Oracle® Warehouse Builder

User's Guide

10*g* Release 1 (10.1)

Part No. B12146-02

September 2007

ORACLE®

Oracle Warehouse Builder User’s Guide 10*g* Release 1 (10.1)

Part No. B12146-02

Contributing Author: Shirinne Alison, Kavita Nayar, Michelle Bird, Julia Stein

Contents

C Using the XML Toolkit

D Warehouse Builder Public Views

Preface

This preface includes the following topics:

Purpose

Oracle Warehouse Builder is a comprehensive toolset for practitioners who move and transform data, develop and implement business intelligence systems, perform metadata management, or create and manage Oracle databases and metadata. This guide describes how to use Warehouse Builder to:

- Create a definition of a data warehouse.
- Configure the definitions for a physical instance of the data warehouse.
- Validate the set of definitions and their configurations.
- Generate a set of scripts to create and populate the data warehouse instance.
- Generate data transformation scripts.
- Deploy and initially load the data warehouse instance.
- Maintain the physical instance by conditionally refreshing it with generated scripts.
- Integrate Warehouse Builder metadata with other Business Intelligence products.
- Populate Oracle Discoverer EULs and OLAP catalogs for analyzing the data warehouse.

Audience

This guide is intended for data warehouse practitioners, including:

- Business Intelligence application developers
- Warehouse architects, designers, and developers—especially SQL and PL/SQL developers
- Data analysts and those who develop extract, transform, and load routines
- Developers of large-scale products based on data warehouses
- Warehouse administrators
- System administrators
- Other MIS professionals

In order to use the information in this guide, you need to be comfortable with the concepts of Relational Database Management Systems and Data Warehouse design. For information on data warehousing, refer to the *Oracle Database Data Warehousing Guide.* Also, you need to be familiar with Oracle's relational database software products such as Oracle Database, SQL*Plus, SQL*Loader, Oracle Enterprise Manager, and Oracle Workflow.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. 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 JAWS, a Windows screen reader, 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, JAWS 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.

How This Guide Is Organized

The Oracle Warehouse Builder User's Guide contains the following chapters and appendixes.

- Chapter 1, "Overview of Oracle Warehouse Builder", describes the overall product architecture and capabilities, as well as the benefits, of using Warehouse Builder. It also describes how to logon and introduces the components displayed in the main console and preferences and options you can set up.
- Chapter 2, "Getting Started with Warehouse Builder", describes how to start building projects using Warehouse Builder features.

Part I "Defining Source and Target Objects" , addresses creating and configuring the metadata definitions of objects from which you source data and to which you load data. This part includes the following chapters:

- Chapter 3, "Defining Oracle Data Objects", describes how to create definitions for relational and dimensional target objects in Warehouse Builder. The relational objects include Tables, Views, Materialized Views, Sequences, and External Tables. The dimensional objects include Dimensions and Cubes.
- Chapter 4, "Importing Data Definitions", describes how to import definitions for data sources using the Warehouse Builder import wizards. This chapter describes how to import data definitions from Oracle and non-Oracle database systems and flat file sources.
- Chapter 5, "Configuring Data Objects", describes how to assign physical properties to the target and source objects you defined in your logical model. The chapter first describes how to create indexes and partitions for design objects such as tables, materialized views, dimensions, and cubes defined in Warehouse Builder. The chapter then discusses how to assign physical properties to Warehouse Builder object definitions.

Part II "Designing ETL Objects" , addresses designing and configuring the extraction, transformation, and loading processes that you use to move data. ETL objects include mappings, transformations, and process flows. This part includes the following chapters:

- Chapter 6, "Designing Mappings", describes how to create and design mappings to extract data from sources, transform it, and load it into targets.
- Chapter 7, "Debugging Mappings", describes the Warehouse Builder debugging capabilities for data flows within the Mapping Editor.
- Chapter 8, "Using Mapping Operators", provides details on how to use the different operators in a mapping to transform data. This chapter also describes how to use the Expression Builder to create expressions that operate on data as it flows from a source to a target.
- Chapter 9, "Using Transformations" provides details on how to import, create, and use transformations in mappings.
- Chapter 10, "Designing Process Flows", describes how to create and design process flows that interrelate mappings and other activities internal or external to Warehouse Builder (such as email, FTP commands, and operating system executables).
- Chapter 11, "Configuring ETL Objects", describes how to configure mappings and process flows for deployment to a specific database instance. It also discusses the advantages and provides instructions on how to use partition exchange loading for faster performance.

Part III "Deploying and Executing", addresses deploying the objects you have defined and executing the processes you have designed to move data. This part includes the following chapters:

- Chapter 12, "Validating Objects", describes how to verify the integrity of the objects you have defined and identify any problems or possible errors that would occur during deployment. It also describes how you can view generated scripts (for example, DDL or PL/SQL scripts).
- Chapter 13, "Deploying Target Systems", describes how to deploy the physical instance of your target data system. It describes how to upgrade your target system and how to save deployment scripts for future uses. It also describes how

to run mapping and process flow scripts that have been previously deployed to a target data system.

- Chapter 14, "Auditing Deployment and Execution", describes how to use Warehouse Builder's web-based Runtime Audit Browser to view information about deployments and executions.

Part IV "Managing Metadata", addresses metadata management tasks, such as import, export, and snapshots for purposes of backup and versioning, and metadata browsing and reporting. This part includes the following chapters:

- Chapter 15, "Importing and Exporting with the Metadata Loader (MDL)", describes the utilities provided to import and export metadata for the purposes of backup and upgrade.
- Chapter 16, "Metadata Change Management", describes how to use metadata snapshots to perform metadata change management. It describes how to use snapshots to backup, versioning, compare, and restore metadata.
- Chapter 17, "Metadata Browsing and Reporting", describes how to use Warehouse Builder's web-based Design Browser to browse metadata and view metadata reports.
- Chapter 18, "Managing Warehouse Builder Browser", describes how to configure Warehouse Builder's portal-based Design Browser and how to create custom metadata reports.
- Chapter 19, "Extending Warehouse Builder Functionality", describes the plug-in interfaces for the Warehouse Builder Security Registration package (including how the central schema owner can register all the prospective Warehouse Builder users to the repository), the Warehouse Builder Metadata Loader, and the Warehouse Builder Runtime Platform.

Part V "Integrating and Extending Warehouse Builder", discusses the features that allow you to integrate Oracle Warehouse Builder with other Business Intelligence products. This part includes the following chapters:

- Chapter 20, "Data Quality: Name and Address Cleansing", describes how to perform parse table maintenance if you have purchased Oracle Pure Name and Address and are using the Name and Address operator to cleanse customer data.
- Chapter 21, "Using SAP R/3 Data in Warehouse Builder", describes how Warehouse Builder integrates with SAP source systems. You can import definitions for SAP data into a Warehouse Builder repository, use the SAP definitions in mappings, generate ABAP or PL/SQL code, and deploy it to your target. This chapter also describes how to extract and load SAP data from the source system into a target.
- Chapter 22, "Integrating Warehouse Builder Metadata with Other BI Products", describes the utilities provided for interchanging Warehouse Builder metadata with that of other Business Intelligence products.

Part VI "Appendix", contains material referenced by earlier parts. This part includes the following appendixes:

- Appendix A, "Keyboard Shortcuts" lists keyboard shortcuts for Warehouse Builder commands.
- Appendix B, "Reserved Words", lists the words reserved for Warehouse Builder.
- Appendix C, "Using the XML Toolkit", describes how to retrieve and store data from XML documents in a target schema.

- Appendix D, "Warehouse Builder Public Views", lists the Warehouse Builder Public Views that can be accessed through SQL. These views can be used to extend the reporting capabilities of Warehouse Builder.
- Appendix E, "Public Objects", lists the first, second, third, and fourth class objects on the navigation tree. You can use this information to better understand the behavior of user-defined properties and metadata snapshots.

New in 10*g* Release 1 (10.1)

Enhancements to the Mapping Editor: Mapping Debugger

Warehouse Builder now provides you with extensive debugging capabilities for your mappings from within the Mapping Editor. Use the Mapping Debugger to locate logical design errors in your mappings. The new features allow you to step through the data flow of a mapping using comprehensive debugging functions such as setting breakpoints and watches and interactively changing test data.

Enhanced Support of Multiple Targets: Correlated Commit

This release introduces a new commit strategy for mappings with multiple targets. In previous releases, Warehouse Builder performed independent commits. That is, Warehouse Builder committed and rolled back each target separately and independently of other targets. In addition to this option, Warehouse Builder now also performs correlated commits. 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.

Direct Partition Exchange Loading

In previous releases, Warehouse Builder by default created a temporary table for mappings that required additional processing of source data before exchanging partitions. This occurred when the mapping contained remote sources or multiple sources joined together. Beginning in this release, you can now by-pass the creation of a temporary table and directly swap a source into a target. Use Direct PEL in a mapping to instantaneously publish fact tables that you loaded in a previously executed mapping.

Data Quality Features

- **Multiple Name and Address Software Providers:** Beginning in this release, Warehouse Builder is compatible with multiple certified Name and Address software providers. Third-party vendors can license Name and Address software directly to you for use with Warehouse Builder. This enables you to choose a name and address provider whose offering is the most appropriate for your project.
- **Name-Address Operator Wizard:** In previous releases, you defined the Name-Address operator using the mapping canvas and the operator Configuration Properties sheet. For improved usability, Warehouse Builder now enables you to use a wizard and Operator editor to create and edit the Name-Address operator.
- **Match-Merge Operator:** Warehouse Builder incorporates the data quality functionality formerly available in Oracle Pure Integrate. You can use the Match-Merge operator available in the Mapping Editor to define business rules for matching and merging records. The Match-Merge operator together with the

Name-Address operator support householding, the process of identifying unique households in name and address data.

Metadata Change Management

In a previous release, you could perform metadata change management using the OMB Plus scripting utility. Beginning in this release, you can also access these functions from the Warehouse Builder client user interface. Metadata change management enables you to take snapshots of metadata objects and use them for backup and history management. Snapshots are supported for any object on the navigation tree and can store information about an object alone (such as a table or module), or the objects within it as well (such as the tables within a module).

Extending Oracle Warehouse Builder Functionality

- **Security:** Warehouse Builder now provides advanced repository security and auditing options that you can implement according to your security requirements. The advanced security options include the following:

  Proactive Security: Warehouse Builder enables you to plug in a customized security PL/SQL implementation package in the Warehouse Builder repository to provide tailored access control to users according to the security rules defined by your organization.

  Reactive Security: Warehouse Builder enables you to track audit information based on the metadata history and to determine security policies from such audit information.

  Data Stewardship: Warehouse Builder enables an individual or a group of individuals to "own" portions of the metadata rather than the technical administrators. Metadata ownership thus becomes an important component of metadata security management.

- **RAC Support:** With the 9.2 release, Warehouse Builder provides increased support for RAC features. Warehouse Builder now supports the use of net service names in the runtime. This enables you to plan maintenance of nodes in a cluster without having to reconfigure the runtime environment. Warehouse Builder also provides an increased availability in the runtime service. For example, if either the service instance or its associated node fails or is taken out of service, then the runtime service instance on a different node can take over. While the Warehouse Builder design repository can also be used in a RAC cluster, it will not take advantage of any failover features of RAC for this release.

Enhancements to Flat File Support

- **ZONED Data Type Support:** Warehouse Builder now enables you to load fixed format data files containing ZONED decimal data. In the Flat File Sample Wizard, specify the ZONED data type for a flat file you import. The format for ZONED data is a string of decimal digits, one for each byte, with the sign included in the last byte. (In COBOL, this is a SIGN TRAILING field.) The length of this field is equal to the precision (number of digits) that you specify. You may also specify a scale, which is the number of digits to the right of the decimal point.

