. 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 By Clauses are not available. After you complete the wizard, you can open the schedule and set the By Clauses 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.

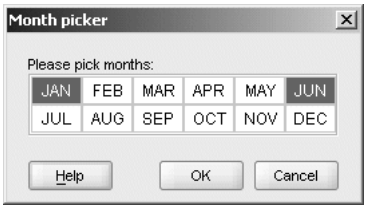
Repeat Every

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, the schedule is evaluated 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, the schedule is evaluated 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 box. If your goal is to know how to quickly type in values, first use the selector dialog box to learn what values are valid and also refer to Example Schedules on page 26-8.

Figure 26–7 Selector dialog box for Picking Months in a Year

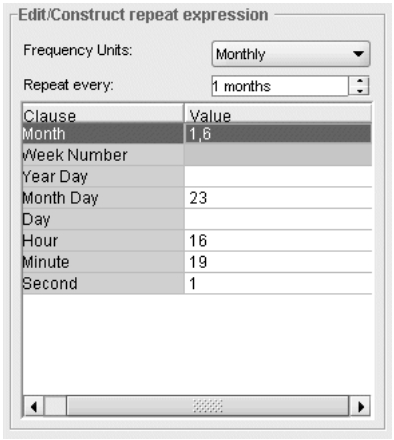


This image displays the selector dialog box for picking months in a year.

****

When you use the selector dialog box and select **OK,** the results are displayed in the schedule editor. In this way, you can use the selector dialog box to learn what values are valid.

Figure 26–8 Month Clause for January and June



This image displays the 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

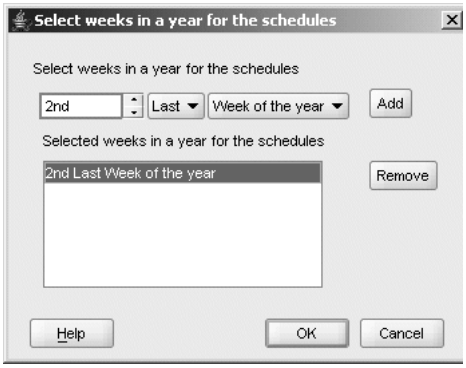
This specifies in which month or months the schedule is valid. If you type in values, use numbers such as 1 for January and 3 for March, or use three-letter abbreviations such as FEB for February and JUL for July.

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 box. 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 box.

Figure 26–9 By Week Number Clause Set to Second To Last Week of the Year



This image displays the By Week Number Clause set to second 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 clause 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 version is 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 per frequency. The supported frequencies are MONTHLY and YEARLY.

27

Auditing Deployments and Executions

Auditing deployment and execution information can provide valuable insights into how your target is being loaded and how you can further optimize mapping and process flow settings and parameters. It also provides immediate feedback on deployment information that enables you to retrieve the deployment history of an object. The Repository Browser in Oracle Warehouse Builder provides the auditing and deployment information.

This section covers the following topics:

- About the Repository Browser
- Opening the Repository Browser
- The Design Center
- Control Center Reports
- Common Repository Browser Tasks

About the Repository Browser

The Repository Browser is a browser-based tool that generates reports from data stored in Oracle Warehouse Builder repositories. Using the Repository Browser, you can view:

- Detailed information about the design of a workspace. Reports are generated from data stored in the workspaces.
- 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 public views. Start a SQL*Plus session and query the public views. Refer to *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 of these activities are stored in the workspace. You can access this information from the following 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. Note that the Repository Browser Listener must be running.

To use the Repository Browser:

1. From the Design Center Tools menu, select **Repository Browser**.
2. On the connection page, log in with your user name and password.

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 in Windows" on page 27-2.
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 27-3.
3. The Repository Browser opens to the Login page where you log in to a workspace as described in "Logging in to a Workspace" on page 27-3.

> **Note:** 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 workspace you want to browse. When you are not the owner of the workspace, 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 in Windows

Open a command prompt window and run `startOwbbInst.bat`. This file is located in the `OWB_ORACLE_HOME\owb\bin\win32` directory. Set the password for the `oc4j` administrator.

Stopping the Repository Browser Listener

Open a command prompt window and run `stopOwbInst.bat`. Provide the `oc4j` administrator password that you set while running `startOwbInst.bat`.

The `stopOwbbInst.bat` file is located in the `OWB_ORACLE_HOME\owb\bin\win32` directory.