- **DECIMAL Data Type Support:** DECIMAL data is in packed decimal format; two digits for each byte, except for the last byte, which contains a digit and sign. The DECIMAL data type includes precision and scale and therefore can represent fractional values.

Enhancements in Database Connectivity

Warehouse Builder now enables you to create public database links that can be shared across a database. Public database links can be created by repository owners, as well as any user with the `CREATE PUBLIC DATABASE LINK` privilege.

Warehouse Builder Available on HP-UX and AIX

Starting with this release, Warehouse Builder is available on HP-UX and AIX platforms. This new availability is an addition to the UNIX (Solaris and Linux), and Windows (NT, 2000, and XP) platforms, which have been available from previous releases. (Note that the MITI Bridges feature is only available on Windows platforms and the Name and Address Server is only available on Windows and Solaris platforms.)

Public Application Programming Interface

Starting in this release, Warehouse Builder now includes a Public Application Programming Interface (API). To access the API, unzip and extract the following file to a folder on your local machine:

```
<owb home directory>\owb\lib\int\pubapi_javadoc.jar
```

Double click the file `index.html`. Select the Help link for information on how to use the API.

Added in 10*g* Release 1 (10.1)

The following new features were introduced in Oracle Warehouse Builder :

Changes in the Warehouse Builder Console

- **Enhanced Navigation Tree:** The navigation tree that displays in the Warehouse Builder console has been enhanced to improve navigation between projects and facilitate direct access to metadata repository objects. All projects are now visible from the tree, whereas previously you could only see one project at a time. Now you can expand a project node to display the contents of the active project. The module tree no longer appears in a separate window.
- **Wizards, Editors, and Properties Sheets:** All Warehouse Builder wizards, editors, and properties sheets are now launched from the navigation tree.
- **Business Areas Renamed to Collections:** In previous releases, you could create business areas in warehouse modules to organize objects in Warehouse Builder and to export metadata to tools such as Oracle Discoverer. Starting in this release, *collections* replace *business areas* in all functions and introduce enhancements, such as the ability to import metadata into and export metadata from a collection.
- **Fact Tables Renamed to Cubes:** The terms *fact* and *fact table* have been replaced with *cube* in this release to be in line with OLAP industry standards.
- **Logical Names Renamed to Business Names:** All references to *logical names* of objects have changed to *business names* in this release.
- **Toolbars in the Warehouse Builder Console:** The utility drawer has been removed and the side and top toolbars in the Warehouse Builder console have been merged at the top to consolidate the most important functionality in one place.

Enhancements to Deployment

- **Addition of Deployment Management Objects**: This release introduces three object types to assist in managing connections to deployment sources and targets: Locations, Connectors, and Runtime Repository Connections. Locations define the physical location of the deployment. Connectors define relationships between locations. Runtime Repository Connections provide information about Runtime Repositories. Using these objects, you can create multiple deployment targets for the same target design.
- **Single Deployment Management Interface:** The Deployment Manager provides a single interface for managing deployments of all objects, and executions of deployed mappings, transformations, and process flows. It also provides immediate access to the history of previously deployed objects. Not only does the Deployment Manager enable you to perform all these tasks from one interface, but Warehouse Builder now keeps track of runtime metadata, providing you with the history of what has previously been deployed.

Enhancements to Warehouse Builder Metadata Browser

- **Design Metadata Browsing:** The Warehouse Builder Design Browser has been enhanced to include all new exposed objects, such as external tables, locations and connectors. In addition, you can now launch the Design Browser as a standalone executable; it no longer requires Oracle DatabaseAS to be installed for a single-user usage.
- **Runtime Metadata Browsing:** The Warehouse Builder Runtime Audit Viewer has been replaced by the Runtime Audit Browser, which provides web-based reporting. The Runtime Audit Browser provides a more extensive set of deployment and execution audit reports than was available in previous releases. This audit data comes from information stored in the Runtime Repository and includes both deployment and execution data.

Enhancements to Warehouse Builder Programmatic Access

- **Warehouse Builder Public APIs:** Starting with this release, Warehouse Builder offers this alternative for programmatic access to Oracle Warehouse Builder features: a full set of Java public APIs for application programmers who want to embed Warehouse Builder features and services in their own applications.
- **Warehouse Builder Scripting Language:** Oracle MetaBase (OMB) Scripting Language provides access to all Warehouse Builder functions without accessing the Warehouse Builder graphical user interface. Users can access Warehouse Builder metadata and functionality by using OMB Plus, Warehouse Builder's scripting utility. This gives developers the power of using Warehouse Builder programmatically and extending its functionality where required. For more information on OMB Scripting Language, refer to the Oracle Warehouse Builder Scripting Reference .

Enhancements to Metadata Management

- **Security:** Warehouse Builder now provides an optional repository security and auditing system that you can implement according to your security requirements. You can create a multiple user account system where multiple identifiable users can access the same Warehouse Builder repository. Warehouse Builder also enables you to plug in a customized security PL/SQL implementation package in the Warehouse Builder repository to provide tailored access control to users according to the security rules defined by your organization.

- **Metadata Change Management (Metadata Snapshots):** Starting in this release, you can take snapshots of metadata objects and use them for backup and history management. Snapshots are supported for any object on the navigation tree and can store information about an object alone (such as a table or module), or the objects within it as well (such as the tables within a module).
- **Multiple Language Support (MLS):** With this feature, you can store the displayed business names and descriptions in languages other than the base language of the repository. Your different translations of business names and descriptions can be used to deploy to an EUL in the language of the target user population.
- **Extensibility Through User-Defined Properties:** Users can define additional properties for any Warehouse Builder objects using the Warehouse Builder OMB Plus scripting utility. After you define user-defined properties through scripting, you can access them in the user interface, the Oracle MetaBase (OMB) Scripting Language, Oracle Warehouse Builder Java API Reference s, and Warehouse Builder Design Browser. This enhances the extensibility of Warehouse Builder and makes it easier to integrate it with other Business Intelligence products.
- **Metadata Loader (Import and Export) Flexibility Enhancements:** Two new features were added to enhance this area of the product. The first is the ability for you to export metadata directly from Collections. The second feature is available from the Metadata Loader command line utility. It provides you with flexibility to specify the type of actions you want to apply when you import a first-class object.

Process Flow Editor

Starting in this release, you can use the Process Flow Editor in Warehouse Builder to create and define process flows. External process operators that you previously defined in mappings are upgraded to user-defined processes and are contained within a process flow module. Process flows now integrate in the same Warehouse Builder design environment and no longer require you to use Oracle Workflow design client to perform these functions. The Warehouse Builder process flow modeler natively understands the semantic of your mappings and enables you to model activities such as FTP, and email.

Performance Improvements

- **Mapping User Interface:** A new pre-defined display set, named Mapping, was added in this release. Selecting this display set causes the Mapping to only display columns that effectively are mapped, or used.
- **Mapping Compression:** This feature automatically detects unused connections between operators and attributes in any given mapping and eliminates them from the repository. This dramatically enhances the performance of loading and storing large mappings that represent significant data flows.
- **Metadata Loader (Import and Export):** Import and export functionality now takes advantage of the new compression feature available for each mapping. This means that the Metadata Loader now exports and imports only those mapping objects that are actually used.

Oracle Database Integration

- **OLAP Integration:** Warehouse Builder enables you to design, deploy, and load multidimensional OLAP objects as ROLAP or MOLAP models from different data sources. *A*fter the data is loaded, you can use BI tools and applications to *run complex analytical queries that answer your business questions. Using* Warehouse

Builder, you can now create and manage both your relational and *and multidimensional objects from the same cube and dimension designs.*

- **Advanced Queue (AQ) Integration:** Warehouse Builder enables you to import Advanced Queue definitions and to use AQs as data sources and targets while designing your data warehouse. Through Advanced Queue functionality coupled with the Messaging Gateways, Warehouse Builder enables you to support messaging applications on MQ Series and Tibco as Warehouse Builder data sources. AQs also enable you to propagate change data capture from your source system to your target. The ability to integrate AQs lays the foundation for providing real time data warehousing in the future.
- **External Tables:** Starting in this release, you can use external tables to represent data from non-relational file sources in a relational, read-only format. You can import an existing external table from an Oracle Database database. Or you can create an external table in Warehouse Builder based on a flat file definition. Warehouse Builder will generate the right DDL for you to deploy you external table to an Oracle Database database.
- **Oracle Database Multiple Table Inserts:** Warehouse Builder takes advantage of Oracle Database database functionality and generates a multiple-table insert statement when the target database is Oracle Database. This enables you to optimize mappings to insert data into multiple tables in one operation.
- **Oracle Database Table Functions:** Warehouse Builder introduces the Table Function operator that enables you to improve performance when loading your target system. Use this operator to develop custom code that can manipulate a set of input rows and return another set of rows possibly of different cardinality. Unlike conventional functions, table functions output a set of rows that can be queried like a physical table.

Enhanced Support for Flat Files

- **Unbound Flat Files as Targets:** In this release, you can create a new, unbound flat file object as you create your mapping. Warehouse Builder creates a new comma-delimited, single-record-type flat file in the specified location. This feature makes it easier to load the contents of a relational object into a flat file.
- **Outbound Reconcile for Flat Files:** Outbound reconciliation makes it possible to create a new repository object from a mapping flat file. This results in a new, comma-delimited file to be created where specified, if the flat file is new to that repository. This feature makes it easier to "quickly dump" the contents of a relational object to a flat file.
- **Logical Records for Delimited Files:** The Flat File Sample Wizard has also been enhanced to display an improved user interface that enables you to define logical records for delimited files.
- **Position-Based Master-Detail Loading:** Position-based master-detail flat files are now easier to load with the use of additional mapping operators.
- **SQL Property Extensions:** You can now specify SQL properties for flat files you import into Warehouse Builder. This enables you to pre-define SQL property values for each flat file field. Thus, if mapping a flat file source to a relational target, the target column will default to these pre-defined SQL property values. These values will be used when building a relational target column or when creating an external table column.

Mapping Editor Enhancements

- **Mapping User Interface:** A new set of property tabs is now available for you to quickly create and edit mapping operators and attribute properties.
- **Pivot and Unpivot Operators:** Starting in this release, you can add a pivot operator or an unpivot operator to a mapping. The pivot operator enables you to transform a single row of attributes into multiple rows. The unpivot operator converts multiple input rows into one output row.
- **Name and Address Operator Enhancements:** The Name and Address operator has been enhanced to include new input roles and output attributes. The United States Postal Service Code Accuracy Support System (CASS) reporting is also supported starting with this release.

Warehouse Builder Is Now Available on UNIX Platforms

Starting with this release, Warehouse Builder is available on UNIX (Solaris, and Linux), as well as Windows (NT, 2000, and XP) platforms. This applies to all the components of Warehouse Builder, with the exception of the Name and Address libraries, which are not available on Linux in this release. (Note that the OLAP Bridges feature is only available on Windows platforms and the Name and Address Server is only available on Windows and Solaris platforms.)

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.

Related Publications

The Warehouse Builder documentation set includes these manuals:

- Oracle Warehouse Builder User's Guide
- Oracle Warehouse Builder Installation and Configuration Guide
- Oracle Warehouse Builder Transformation Guide
- Oracle Warehouse Builder Scripting Reference
- Oracle Warehouse Builder Release Notes

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

Oracle provides additional information sources, including other documentation, training, and support services that can enhance your understanding and knowledge of Oracle Warehouse Builder .

- For more information on Oracle Warehouse Builder technical support, contact Oracle World Wide Support services at:

  http://www.oracle.com/support
- For the latest information on, and downloads of, software and documentation updates to Oracle Warehouse Builder , visit MetaLink at:

  http://metalink.oracle.com
- You can order other Oracle documentation at:

  http://oraclestore.oracle.com

Contacting Oracle

Oracle Metalink

Metalink is the Oracle support Web site where you can find the latest product information, including documentation, patch information, BUG reports, and TAR entries. Once registered, you can access email, phone and web resources for all Oracle products. MetaLink is located at:

http://metalink.oracle.com

Check MetaLink regularly for Warehouse Builder information and updates.

Documentation

You can order Oracle product documentation by phone or through the World Wide Web:

- **Phone:** Call 800-252-0303 to order documentation or request a fax listing of available Oracle documentation.
- **Oracle Documentation Sales Web site:** http://oraclestore.oracle.com
- **Oracle Support Services Web site:** http://www.oracle.com/support

1

Overview of Oracle Warehouse Builder

Oracle Warehouse Builder is a business intelligence tool that provides an integrated solution for designing and deploying enterprise data warehouses, data marts, and business intelligence applications. It solves the complex problem of data integration between dispersed data sources and targets. In addition, Warehouse Builder provides all the necessary functionality to maintain the life cycle of the system you develop.

The chapter is divided into the following sections:

- Product Architecture and Capabilities on page 1-1
- Benefits of Using Warehouse Builder on page 1-5

Product Architecture and Capabilities

Warehouse Builder is a complete design and implementation tool for building and managing data warehouses and business intelligence systems. It combines the key components of both an extraction, transformation, and loading (ETL) tool and a design tool into one product. In addition, Warehouse Builder leverages Oracle database technology. It is the central point of integration of the Oracle Business Intelligence tools suite and provides integration with ad-hoc query tools as well as OLAP and relational database features.

The architecture of Warehouse Builder is comprised of two components, the design environment and the runtime environment. Each of these components handles a different aspect of the system. The design environment manages the metadata, while the runtime environment handles the physical data.

The Design Component

The Warehouse Builder design component consists of a highly scalable metadata repository that is stored in an Oracle database and a set of client design and reporting tools written in Java or HTML. Using these tools, metadata can be viewed and manipulated.

Creating metadata is a design activity that uses editors in the client tool to design objects, processes and jobs. This interactive way of creating metadata is typically used to design a new system. Warehouse Builder supports the design of relational database schemas, multi-dimensional schemas, ETL processes and End User tool environments through the client.

Source systems play an important role in any ETL solution. Instead of creating metadata manually, Warehouse Builder provides integrated components that import the relevant information into its repository.

One of the strengths of the architecture is that it supports life-cycle management which enables metadata to be updated based on changes in the source systems. Warehouse Builder then facilitates propagating these changes to the ETL processes and the target systems.

To ensure the quality and completeness of the metadata in the repository Warehouse Builder provides extensive validation within the repository. Validation helps to keep a complex system that is created by multiple users in an accurate and coherent state.

To further aid in the development and evaluation of the metadata, a web based metadata reporting environment is available. The reporting environment enables developers and business users to browse and investigate system elements without using the design tools. A very important component of this reporting environment is the Impact Analysis capabilities allowing the identification of the impact of changes throughout the system before they are made. Impact Analysis reporting enables you to have better control on changes and better planning for the implementation of these changes. The opposite capability, tracing back where data originated from, is called Data Lineage reporting and is also provided in Warehouse Builder.

The Runtime Component

Once the user has designed the ETL system on a logical level, he needs to move it to the physical database environment. Before this can be done, information about the database environment is added to the logical design when the target is configured for deployment. After the configuration is complete, code can be generated.

Warehouse Builder generates extraction specific languages for the ETL processes and SQL DDL statements for the database objects. The generated code is deployed, either to the file system or into the database.

Performing the ETL functions means running the deployed code in the database. This can be done using the Warehouse Builder Deployment Manager or from an external tool such as Oracle Enterprise Manager. The ETL process then pulls the source data into the target database . This can be a staging area, an operational data store, a warehouse or any other schema. The code sections external to the Oracle database are executed in their respective environments. For example, ABAP code to extract from SAP systems is run in the SAP environment.

To allow reporting on data loads, the code generated by Warehouse Builder contains audit routines. These routines write information about the ongoing load into the Warehouse Builder runtime tables. Information captured while running the code can include the number of rows selected, inserted and updated. If an error occurs while transforming or loading data, the audit routines report the errors into the runtime tables. To allow easy access and convenient reporting on this runtime information, Warehouse Builder provides the Runtime Audit Browser.

Dependency management and scheduling is provided by a close integration with specific Oracle tools. Oracle Enterprise Manager is Oracle’s scheduling and database management tool. Warehouse Builder creates jobs in the Oracle Enterprise Manager repository, which can be scheduled and monitored along with other database activities. Through the interaction with Oracle Workflow (OWF) the Warehouse Builder user can create full-blown processes for dependencies between the ETL processes including notifications.

How Warehouse Builder Achieves Its Goal

Creating a business intelligence application is a complex process. It involves various steps and phases, that may span a great number of systems, resources, and functional areas. Warehouse Builder reduces this complexity as it enables you to manage these tasks from a single interface while ensuring scalability, reliability and flexibility by leveraging the latest Oracle database technology.

Key Warehouse Builder functions include:

- Importing source data definitions.
- Designing and creating target database schema.
- Defining data movement and transformation between sources and targets.
- Assigning dependencies between ETL processes.
- Managing and updating source definitions.
- Deploying, upgrading, and managing target schemas.
- Designing and creating an ad-hoc query tool environment.
- Designing and creating an OLAP environment.

Warehouse Builder Components

Warehouse Builder is organized into the following key components:

Warehouse Builder Client Application

The Warehouse Builder client application provides an easy to use graphical interface that enables you to define, design, and deploy business intelligence systems. Many components contribute to each part of the process. The code generator and Deployment Manager are the components of the client application that control the creation and management of the systems you create.

Code Generator This component generates scripts such as DDL and PL/SQL based on the metadata in the repository. The generator is designed to utilize the Oracle8*i* and 9*i* database features. The generated scripts provide optimal performance for Oracle database systems.

Deployment Manager This component manages all aspects of deployment and deployed objects. You can select objects for deployment and determine how you want those objects deployed. You can then execute deployed objects. You can also manage the lifecycle of your system through immediate access to deployment history.

Warehouse Builder Runtime Platform Service

The Runtime Platform Service is the server-side component of Warehouse Builder software that provides execution and deployment services. In order for you to be able to run these services, the Runtime Platform Service must be active.

The Runtime Platform Service manages the execution of mappings and process flows from within Warehouse Builder and ensures that all execution and deployment audit data is stored in a Runtime Repository. For remote executions, it connects to Oracle Enterprise Manager's Management Server. The Runtime Platform Service is invoked through a database job that is automatically started when the database is started and shut down when the database is shut down.

Warehouse Builder Design Repository

The Design Repository, installed in an Oracle database, stores the metadata definitions for all of the objects used in Warehouse Builder. This is where all of the design information is stored for the target systems you are creating. You can access metadata stored here using the client user-interface, or through OMB Plus, the Warehouse Builder scripting utility.

Warehouse Builder Runtime Repository

The runtime repository, installed in an Oracle database, stores all of the deployment data as well as data from executed mappings and process flows. This is where the target environment information is stored for the business intelligence systems you create. This includes connection information for all of the deployment locations.

Audit Reporting Browser

This reporting browser enables you to view your deployment and execution audit information from a web-based application. The reports are based on data stored in the Runtime Repository.

Metadata Reporting Browser

This reporting browser enables you to view your metadata stored in the design repository from a web-based application and provides information to larger audiences. The information is organized into a format targeted for business-oriented users. The reports are based on data stored in the Warehouse Builder Design Repository.

Warehouse Builder Objects

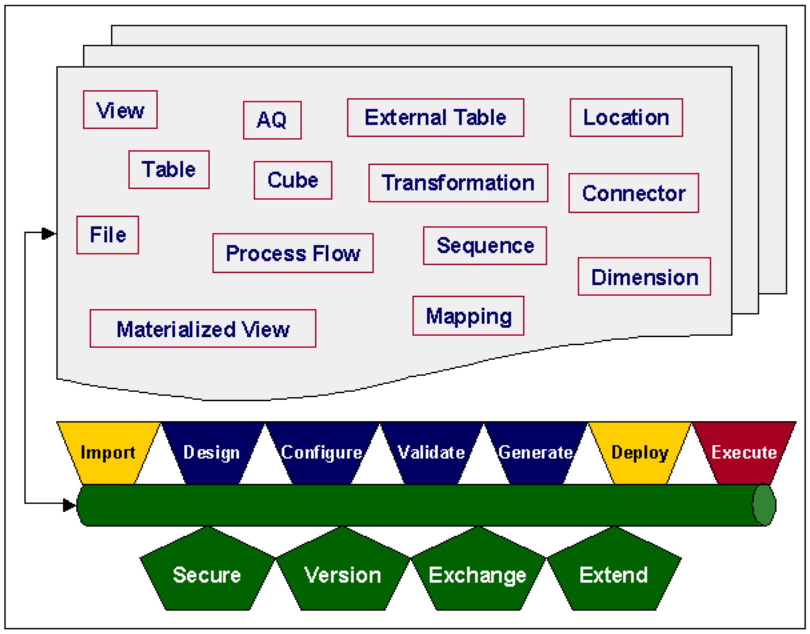
Warehouse Builder contains a catalog of objects known as First Class Objects. These include objects that can be imported, designed, and deployed.

Warehouse Builder First Class Objects include the following:

- Advanced Queues
- External Tables
- Connectors
- Cubes
- Dimensions
- Files
- Locations
- Mappings
- Materialized Views
- Process Flows
- Sequences
- Tables
- Transformations
- Views

First Class Objects include the set of objects that can be run through Warehouse Builder services. Figure 1–1 shows the First Class Objects along the top of the diagram and the available Warehouse Builder services on the bottom.

Figure 1–1 First Class Objects and Services



Warehouse Builder Deployment Targets

When you design targets in Warehouse Builder, you have a wide variety of deployment targets from which to choose. Depending on your business intelligence needs you can decide how you want to deploy your design. You can use Warehouse Builder to deploy to any of the following systems:

- Relational Databases
- OLAP Databases
- Oracle Enterprise Manager
- Oracle WorkFlow
- Oracle BI Beans
- Oracle Discoverer
- CWM Bridges

Benefits of Using Warehouse Builder

Warehouse Builder offers the following benefits for data integration and business intelligence system deployment:

- **Rapid Design:** Warehouse Builder reduces the design time by providing easy to use visual editors, wizard-driven user interfaces, and a library of predefined transformations.
- **Centralized Design:** All information about the design of the system is stored and managed in one central place, the Warehouse Builder design repository. This is a single source of the entire design and eliminates the extra cost of inconsistencies caused by reducing duplication of metadata.
- **Reduce Time to Incorporate Changes:** Advanced Life Cycle management features based on the single repository enable a smooth maintenance process.

- **Error-Free Code:** Because it is generated from a design stored in one place, the code is not only error-free, but also easy to re-create, upgrade, and maintain.
- **Leverage Technology Investments:** By using the Oracle database as both a transformation engine and storage facility, Warehouse Builder leverages Oracle scalability, performance, security and high availability.

2

Getting Started with Warehouse Builder

Warehouse Builder provides many wizards, editors and tools that assist you with designing and deploying business intelligence systems. This chapter provides a brief introduction of how to use Warehouse Builder to design and create your target system.