Starting the Repository Browser Listener in Unix

Run `./startOwbbInst.sh`. This file is located in the `OWB_ORACLE_HOME/owb/bin/unix` directory. Set the password for the `oc4j` administrator.

Stopping the Repository Browser Listener in Unix

Run `./stopOwbInst.sh`. Provide the `oc4j` administrator password that you set while running `startOwbInst.sh`.

The `stopOwbbInst.sh` file is located in the `OWB_ORACLE_HOME/owb/bin/unix` directory.

Starting the Repository Browser

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 Warehouse Builder folder**,** and then **Warehouse Builder, Repository Browser.**
- From the menu of the Design Center of the Warehouse Builder, select **Tools**, **Repository Browser**.
- From within any web browser, type the location of the Repository Connection page. For example, if the Repository Browser Listener is running on a machine named `owb_server`, then typing the following address will start the Repository Browser:

  ```
  https://owb_server:8999/owbb/RABLogin.uix?mode=design
  ```

  or

  ```
  https://owb_server:8999/owbb/RABLogin.uix?mode=runtime
  ```

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 Workspace

To log in to a workspace, specify the connection information for the workspace you would like to access. If you do not know the connection information, contact your database administrator.

Once you log in to the repository browser, you can view any of the following reports:

- Deployment
- Execution
- Management

If you want to browse through the workspace using the **Repository Navigator** page, select **Design Center: Navigator**.

The Design Center

When you log in, the Repository Browser opens the Reports page. From this page you can navigate to the Navigator page to obtain information about the design of Oracle Warehouse Builder Repositories such as:

- Object Reports

- Object Properties
- Object Lineage
- Object Impact

Repository Navigator

Use the Repository Navigator page to search the metadata of a workspace and to access metadata main properties, lineage, impact analysis, and a list of related reports and Control Center reports for the workspace.

Search

Search by the object type, or name, or both.

- To search by object type, select the type of object you want to search from 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 of the name, in which case the search result is a list of objects whose names begin with that character.

Click **Go** to start your search.

The Repository Browser displays the results of the search in a new page called the **Repository Objects Search** page. You can also search for the new objects from this page.

All

Contains a navigator tree for the workspace.

The use of the columns is described in 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 (+) or minus (-) sign 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.
Reports	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 page for the related item.

Refresh

Click to refresh your view of the workspace.

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
- Impact Analysis Reports

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 about how physical objects are used to implement logical objects.

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.

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 pages by clicking on the corresponding link on the left side of the page.

See also: Summary Reports on page 27-5 and Detailed Reports on page 27-5

Object Lineage

The Object Lineage page displays the lineage diagram for the object that you selected in Repository Navigator. 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.

From this page you can go to the Object Properties, Object Reports, or Object Impact pages by clicking on the corresponding link on the left side of the page.

Object Impact

The Object Impact 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. The subject is displayed on the left side of the screen.

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.

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.

Control Center Reports

The Control Center section 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 attributes and display a node-tree giving details of all deployments in time order.
- Object Summary Reports that show basic 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 the 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 flow, mapping, or data object.

Execution Reports

Top-level execution reports

- Execution Schedule Reports that show basic 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 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 allows 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 Report, Object Summary Report, andLocations Report. 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 attributes and displays a node-tree giving details of all deployments in time order.

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 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. These include the following: ■ Complete: Indicates that the deployment is complete. ■ Busy: Indicates that the deployment is in progress. ■ Ready: Indicates that the job has just started or is about to end. **Note:** If the deployment status shows a Ready status for a very long time, use the script `deactivate_deployment.sql` located in the `OWB_ORACLE_HOME/owb/rtp/sql` folder to deactivate 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 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 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 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 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 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, and 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, and 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 when the Map Run is executed with a particular setting of the Audit Level runtime parameter. Use this trace facility only if required because 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, and 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 displays and allows you to manage service node information for the RAC system. Specifically, it performs the following functions:

- shows the basic 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
- 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 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 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
- Viewing Error Tables Created as a Result of Data Auditor Execution
- Unregistering a Location
- Updating Location Connection Details for a Changed Database Environment
- Updating Service Node Details in a Changing RAC Environment

Identifying Recently-Run Processes

1. Open the Execution Schedule Report to see the latest Processes runs.
2. Filter the displayed information 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 displayed information 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 displayed information 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 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, which 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 displayed information by using execution name.
3. Select all the executions which are to be removed, and click **Purge Selected Audit Details**.