This chapter includes the following topics:

Overview of Creating a Business Intelligence System

Warehouse Builder enables you to create business intelligence solutions by integrating data from disparate sources. There are many reasons behind building data systems from existing sources. For many, this allows data to be consolidated and transformed into information that can be used to make business decisions. Warehouse Builder makes the process of designing such a system simple by providing easy to use wizards and editors. After you design a system, Warehouse Builder supports several types of deployment targets, including data warehouses, data marts, and business intelligence applications.

To build a target system, use the following steps:

- Step 1: Creating a Project
- Step 2: Defining Source and Target Modules
- Step 3: Defining Data Movement and Transformation
- Step 4: Validating and Generating
- Step 5: Deploying and Executing

Step 1: Creating a Project

Create a new project to begin designing your target system. Warehouse Builder projects are the largest storage objects within the Warehouse Builder design repository.

Projects store and organize metadata definitions for your target system. These definitions include data sources, target warehouse objects, mappings from source to targets, transformation operations, and configuration parameters. These definitions are organized into folders within the project. Typically a project contains related metadata. Creating multiple projects enables you to create your design in an organized manner.

Step 2: Defining Source and Target Modules

Define source and target modules to begin designing the model of your system. Modules are storage objects within projects that help to organize source and target objects. Source modules contain the metadata from existing source systems from which you are pulling. Target modules contain the metadata you are designing. You can create these modules in either order, source then target or target then source.

Defining Sources

Create source modules by selecting data from existing source systems. There are three types of source modules:

- Databases
- Files
- Applications

When you define data sources in Warehouse Builder, you import logical definitions of the sources and objects to be included in the model. These definitions are stored within a project in the design repository. The steps you take to create source modules depend on the type of sources you are using. For more information, see Chapter 4, "Importing Data Definitions".

Defining Targets

Define targets by creating a model of the target warehouse you intend to create upon deployment. Use Oracle database warehouse modules to store the metadata definitions. You can create and design new objects, or you can also import definitions from source data locations. For more information, see Chapter 3, "Defining Oracle Data Objects".

Defining Locations and Connectors

When you define a source or target module you must also register them to a location for deployment to be successful. Locations define the type and version, when appropriate, of a source or target module. Locations are saved in the design repository as logical definitions. During deployment, the connection information is requested and saved in the runtime repository.

You must also define connectors for any module that is linked through a mapping to another module. The connector provides a link between the locations of each module. Connector definitions are saved in the design repository as logical definitions and may be used to generate database links during deployment if the two locations, linked by the connector, are located on separate machines. For more information, see "Defining Connectors" on page 3-5 and "Defining Runtime Repository Connections" on page 13-2.

Configuration

Configure objects in the target module to define the physical characteristics of the objects. There are many configuration parameters you can use to define your target.

You can also use the default values. For information about configuring data objects, see Chapter 5, "Configuring Data Objects".

Step 3: Defining Data Movement and Transformation

After defining source and target modules, create data movement and transformation logic. This is also known as ETL logic. The majority of design work you do in Warehouse Builder is defining the ETL logic. You can use the following components to define data movement and transformation in Warehouse Builder:

- Transformations
- Mappings
- Process Flows

Use these objects to define how you want to manipulate source data to create your target system. There is a large set of mapping operators available within the Mapping Editor as well as a full library of transformations. For more information, see Part II "Designing ETL Objects".

Configuration

Configure mappings and process flows to define the physical characteristics of the objects. Use the configuration parameters to optimize your target system deployment. You can also use the default values. For information about configuring data movement and transformation objects, see Chapter 11, "Configuring ETL Objects".

Step 4: Validating and Generating

After you define source and target modules and define the ETL logic you can validate and generate the code to check for errors prior to deployment.

Validation

You can validate all objects and modules to ensure that no definitions are invalid or incomplete. If you find invalid definitions, use the editors to correct the problems. If all definitions are valid, you are ready to configure the modules and their components for deployment. For more information, see "About Validation" on page 12-1.

Generation

You can generate and view scripts prior to deployment. The type of code generated depends on the configuration parameters defined for the type of target. When you generate the code, you can also save it if you want to deploy it from outside Warehouse Builder. For more information, see "Viewing Generated Scripts" on page 12-5.

Step 5: Deploying and Executing

Deploy the design to the target environment after the target model is complete. You must install a runtime repository and define a Runtime Repository Connection before deployment. Warehouse Builder provides two methods for deployment. You can deploy directly from the main console or you can use the Deployment Manager. You can then execute mappings and process flows from the Deployment Manager. You can also validate and generate objects prior to deployment.

Creating a Runtime Repository Connection

When you deploy objects, you must select a Runtime Repository Connection. The connection you specify defines the Runtime Repository that stores the deployment data. Before you can use deployment, you must install Runtime Repositories and create Runtime Repository Connections in the navigation tree. Runtime Repository Connections describe a connection to the Runtime Repository that you use to manage deployments. For more information about the Runtime Repository, see the Oracle Warehouse Builder Installation and Configuration Guide.

Using the Deployment Manager

Use the Deployment Manager to assist you in the deployment and execution process. The Deployment Manager provides a complete deployment platform as well as intelligent components that allow you to manage the deployment and future upgrades and execute mappings and process flows immediately. The Deployment Manager also enables you to view data about previously deployed objects in order to make decisions about upgrading the system. For more information about deployment and execution and the Deployment Manager, see Chapter 13, "Deploying Target Systems".

Viewing Deployment and Execution Audit Reports

Use the Warehouse Builder Runtime Audit Browser to view reports about deployments and execution of mappings and process flows. When you deploy objects to a target or execute scripts, the runtime repository stores data about each deployment and execution. This data can be accessed in reports using either the Runtime Audit Client-based Browser or the Runtime Audit Portal-based Browser. For more information, see Chapter 14, "Auditing Deployment and Execution".

Accessing the Warehouse Builder Repository

To log in to your Warehouse Builder client, you need to create a new Warehouse Builder repository schema in your database. The Warehouse Builder repository stores definitions for all the objects created or imported into Warehouse Builder to design your data warehouse. You use the Warehouse Builder Repository Assistant to create a new repository to run the Warehouse Builder client and access the Warehouse Builder metadata.

Optionally, Warehouse Builder enables multiple user schemas to access the same repository. Warehouse Builder provides a repository security and auditing system by implementing the following:

- **Multiple User Account System:** Enables multiple identifiable users to access the same Warehouse Builder repository. The owner of the Warehouse Builder repository schema sets up these users.
- **Security Registration Service:** Enables you to plug-in your customized security PL/SQL implementation package in the Warehouse Builder repository. Warehouse Builder provides the correct access control according to the security rules defined in the PL/SQL package by your organization.

When you use Warehouse Builder, you can:

- Use both the multiple user account system and the security registration mechanism to implement security in Warehouse Builder.
- Use only the multiple user account system to enable multiple identifiable users to access a central Warehouse Builder repository. Although these users will leave an

audit trail, there will be no access control and all users can invoke all operations in Warehouse Builder.

- Choose not to implement the security feature or not use the multiple user account system. You can create a separate repository schema for every user or let multiple users use the same repository by sharing the repository login and password.

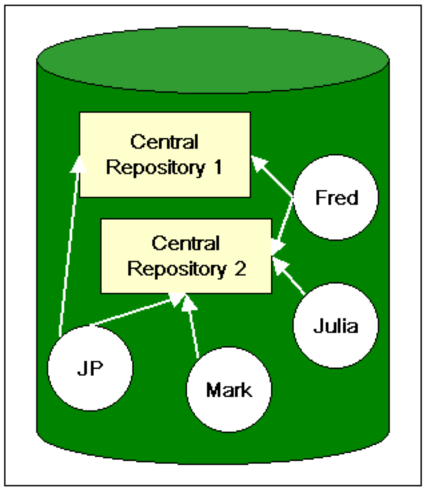
The following sections show you how to set up multiple user accounts and the security registration mechanism system in Warehouse Builder.

Multiple User Account System

The multiple user account system enables multiple user schemas to access a central Warehouse Builder repository schema using the Warehouse Builder client. This enables the central schema owner to audit and trace changes made to the same repository by different users. However, these repository users should not access the Warehouse Builder repository using other methods such as SQL*PLUS. This is enforced by Warehouse Builder.

Figure 2–1 shows how multiple warehouse users (inside circles) can access the central repositories (inside rectangles).

Figure 2–1 Warehouse Builder Repositories with Multiple Users



To enable multiple Warehouse Builder user schemas to access the central Warehouse Builder repository:

- Each Warehouse Builder user must have their own Oracle database schema created in the same database instance as the central Warehouse Builder repository.
- The central schema owner should register all the prospective Warehouse Builder users to the repository. See "Registering a Warehouse Builder User" on page 2-6.

Multiple Warehouse Builder users can now start the Warehouse Builder client session and connect to the same central Warehouse Builder repository.

Tip: Warehouse Builder users can only access the central Warehouse Builder schema using the Warehouse Builder client, not through SQL*PLUS. This is enforced by Warehouse Builder.

Registering a Warehouse Builder User

To create and register a Warehouse Builder user, the user must have a schema on the same database instance as the central Warehouse Builder repository.

To register a Warehouse Builder user who is not a database user:

1. The DBA must create a normal database user (User1) using SQL*Plus or Oracle Enterprise Manager.
2. The DBA must execute the following SQL statement to limit the default roles for USER1.

   ```
   ALTER USER user1 DEFAULT ROLE NONE;
   ```

   When a database user is first created, the user's default role setting is ALL. Because the multiple user account system requires that the user's default role cannot be set to `ALL,` you must change this setting. Otherwise, the following registration step will fail.
3. The Repository Owner must log in to the repository using SQL*Plus and type the following command:

   ```
   call WBSecurityHelper.registerOWBUser('User1');
   ```

   Now the database user User1 becomes a Warehouse Builder user and can access the central Warehouse Builder repository using the Warehouse Builder client.

To register a Warehouse Builder user when a Warehouse Builder user is already a database user:

1. The Repository Owner must log in to the repository using SQL*Plus and type the command:

   ```
   call WBSecurityHelper.registerOWBUser('user1');
   ```

   Now the database user User1 becomes a Warehouse Builder user and can access the central Warehouse Builder repository using the Warehouse Builder client.

If the preceding procedure call fails due to the following error message,

"user: User1 has the DEFAULT ROLE set to ALL.The database administrator should use ALTER USER statement to limit the default roles of user: User1.Then register the OWB user again."

then the database administrator should limit the user's default role (it should not be ALL) by using ALTER USER statement.

Other User Management Utilities

Warehouse Builder includes utility procedures for the following maintenance tasks:

- Update the role password.

  Although the repository owner does not explicitly use the protecting password for the role, it is recommended that the repository owner change the password often.

  The repository owner must connect to the database containing the repository and issue the following statements from SQL Plus:

  ```
  Call WBSecurityHelper.updateRolePwd('mewpwd');
  ```
  where `'newpwd'` is the new password used to protect the role chosen by the repository owner. The changed password encryption appears in the OWB_Role_ Info table.
- Unregister repository users

To unregister a user from a repository, the repository owner must connect to the repository schema and execute the following statement from SQL Plus:

Call WBSecurityHelper.unregisterOWBUser('username');

- List all repository users

  The repository owner must connect to the repository schema and issue the following statement:

```
Set serveroutput on;
Call WBSecurityHelper.list OWBUsers();
```

  You must use `Set serveroutput on` because `list OWBUsers()` uses the `DBMS_OUTPUT.put_line` for the output to dump the data to the user interface. Otherwise, `DBMS_OUTPUT.put_line` only dumps the output into an intermediate data structure.