   A confirmation screen is shown, requesting you to confirm the action.

Removing Old Deployment Audit details

1. Open the Deployment Schedule Report to see the latest deployments.
2. Filter the displayed information by using date range.
3. Select all the deployments which are to be removed, and click the **Purge Selected Audit Details**.

   A confirmation screen is shown, requesting you to confirm the action.

Viewing Error Tables Created as a Result of Data Auditor Execution

1. Ensure that you grant privileges to the Repository Browser on the error table.

   See "Granting Privileges on Error Tables" on page 23-51.
2. Open the Execution Summary Report or Execution Schedule Report to see a list of all the Processes.
3. Click the link in the Related Information column corresponding to the process flow that contained the data auditor to display the Execution Job Report for the process flow.
4. Click the link in the Related Information column corresponding to the data auditor activity to display the Execution Job Report for the activity.
5. Data that violated the data auditor constraints is listed in the Logical Errors table. Click on the link in the Error Table column to display the Execution Report for the error table.

If you have sufficient privileges, you can use the Execution Report for an error table to purge data from that error table.

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 is shown, requesting you to confirm the action.

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.

28

Troubleshooting and Error Handling for ETL Designs

This chapter contains information about error logs and using DML error logging in Warehouse Builder. The chapter includes the following sections:

- Inspecting Error Logs in Warehouse Builder
- Using DML Error Logging
- Using Pseudocolumns ROWID and ROWNUM in Mappings

Inspecting Error Logs in Warehouse Builder

Scenario

While working with Warehouse Builder, the designers need to access log files and check on different types of errors. This case study outlines all the different types of error messages that are logged by Warehouse Builder and how to access them.

Solution

Warehouse Builder logs the following types of errors when you perform different operations:

- Validation Errors on page 28-1
- Generation Errors on page 28-2
- Deployment and Execution Errors on page 28-3
- Name and Address Server Errors on page 28-4

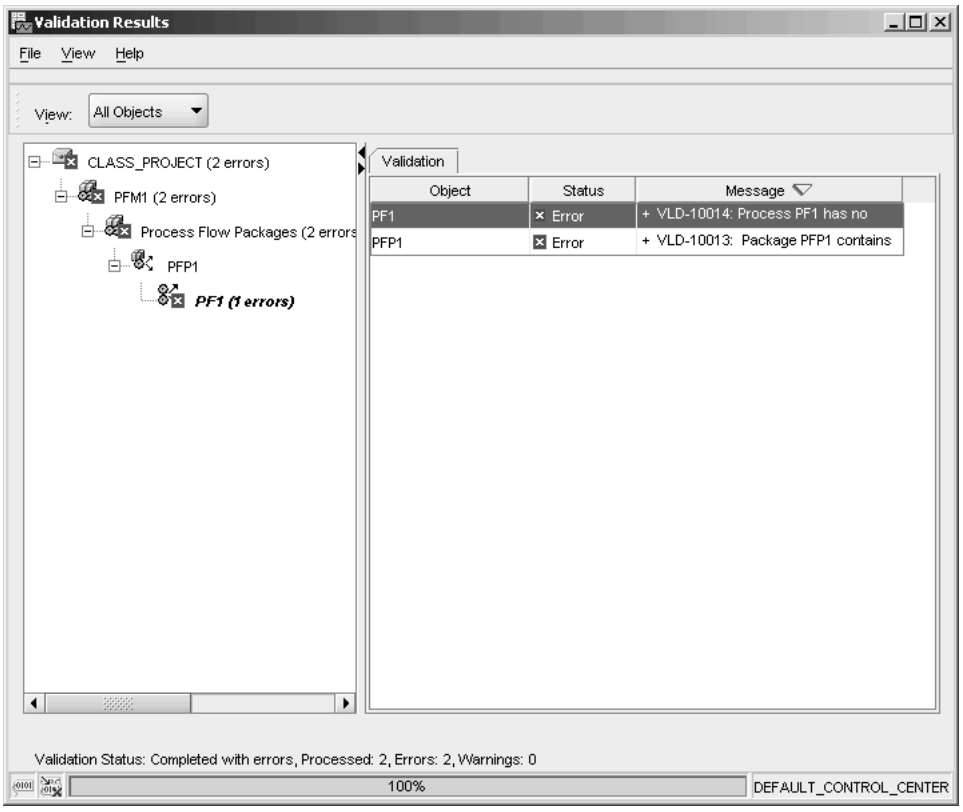
Case Study

This case study shows you how to retrieve error logs after performing different operations in Warehouse Builder.

Validation Errors

In Warehouse Builder, you can validate all objects by selecting the objects from the console tree and then selecting **Validate** from the Object menu. After the validation is complete, the validation messages are displayed in the Validation Results window, as shown in Figure 28–1.

Figure 28–1 Validation Error Messages



This screenshot displays the Validation Results window. At the top of the window is a menu bar containing the following items, ordered from left to right: File, View, and Help. Below the menu bar is a View list. Below the View list, the left pane displays a navigation tree containing selected objects. The selected objects list contains all the objects in the project in a collapsible tree. The right pane contains the Validation tab. On this tab are the following fields, ordered from left to right: Object, Status, and Message.

****

You can also validate mappings from the Mapping Editor by selecting **Mapping,** then **Validate.** The validation messages and errors are displayed in the Validation Results window.

On the Validation tab of the Validation Results window, double-click an object name in the Object column to display the editor for that object. You can fix errors in the editor. Double-click a message in the Message column to display the detailed error message in a message editor window. To save the message to your local system, select **Code** in the menu bar, then select **Save as File**.

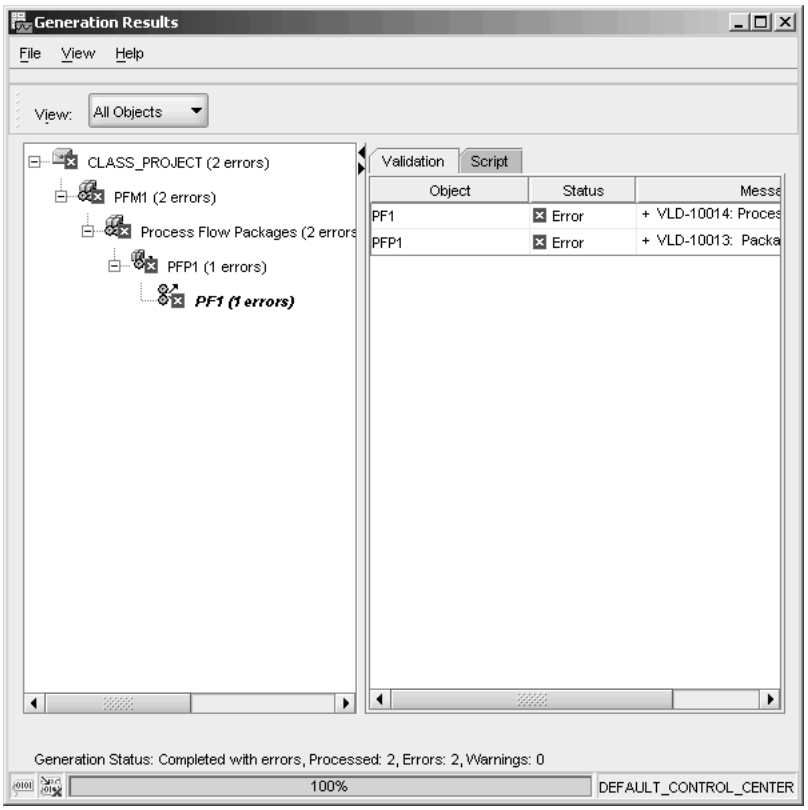
Warehouse Builder saves the last validation messages for each previously validated objects. You can access these messages at any time by selecting the object from the console tree in the Project Explorer, select **View** from the menu bar, and then click **Validation Messages.** The messages are displayed in the Validation Results window.

Generation Errors

After you generate scripts for Warehouse Builder objects, the Generation Results window displays the generation results and errors, as shown in Figure 28–2.

Double-click an error under the Messages column on the Validation tab to display a message editor that enables you to save the errors to your local system.

Figure 28–2 Generation Results Window



This screenshot displays the Generation Results window. At the top of the window is the menu bar containing the following items, ordered from left to right: File, View, and Help. Below the menu bar is the View list which contains the following options, ordered from top to bottom: All Objects, Warnings, and Errors.

Below the View list, on the left, are the objects that were used for the generation results presented in a tree structure. To the right of this are the two tabs, ordered from left to right, Validation and Script. The Validation tab is currently displayed. The information about the validation is displayed in a table with the following columns, ordered from left to right: Object, Status, and Message.