How the Multiple User Account System Works

The following steps show you how the multiple user account system works in Warehouse Builder:

1. Collect the privileges required by the Warehouse Builder client.
2. During Warehouse Builder installation, run the Warehouse Builder Repository Assistant to automatically create a password protected database role called `OWB USER ROLE.`
3. Grant these privileges to the newly created `OWB USER ROLE.`
4. Whenever a new Warehouse Builder user is registered, the `OWB USER ROLE` is granted to that user through the user registration process.
5. When the Warehouse Builder client session is started, Warehouse Builder allows the users to access the central repository schema by enabling the `OWB USER ROLE.`
6. Optionally, the central schema owner can grant the `OWB PUBLIC VIEW` role to a Warehouse Builder user if they need to access the public views provided by the repository. For example, `"OWBR_"+ repos_name` is the role name of OWB public view, where the repository name is the central repository schema name. This role is also created by the OWB Repository Assistant.

Security Registration Service

Warehouse Builder provides a security registration service that enables you to extend Warehouse Builder to have access control functionality. Using this service, you can plug in your own security service implementation according to your organization's security policy and Warehouse Builder will provide the correct access control on any operation invoked by the user through Warehouse Builder. Currently there are no restrictions on READ privileges.

Warehouse Builder provides a PL/SQL package interface that you can implement if you want to plug in your own security service implementation. This PL/SQL package interface and a dummy implementation is installed in the Warehouse Builder repository. Because the dummy implementation has no intelligence, it grants all users the permission to invoke any operation in Warehouse Builder. If you want to use Warehouse Builder security, you need to replace this dummy PL/SQL package implementation with your own by installing your customized implementation in the repository.

The main procedures defined in the PL/SQL package interface are used as call out procedures by Warehouse Builder to know whether the operation invoked by the login user is acceptable or not. Your PL/SQL implementation enables Warehouse Builder to make the access control decision. The use of this PL/SQL package as a plug-in mechanism has the following benefits:

- **Security:** Because the repository owner alone has the privilege to create or update a PL/SQL package in the repository schema, no other user can substitute the plug-in package.
- **Flexibility:** You can define and enforce your own security policy by implementing this PL/SQL package.
- **Convenience:** You install the PL/SQL package body file in the repository schema through SQL*PLUS and the security mechanism is implemented.

For detailed information on how to implement the security registration service, refer to "Managing Security with PL/SQL" on page 19-5.

Privileges

In Warehouse Builder, there are two types of privileges:

- **System Privilege:** If the user is granted this type of privilege, the user can use it throughout the system no matter what the object under the operation affects. The following is a list of system privileges, also known as service type privileges:

  DEPLOY
  MDL_IMPORT
  MDL_EXPORT
  BRIDGE_IMPORT
  BRIDGE_EXPORT
  RUNTIME_EXECUTE
  SNAPSHOT_RESTORE
  SOURCE_IMPORT

- **Object Privilege:** This privilege can be connected to a particular Warehouse Builder object. You grant the user this type of privilege at an object level, object type level, module folder level (user can invoke the operation on all objects under the module), or project folder level. The following is a list of object privileges:

  EDIT
  DELETE
  REFERENCE
  VALIDATE
  GENERATION
  VERSION
  CREATE

  For a complete list of objects and their privileges, see "Managing Security with PL/SQL" on page 19-5.

Note: In the current release, all users are granted READ privilege on all objects.

Post Installation

After you install your PL/SQL package in the repository, Warehouse Builder will provide the correct access control according to the security policy embedded in your implementation. For example, if a user who has no edit privilege on a project tries to edit the project, Warehouse Builder will stop the user and display an error message.

Security Requirements

Please keep in mind the following caveats regarding Warehouse Builder security:

- The database administrator must not grant `EXECUTE ALL PROCEDURES` to the public or any Warehouse Builder user.
- The default role in the Warehouse Builder user schema should NOT be set to `ALL`.
- Warehouse Builder users should not be granted any access to the central Warehouse Builder schema by the central schema owner or administrator.
- The implementation of the security service package must be correct.

Starting Warehouse Builder

To use Warehouse Builder you must either have access to an existing Warehouse Builder Repository or create a new one using the Repository Assistant. You need to have the connection information available when you open Warehouse Builder. For more information about creating a Warehouse Builder Repository, see the Oracle Warehouse Builder Installation and Configuration Guide.

For Windows NT/2000/XP Users

To start Warehouse Builder using Windows NT/2000/XP:

1. From the **Start** menu, select **Programs,** then **Oracle Database Developer Suite,** then **Warehouse Builder,** and then **OWB Client.**

   The Warehouse Builder logon dialog displays.

2. Click **Connection Info.**

   The Connect Information dialog displays.

3. Type the following connection information for the machine that contains the Warehouse Builder repository: Host Name of the database server, Port Number of the database instance, and Oracle Service Name.

   After you specify the connection information, it remains the same whenever you open Warehouse Builder. To log on to a different repository, you must open this dialog and edit the connection information.

4. Click **OK.**

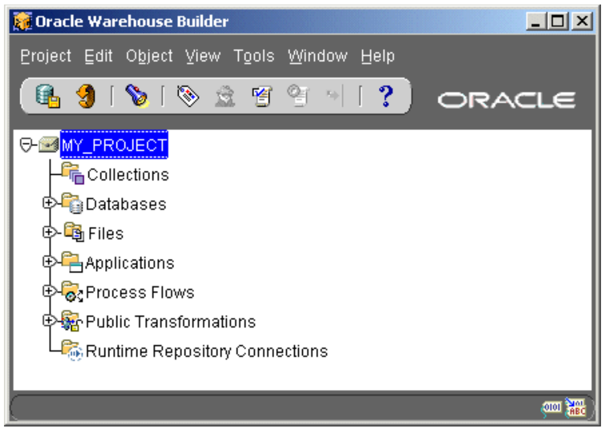
   The Warehouse Builder client saves the connection information and the logon dialog displays again.

5. Enter the User Name and Password of the Warehouse Builder repository and click **Logon.**

   If the Oracle8i/9i database instance is inactive, Warehouse Builder will display a connection error message.

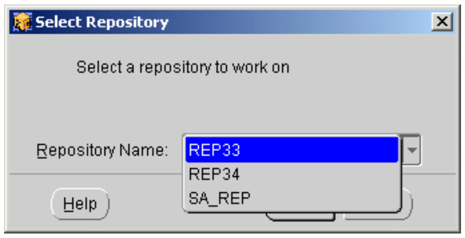
   The Warehouse Builder console opens and displays the contents of the repository in the navigation tree. If this is a newly created repository, the default project MY_ PROJECT displays. Figure 2–2 shows the Warehouse Builder console with the default project called MY_PROJECT.

Figure 2–2 Warehouse Builder Console



If the login user is not a Warehouse Builder repository owner, then Warehouse Builder displays the Select Repository dialog as shown in Figure 2–3. Choose the repository you want to access and click **OK.**

Figure 2–3 Select Repository Dialog



If you access a repository that is used by multiple users, you may want to synchronize periodically in order to update the console with what is stored in the repository. For more information, see "Supporting Multiple Users" on page 2-11.

For UNIX Users

To start Warehouse Builder using UNIX:

1. Start a shell.
2. Navigate (cd) to `<OWB ORACLE HOME>/owb/bin/unix`

   For example: `/private/home/OWB904/owb/bin/unix`

Table 2–1 UNIX

Warehouse Builder Component	Invoke
Warehouse Builder Client	owbclient.sh
Warehouse Builder Browser Assistant	browserasst.sh
Warehouse Builder MDL File Upgrade Utility	mdlconvertui.sh
Warehouse Builder Repository Assistant	reposasst.sh
Warehouse Builder Runtime Assistant	runtimeinst.sh
Warehouse Builder OMB Plus	OMBPlus.sh

Closing Warehouse Builder

To close Warehouse Builder:

1. From the **Project** menu, select **Exit.**

   The Commit Confirmation dialog displays if you have not already committed your changes.

2. Click **Yes** to commit any changes you have made. Click **No** to discard changes.

   Warehouse Builder closes and ends the session.

Committing Your Work

You can commit changes to the design repository at any time during a Warehouse Builder session. To commit changes during a session, go to the console window and click the Commit icon on the toolbar. Warehouse Builder displays the Commit Confirmation dialog. Click **Yes** to commit the changes.

You can also commit changes at the end of each session. To commit changes upon exiting Warehouse Builder, click **Yes** when the Commit Confirmation dialog displays.

Supporting Multiple Users

Multiple users can access the Warehouse Builder Repository at the same time. Warehouse Builder ensures that only one user has write privileges to an object and all other users have read-only access. Once a user has write access to any object, Warehouse Builder maintains a lock on the object for the duration of the transaction. Warehouse Builder releases the lock when the user commits the changes or performs a rollback and closes all editors associated with the object.

Note: If you validate, generate, or deploy code at any time during a Warehouse Builder session, multiuser locking remains in effect until you commit or rollback your changes.

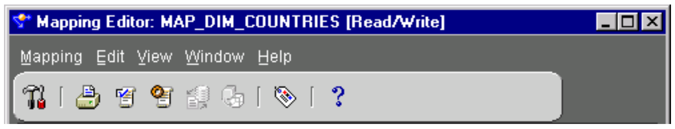
You lock an object when you open the editor, property sheet, or dialog for the object. You acquire a folder-in-use lock when you access an object in a Warehouse Builder hierarchy. This lock prevents other users from deleting a higher object in the hierarchy. For example, if you are editing an object in a module, other users cannot delete the module.

Read/Write Mode

To enter read/write mode for an object and lock the object, you open an editor, property sheet, or dialog. By default, you access objects in read/write mode.

Editors, property sheets, and dialogs indicate in the title bar that the object is in read/write mode, as shown in Figure 2–4.

Figure 2–4 Read/Write Indicator in the Title Bar



To end the read/write mode for an object and unlock the object for others, do one of the following:

- Close the object editors and commit or rollback the changes.
- Close or cancel out of the property sheet, editor or wizard you are using without making any changes.
- Exit Warehouse Builder.

Read-Only Mode

If you attempt to open an object locked by another user, Warehouse Builder displays a message that prompts you either to cancel the request or access the object in read-only mode.

If you choose to continue in read-only mode, the editor displays "Read only" in the title bar. You can move objects around on the editor canvas or expand and collapse the object icons. You cannot edit the object in read-only mode. If you try to edit an object in read-only mode, an error message dialog displays. On a property sheet opened in read-only mode, the **OK** button is disabled.

Synchronization

When a Warehouse Builder object is locked by a user, other users can view the changes by using synchronize only after the changes have been committed.

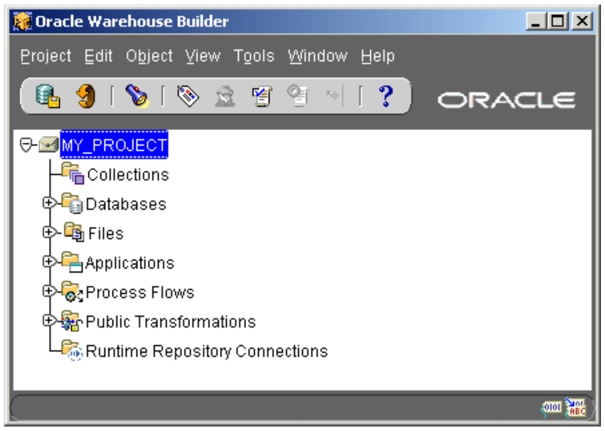
Warehouse Builder includes the following types of synchronization:

- **Automatic Synchronization:** Warehouse Builder automatically synchronizes objects when you access them in read/write mode.
- **Synchronization on command:** You can synchronize objects by selecting **Synchronize** from the **Project** menu or by pressing **F5.** This method displays changes to objects that do not show up with automatic synchronization. You can also use the synchronize button on the toolbar. This is useful for synchronizing the tree and synchronizing when you are in read-only mode.

Introducing the Console

After you log on to a repository in Warehouse Builder, the console displays the contents of the repository in the navigation tree, as shown in Figure 2–5. There are several components that make up the Warehouse Builder console.

Figure 2–5 Warehouse Builder Console



The main components of the Warehouse Builder console include:

- The Project Navigation Tree
- The Toolbar

The Project Navigation Tree

Warehouse Builder enables you to organize objects into projects within a navigation tree, as shown in Figure 2–6. The navigation tree is similar to a file system with all of the objects organized into expandable folders. If you have multiple projects in a repository, all projects initially display compacted. You can expand one project at a time. You must commit or rollback when switching between projects.

Figure 2–6 Navigation Tree

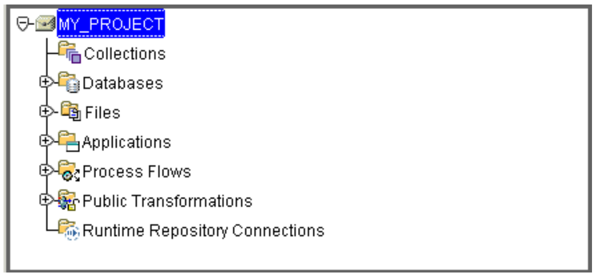


Table 2–2 describes the main folders in the navigation tree. You can expand these folders to view their contents and create objects.

Table 2–2 Navigation Tree Folders

Folder	Description
Collections	Groups of objects within a project. Use Collections to organize the contents of your project and to export metadata to other tools using the Warehouse Builder Transfer Wizard.
Databases	Contains all database objects. These objects are stored within modules. A modules contains either source or target database objects. At the highest level, databases are split into Oracle and non-Oracle databases.
Files	Contains file-based data objects stored within modules.

Table 2–2 (Cont.) Navigation Tree Folders

Folder	Description
Applications	Contain application-based data objects for packaged applications such as SAP. Objects are grouped into modules.
Process Flows	Contains process flows grouped in modules.
Public Transformations	Contains public and pre-defined transformation libraries.
Runtime Repository Connection	Contains runtime repository connection information.

The Toolbar

The toolbar is located along the top of the console under the menu bar. The toolbar provides shortcuts to frequently used Warehouse Builder tasks. Table 2–3 lists the shortcuts in the order they appear on the toolbar. Click the icon to run the task. All toolbar tasks can also be run from the menu bar. Table 2–7 shows the Warehouse Builder console toolbar.

Figure 2–7 Toolbar

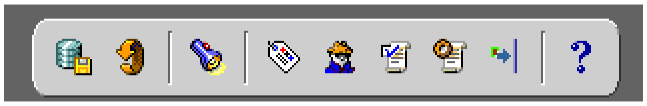


Table 2–3 shows the icons available on the Warehouse Builder console toolbar and a description of their functions.

Table 2–3 Toolbar Icons

Icon	Task	Description
	Commit	Commits changes to the database. You can select this icon at anytime.
	Synchronize	Synchronizes the objects displayed in the navigation tree with the objects in the repository. This is useful if you are working in a multiple user environment. It updates the objects displayed in the console with changes that other users have committed. You can select this icon at anytime.
	Find	Searches for an object within the currently expanded project. The Object Find dialog displays and you can type in the name or part of the name of an object. You can select this icon at anytime.
	Properties	Select an object in the navigation tree and click this icon to display the object property sheet. This icon is only available if the selected object has properties. It is greyed out if you select a folder or label.
	Configure	Select an object and click this icon. The configuration parameters dialog displays and can be used to define or edit parameters. This icon is only available if the selected object has configuration parameters. It is greyed out if you select a folder or label.
	Validate	Select an object or set of objects and click this icon. Warehouse Builder validates the object and displays the validation results. This icon is only available if the selected object can be validated.

Table 2–3 (Cont.) Toolbar Icons

Icon	Task	Description
⚙	Generate	Select an object or set of objects and click this icon. Warehouse Builder generates the object and displays the generation results.This icon is only available if the selected object can be generated.
➔|	Deploy	Select an object or set of objects and click this icon. Warehouse Builder prompts you to select a runtime repository connection for the deployment if you have not yet selected one. This icon is only available if the selected object can be deployed.
?	Help	Click this icon to view the Oracle Warehouse Builder User's Guide in the online help navigator. You can select this icon at anytime.

Setting Preferences

Warehouse Builder has a set of preferences that you can use to configure the client environment. To open the preferences, select **Project** from the menu bar and then **Preferences.**

Table 2–4 describes the types of preferences in the Preferences window. These are described in detail in the following sections.

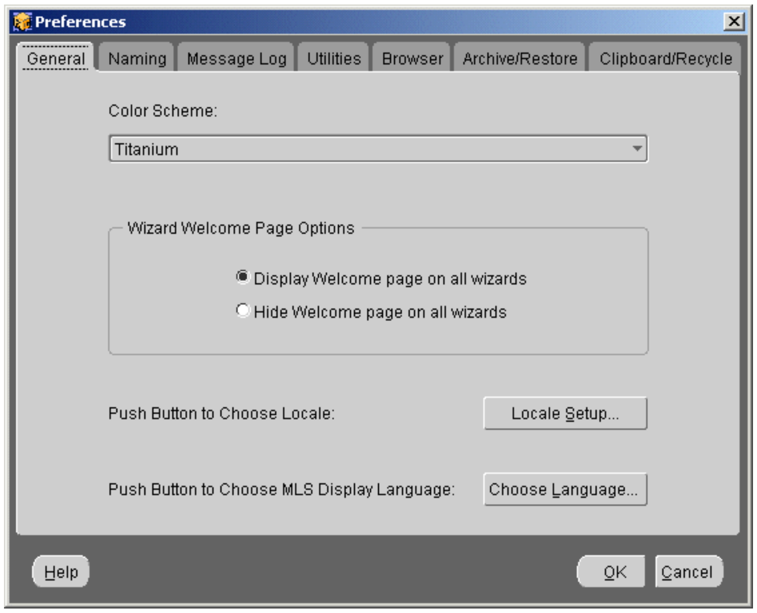
Table 2–4 Preferences

Preference Type	Description
General	Use to verify or define color and wizard welcome page preferences. Also use to define locale and MLS language.
Naming	Use to set naming preferences.
Message Log	Use 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.
Utilities	Use to provide information about the utilities you have selected for the Utility Tools menu.
Browser	Use to define the connection to the web server that is hosting the Warehouse Builder Design Browser. This enables you to access metadata reports from the within the client.
Clipboard/Recycle Bin	Use to set preferences for both the Recycle Bin and the Clipboard.

General Preferences

Use the General tab to define color scheme, define wizard welcome page preferences, set locale and choose display language.

Figure 2–8 General Preferences Tab



Color Scheme: Titanium

Wizard Welcome Page Options

● Display Welcome page on all wizards

○ Hide Welcome page on all wizards

Push Button to Choose Locale: Locale Setup...

Push Button to Choose MLS Display Language: Choose Language...

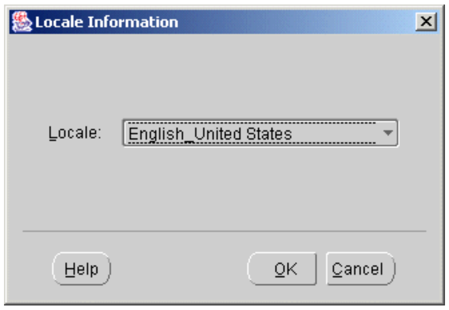
Use the drop-down menu to choose a color scheme for the Warehouse Builder console. The default color is Titanium.

Select the appropriate radio button to display or hide wizard welcome pages. Every wizard in Warehouse Builder starts with a welcome page that summarizes the steps you follow to complete that task.

- **Display Welcome Page on all Wizards:** Indicates you want the wizard welcome page to display when you start a wizard in Warehouse Builder.
- **Hide Welcome Page on all Wizards:** Indicates you want to skip the wizard welcome page when you start a wizard in Warehouse Builder.

Click **Locale Setup** and then, from the drop-down list, select the language you want the client text to display. This 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. Warehouse Builder prompts you to restart the computer in order to use the new language setting.

Figure 2–9 Locale Set-up

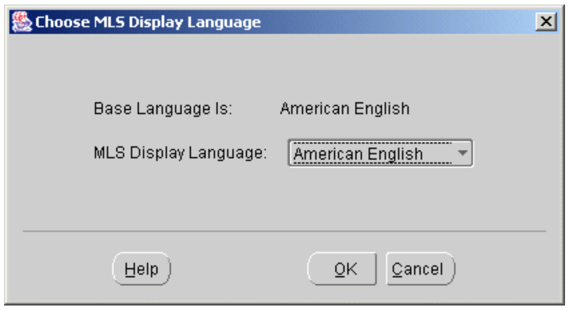


Locale: English_United States

Click **Choose Language** to change the language in which you edit the business name and description of objects created in Warehouse Builder. The business name is the name you assign an object when you create object using business naming mode. After you have changed the MLS display language, you can only change business names and descriptions for pre-existing objects. You must set the MLS language back to the base language in order to create new objects.

The supported languages must have already been defined during repository installation or upgrade in order to use this feature. For more information, see the Oracle Warehouse Builder Installation and Configuration Guide.

Figure 2–10 MLS Language Set-up



Base Language Is: American English

MLS Display Language: American English

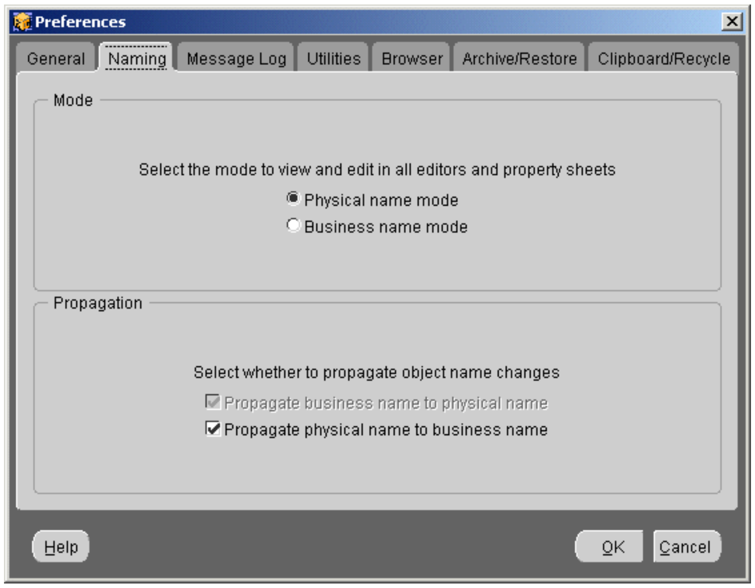
- **Base Language:** This is the language of your Warehouse Builder repository.
- **MLS Display Language:** Choose the language in which you want to edit or display the business name and description of an object in Warehouse Builder. The list of languages available to you are the languages you selected while creating or upgrading your repository.

For more information about MLS, see the Oracle Warehouse Builder Installation and Configuration Guide.

Naming Preferences

This page 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.

Figure 2–11 Naming Preferences Tab



Mode

Select the mode to view and edit in all editors and property sheets

● Physical name mode

○ Business name mode

Propagation

Select whether to propagate object name changes

☑ Propagate business name to physical name

☑ Propagate physical name to business name

About Naming Modes

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

When you generate DDL scripts for a named object, the physical names are used. Physical names must conform to the syntax rules for basic elements as defined in the Oracle Database SQL Reference.