****

Deployment and Execution Errors

You can store execution or deployment error and warning message logs on your local system by specifying a location for them. In the Project Explorer, select the project. Then from the Tools menu, select **Preferences**. In the Preferences dialog box, click the Logging option in the object tree to the left. In the list box on the right, you can set the log file path, file name and maximum file size. You can also select the types of logs you want to store.

You can view this log of deployment and error messages from the Warehouse Builder console by selecting **View** from the menu bar, and then **Messages Log.** This Message Log dialog box is read-only.

Runtime Audit Browser

If an error occurs while transforming or loading data, the audit routines report the errors into the runtime tables. You can easily access these error reports using the Runtime Audit Browser (RAB). The RAB provides detailed information about past deployments and executions. These reports are generated from data stored in the runtime repositories. Click the Execution tab in the Execution reports to view error messages and audit details.

Name and Address Server Errors

If you are using the Name and Address cleansing service provided by Warehouse Builder, you can encounter related errors.

Name and address server start up and execution errors can be located at:

OWB_ORACLE_HOME\owb\bin\admin\NASver.log

If your Name and Address server is enabled in:

OWB_ORACLE_HOME\owb\bin\admin\NameAddr.properties:TraceLevel=1,

then it produces the log file NASvrTrace.log.

Using DML Error Logging

Error logging enables the processing of DML statements to continue despite errors being encountered during the statement execution. The details of the error such as the error code and the associated error message are stored in an error table. After the DML operation completes, you can check the error table to correct rows with errors. DML error logging is supported for SQL statements such as INSERT, UPDATE, MERGE, and multi-table insert. It is useful in long-running, bulk DML statements.

Warehouse Builder provides error logging for the tables, views, and materialized views used in set-based PL/SQL mappings. To enable error logging, you set the Shadow table name property of the table, view, or materialized view. DML error logging is supported only for target schemas created on Oracle Database 10*g* Release 2 or later versions.

About Error Tables

Error tables store error details. You can define error tables for tables, views, and materialized views only.

Error tables are used for the following purposes:

- DML error logging (including physical errors)
- Capturing logical errors when data rules are applied to tables, views, or materialized views

An error table is generated and deployed along with the base table, view, or materialized view. When you drop a data object, the shadow table associated with it is automatically dropped.

Error Tables and DML Error Logging

When DML error logging is enabled for a data object by setting the Shadow table name property for the object, the error table contains the following:

- DML error columns, as described in Table 28–1.

- All columns from the data object with which the shadow table is associated.

Table 28–1 DML Error Columns in Error Tables

Column Name	Description
ORA_ERR_NUMBER$	Oracle error number
ORA_ERR_MESG$	Oracle error message text
ORA_ERR_ROWID$	Rowid of the row in error (for update and delete)
ORA_ERR_OPTYPE$	Type of operation: insert (I), update (U), delete (D)
ORA_ERR_TAG$	Step or detail audit ID from the runtime audit data. This is the STEP_ID column in the runtime view ALL_RT_AUDIT_STEP_RUNS.

For scalar data types in the source data object, if no data rules are applied to the data object, the columns in the error table are of data type VARCHAR2(4000). This allows physical data errors such as ORA-12899: value too large for column, to be captured. If data rules are applied, the columns in the error table are of the same data type as the source columns.

For example, the table TEST has the two columns C1, of data type NUMBER, and C2, of data type VARCHAR2(10). The error table generated for TEST will contain the DML error columns, C1, and C2. If no data rules are applied to TEST, the data type for both C1 and C2 will be VARCHAR2(4000). If data rules are applied to TEST, C1 will be NUMBER and C2 will be of data type VARCHAR2(10).

Error Tables and Data Rules

When one or more data rules are defined for a data object, the error table for this data object contains the following:

- Columns from the data object

  These columns are of the same data type and precision as the ones in the data object.

- DML error columns, as described in Table 28–1
- Data rule columns

  The data rule columns store details such as the operator that caused the data rule violation, the cause of the error, severity, and the audit run details.

Using Error Tables for DML Error Logging and Data Rules

When you define data rules on a data object for which DML error logging is enabled, the error table generated by Warehouse Builder contains the columns from the data object, the data rules columns, and the DML error columns. The data type and precision of the columns from the data object are the same as the ones in the base data object. This could result in the failed inserts into the error table when errors occur during DML operations. For example, some errors, such as value too small, may cause error table insert failure.

Thus, if you want to perform DML error logging for a data object that has data rules applied, it is recommended that you create your own error tables. Ensure that the error table that you create contains the columns required for data rules and the DML error logging columns.

Enabling DML Error Logging

DML error logging is generated for set-based PL/SQL mappings if the following conditions are satisfied:

- The Error table name property is set for the Table, View, or Materialized View operator.
- The PL/SQL Generated Mode configuration property of the module that contains the mapping is set to Oracle 10gR2, Oracle 11gR1, or Default.

  If the value is set to Default, ensure that location associated with this module has the Version property set to 10.2 or 11.1.

To enable error logging for a data object:

1. In the Project Explorer, right-click the data object for which DML error logging should be enabled, and select **Open Editor**.

   The Data Object Editor for the data object is displayed.
2. On the canvas, select the data object.
3. In the Properties panel, specify a value for the Shadow table name property.

If you do not specify a shadow table name for a data object, DML error logging is not enabled for that object. However, when a data object has data rules associated with it, if you do not specify a error table name for the object, Warehouse Builder creates an error table using a default name. For example, if the name of the table for which you specified data rules is `EMP`, the error table is called `EMP_ERR`.

When you use a data object in a mapping, the Error Table Name property for this data object is derived from the Shadow table name property of the data object.

> **Note:** If you modify the error table name for a data object (using the Shadow table name property), you must synchronize all the operators bound to this data object.

DML Error Logging and ETL

The execution of mappings that contain data objects for which DML error logging is enabled fails if any of the following conditions occur:

- The number of errors generated exceeds the specified maximum number of errors for the mapping

  The default set for this value is 50. You can modify this value by setting the Maximum number of errors configuration property of the mapping. In the Project Explorer, right-click the mapping and select **Configure**. In the Maximum number of errors property, specify the maximum number of errors that can generated before the mapping execution is terminated.
- Errors occur due to functionality that is not supported.

  See "DML Error Logging Limitations" on page 28-7.

You can truncate the error table and delete error details generated during a previous load. This helps in housekeeping of the error tables. To truncate an error table before the map is executed, select the Truncate Error Table property of the operator bound to the data object that has DML error logging enabled.

The properties Roll up Errors and Select only errors from this property are not used for DML error logging.

The Error table name and Truncate error table properties of Table, View, or Materialized View operators are not used for row-based code.

DML Error Logging Limitations

DML error logging has certain limitations. DML error logging is not supported for non-scalar datatypes. In addition, each DML statement has specific limitations, which are listed in documentation related to that statement.

> **See Also:** *Oracle Database SQL Language Reference* for limitations on DML error logging for each DML statement

Depending on your error logging needs you can configure the table operator in a mapping to use the APPEND or NOAPPEND hint. For example, direct-path insert does not support error logging for unique key violations. To log unique key violations, use the NOAPPEND hint.

If you have an error table defined for a data object, you cannot upgrade the data object using the Upgrade option in the Control Center Manager. If you modify the Shadow table name property after the data object is deployed, you must first drop the data object and then redeploy it. If this data object was used in a mapping, ensure that you synchronize the mapping operator with the data object, drop the data object, redeploy the data object and the mapping.

Using Pseudocolumns ROWID and ROWNUM in Mappings

You can use the pseudocolumns `ROWID` and `ROWNUM` in mappings. The `ROWNUM` pseudocolumn returns a number indicating the order in which a row was selected from a table. The `ROWID` pseudocolumn returns the rowid (binary address) of a row in a database table.

You can use the `ROWID` and `ROWNUM` pseudocolumns in Table, View, and Materialized View operators in a mapping. These operators contain an additional column called COLUMN USAGE that is used to identify attributes used as `ROWID` or `ROWNUM`. For normal attributes, this column defaults to TABLE USAGE. To use an attribute for `ROWID` or `ROWNUM` values, set the COLUMN USAGE to ROWID or ROWNUM respectively.

You can map a ROWID column to any attribute of data type `ROWID`, `UROWID`, or `VARCHAR2`. You can map ROWNUM column to an attribute of data type `NUMBER` or to any other data type that allows implicit conversion from `NUMBER`.

Note that `ROWID` and `ROWNUM` pseudocolumns are not displayed in the Data Object Editor since they are not real columns.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X