Names must be unique within their category:

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

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

A business name must conform to these rules:

- The length of a name cannot exceed 4000 characters.
- The name must be unique within its category.
- All source modules reflect the case of the imported source and are subject to the double-quotes rules as defined in the Oracle Database SQL Reference.
- Copy operations from a source to a target in a mapping do not propagate case.

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

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

A physical name must:

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

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

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

Setting the Name Mode

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

The default naming preferences for Warehouse Builder are:

- **Mode:** Physical name mode.
- **Propagation:** Propagate physical name to business name.

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

To set the name mode:

1. From the **Naming** tab, select physical or business name mode.

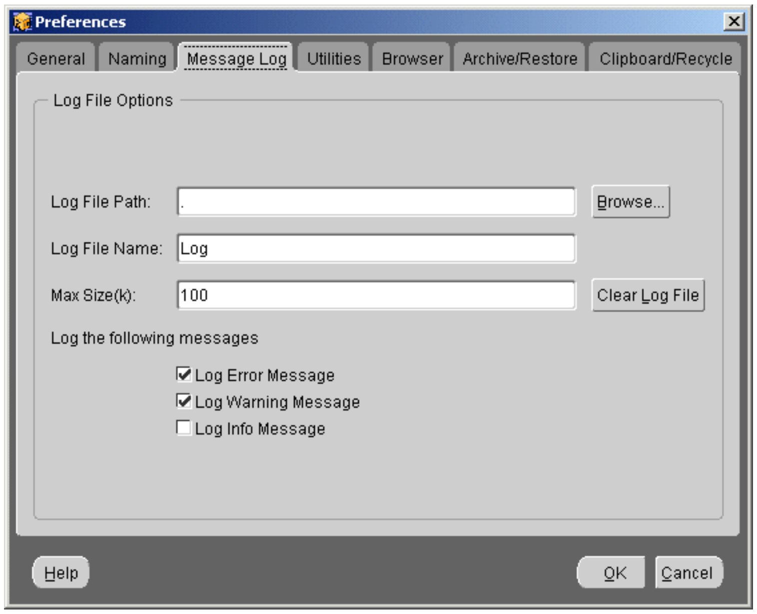
   You can switch the naming mode at any time during a session.
2. Select how the names propagate.
3. Click **OK.**

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.

Message Log Preferences

Use the Message Log tab to set Warehouse Builder log file options such as file location and name, file size, and types of messages saved to the log file. These files store Warehouse Builder design operations and errors. By default a message log is saved to the default location.

Figure 2–12 Message Log Preferences Tab



Log File Path: .

Log File Name: Log

Max Size(k): 100

Log the following messages

- [✓] Log Error Message
- [✓] Log Warning Message
- [ ] Log Info Message

Log File Path: Type the location or use the Browse button to select the location where you want to save the log files. The default location is `owbhome\owb\bin\admin`.

Log File Name: Type in the log file name. Do not include a file extension.

Max Size: Indicate the maximum file size for the log file(s). 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.

Clear Log File: Use this button to erase the contents of the log files.

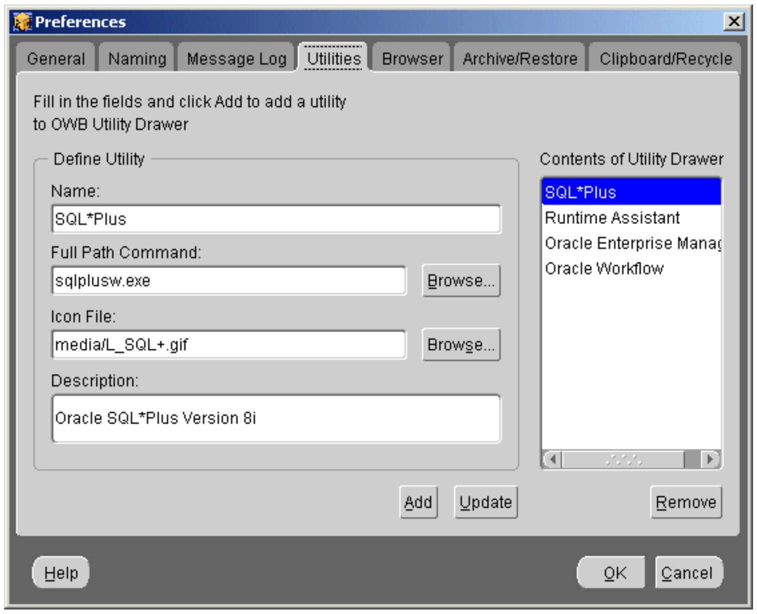
Select the following options to indicate which messages you want to capture:

- **Log Error Message:** Writes all error messages to the log file.
- **Log Warning Message:** Writes all warning messages to the log file.
- **Log Info Message:** Writes all informational messages to the log file.

Utilities Preferences

Use the Utilities tab to add, update, or remove utilities. The utilities are available from the Tools menu.

Figure 2–13 Utilities Preferences Tab



Fill in the fields and click Add to add a utility to OWB Utility Drawer

Define Utility

Name: SQL*Plus

Full Path Command: sqlplusw.exe

Icon File: media/L_SQL+.gif

Description: Oracle SQL*Plus Version 8i

Contents of Utility Drawer

SQL*Plus
Runtime Assistant
Oracle Enterprise Manag
Oracle Workflow

Adding a Utility

To add a utility to the Tools menu:

1. Select the Utility tab and replace the fields with the following information:

   Name of the utility.

   Location of the utility. Click **Browse** to search for program folders and files.

   Location of the icon for the utility. Click **Browse** to search for icon folders and files.

   Description of the utility.

2. Click **Add.**
3. Click **OK.**

Updating a Utility

To update the configuration of a utility:

1. From the Utilities sheet, select the utility name from the Contents list.

   Warehouse Builder displays the configuration information.

2. Modify the following configuration details:

   Name of the utility.

   Location of the utility. Click **Browse** to search for program folders and files.

   Location of the icon for the utility. Click **Browse** to search for icon folders and files.

   Description of the utility.

3. Click **Update.**
4. Click **OK.**

Removing a Utility

To remove a utility from the Tools menu:

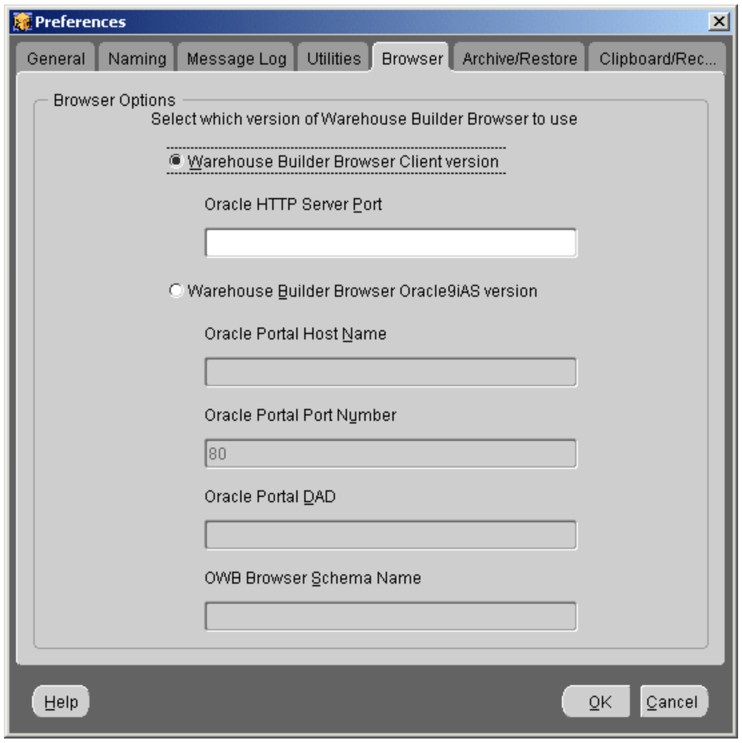
1. From the Utilities sheet, select the utility name from the Contents list.
2. Click **Remove.**
3. Click **OK.**

   Warehouse Builder removes the utility from the Tools menu.

Browser Preferences

Use the Browser tab to select which version of the Warehouse Builder Browser you want to use when you invoke metadata reports from within the client console. You can choose the Client version, or the Oracle9iAS version.

Figure 2–14 Browser Preferences Tab



Browser Options
Select which version of Warehouse Builder Browser to use

● Warehouse Builder Browser Client version

Oracle HTTP Server Port

○ Warehouse Builder Browser Oracle9iAS version

Oracle Portal Host Name

Oracle Portal Port Number: 80

Oracle Portal DAD

OWB Browser Schema Name

Using the Client Version

If you choose the Warehouse Builder Browser Client version, you must provide the Oracle HTTP Server Port. This requires that you run the HTTP Server on the same machine as the Warehouse Builder Repository that you are accessing. Contact your DBA to obtain your Oracle HTTP Server Port:

Using the Oracle9*i*AS Version

If you choose to use the Oracle Portal version, you must establish a connection with Oracle Portal which is installed with Oracle9*i*AS. Enter the following connection details to use this option:

Portal Host Name: Name of the system where you installed Oracle Portal.

Portal Port Number: Default:7778.

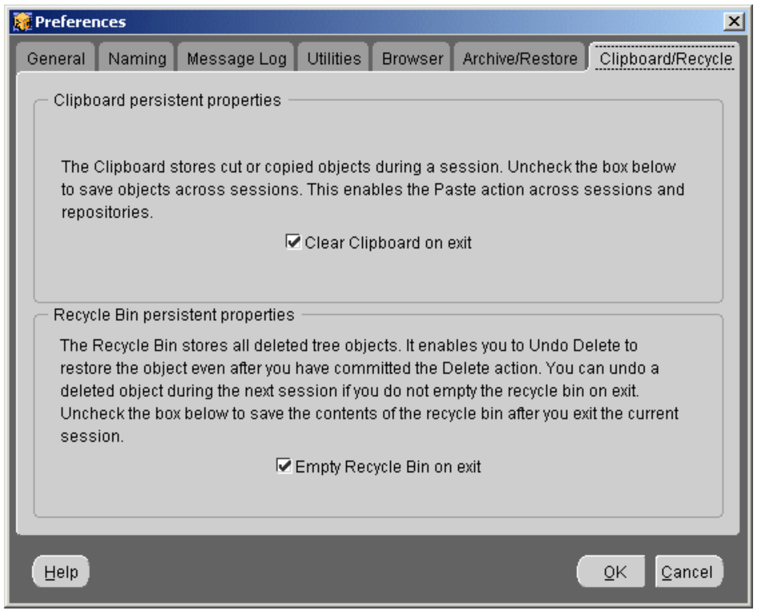
Portal DAD: Portal Database Access Descriptor chosen during Oracle Portal installation.

Browser Schema Name: The username provided during the Warehouse Builder Browser installation using the Browser Assistant.

Clipboard/Recycle Preferences

Use the Clipboard/Recycle tab to determine when objects are removed from the Clipboard and Recycle Bin.

Figure 2–15 Clipboard/Recycle Preferences Tab



Clipboard persistent properties

The Clipboard stores cut or copied objects during a session. Uncheck the box below to save objects across sessions. This enables the Paste action across sessions and repositories.

[✓] Clear Clipboard on exit

Recycle Bin persistent properties

The Recycle Bin stores all deleted tree objects. It enables you to Undo Delete to restore the object even after you have committed the Delete action. You can undo a deleted object during the next session if you do not empty the recycle bin on exit. Uncheck the box below to save the contents of the recycle bin after you exit the current session.

[✓] Empty Recycle Bin on exit

Clipboard Persistence Properties

To preserve the objects in the Clipboard across sessions, you must clear the **Clear Clipboard on exit** box.

The Clipboard stores the object you last cut or copied from the Warehouse Builder navigation tree. The Clipboard is stored on the Warehouse Builder client. Set this preference to indicate whether you want objects stored for use only during the current Warehouse Builder session or across all Warehouse Builder sessions using this client machine. The default setting is to empty the Clipboard each time you log out of a session.

Recycle Bin Persistence Properties

To preserve the objects in the Recycle Bin across sessions, you must clear the **Empty Recycle Bin on exit** box.

The Recycle Bin stores all the objects you delete from the Warehouse Builder navigation tree. The Recycle Bin is stored on the Warehouse Builder client. Set this preference to indicate whether you want objects stored for use only during the current Warehouse Builder session or across all Warehouse Builder sessions using this client machine. The default setting is to empty the Recycle Bin each time you log out of a session.

If you choose to persist the deleted objects in the Recycle Bin, you can restore them to the repository during another session. Commit and Rollback actions do not affect objects in the Recycle Bin.

Creating and Editing Warehouse Builder Objects

When you begin using Warehouse Builder, begin by creating a Project. After you create a project, you can create all the other Warehouse Builder objects. Warehouse Builder contains wizards, object editors, property sheets, and object finding tools that assist you use in designing your business intelligence system. Use these components to aide you in the design process.

Creating Projects

When you install a Warehouse Builder design repository, Warehouse Builder creates a default project called MY_PROJECT with a set of pre-defined public transformations. The default project is required for the initial logon sequence. You can delete MY_PROJECT after you create other projects. For information about installing a Warehouse Builder design repository, see the Oracle Warehouse Builder Installation and Configuration Guide.

Creating a New Project

When you begin using Warehouse Builder, create new projects to manage your design work.

To create a new project:

1. From the Project menu, select **Create Project.**

   The welcome page for the New Project Wizard displays.

2. Click **Next.**

   The Name page displays.

3. Enter the project name and an optional description, and click **Next.**

   The Version Properties page displays.

4. Enter the optional version label and then click **Finish.**

   The new project is created and added to the bottom of the navigation tree and Warehouse Builder does an implicit commit and saves the new project in the design repository.

   You can now use wizards to create objects within the project. Expand the project and select an object type to begin.

Deleting Projects

Deleting a project is not as simple as deleting any other object in Warehouse Builder. Since projects are the main design component in Warehouse Builder, this feature is designed to protect them from being deleted unintentionally.

The following guidelines apply:

- The currently active or expanded project cannot be deleted.
- The last remaining project in a repository cannot be deleted.
- Each design repository must contain at least one project.

To delete a project:

1. Select and expand any project in the navigation tree other than the project you want to delete.

   The project you expand becomes the current active project.

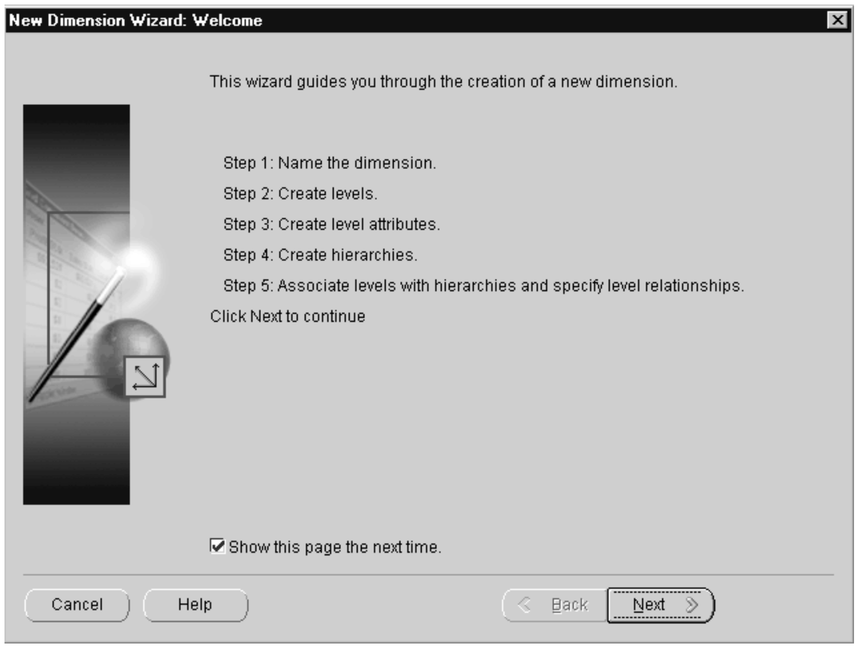
2. Select, but do not expand the project you want to delete.
3. From the **Edit** menu, select **Delete.** You can also right-click the project and select **Delete.**

   The Deletion Confirmation dialog displays. You can choose to delete the object, or put the object in the recycle bin.Warehouse Builder does not implicitly commit the deletion of a project as it does with the creation of a new project.

Using the Warehouse Builder Wizards

Warehouse Builder provides wizards for creating all objects. To open a wizard, right-click an object type and select **Create.** The first page of each wizard is the welcome page, as shown in Table 2–16. This page lists the number of pages in the wizard and describes the task performed on each page.

Figure 2–16 Wizard Welcome Page


New Dimension Wizard: Welcome

This wizard guides you through the creation of a new dimension.

Step 1: Name the dimension.

Step 2: Create levels.

Step 3: Create level attributes.

Step 4: Create hierarchies.

Step 5: Associate levels with hierarchies and specify level relationships.

Click Next to continue

☑ Show this page the next time.

You can configure wizards to skip the welcome pages by clearing the **Show this page the next time** check box. To re-enable the welcome page, open the Preferences dialog from the Project menu and check the **Display Welcome page on all wizards** box.

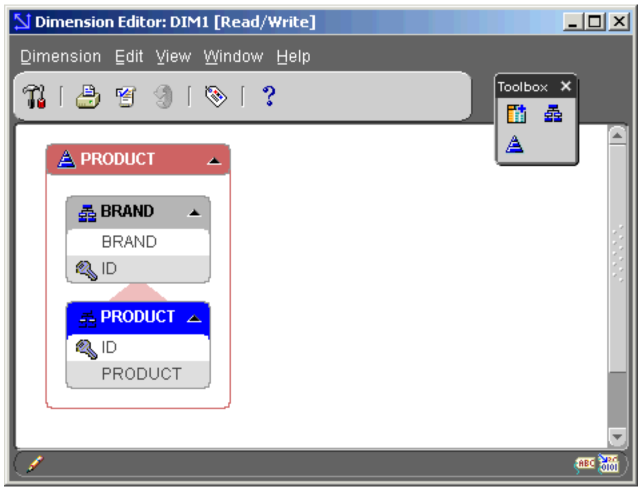
To navigate through the pages of a wizard, click the **Next** and **Back** buttons. Click **Cancel** to exit without saving your work. You can move around a page using the mouse or using the Tab key. Use Control-Tab to move the cursor out of a long description text box.

For help on a specific topic in a wizard, click **Help** at the bottom of the wizard page.

Using Object Editors

After you create modules and objects, you can edit them using editors and property sheets. Object editors provide a separate window with menu items and a toolbox to edit the object. To display an editor, right-click the object and select **Editor.** Figure 2–17 shows an example of a Dimension Editor.

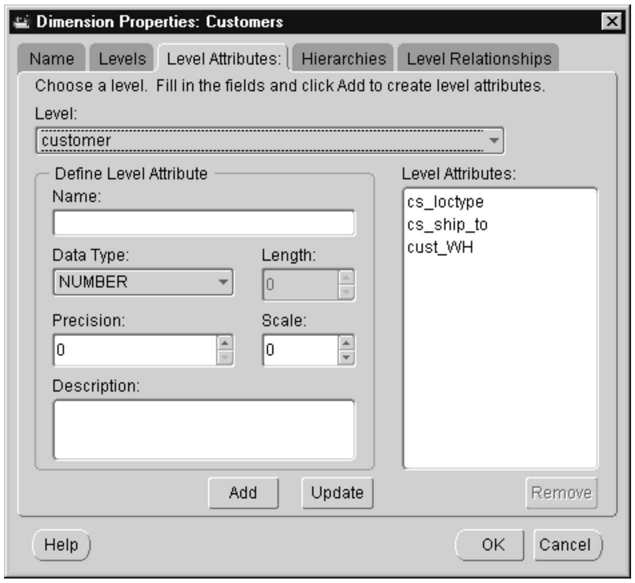
Figure 2–17 Dimension Editor


Dimension Editor: DIM1 [Read/Write]
Dimension Edit View Window Help
Toolbox
PRODUCT
BRAND
BRAND
ID
PRODUCT
ID
PRODUCT

Using Property Sheets

Selecting the Property icon from the console toolbar displays the property sheet for the selected object. Each object stored in a project has a property sheet that defines the object. You can update the stored definition of an object by editing its property sheet. Property sheets vary by object. Figure 2–18 shows an example of a property sheet for a dimension.

Figure 2–18 Property Sheet for a Dimension



Choose a level. Fill in the fields and click Add to create level attributes.

Level: customer

Define Level Attribute

Name:

Data Type: NUMBER

Length: 0

Precision: 0

Scale: 0

Description:

Level Attributes:

cs_loctype
cs_ship_to
cust_WH

Creating User-Defined Properties

To enhance the metadata for your unique needs, Warehouse Builder enables you to create additional properties for objects. You can create user-defined properties for any object type on the navigation tree. To create user-defined properties, use the Oracle MetaBase (OMB) Scripting Language, OMB Plus. This is the scripting utility packaged with Warehouse Builder. Create user-defined properties to store additional business, design, or versioning information for objects.

When you create user-defined properties, you must prefix them with `UDP_`. This naming convention enables you and Warehouse Builder to distinguish core properties from customized ones. If you try to create a property without following the naming convention, an error message directs you to use the `UDP_` prefix. You can view and populate the properties you created either in the user interface Properties sheets for the corresponding objects or directly in OMB Plus.

For instance, you can add a property called `UDP_BUSINESS_PURPOSE` to table objects so that you can describe the business purpose of each table. The property then appears in the user interface as "UDP Business Purpose" when you open the Properties sheet for any table. You can populate the property with a description of the business purpose either in the user interface Properties sheets or directly in OMB Plus.

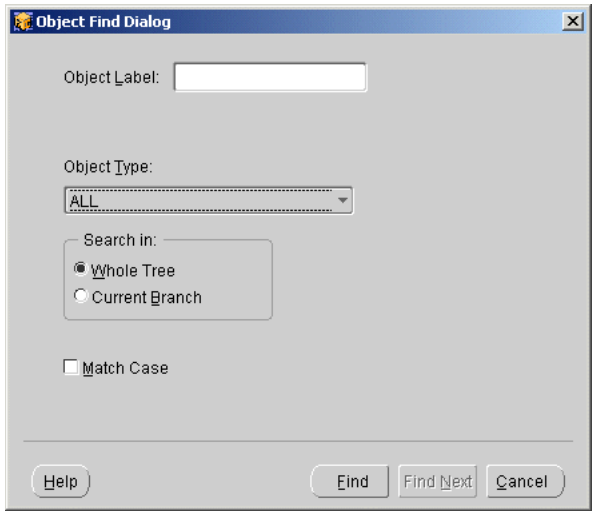
Tip: Try to finalize any user-defined properties you plan to create before defining data objects. This enables you to populate the user-defined properties as you go along instead of having to retroactively edit objects later in the design process.

For the scripting commands and arguments related to user-defined properties, consult the Oracle Warehouse Builder Scripting Reference .

Finding Objects in a Project

Objects stored in a Warehouse Builder repository are organized in a navigation tree. To find an object in the repository, you can click **Find** and type the name or part of the name. Use the asterisk (*) as a wildcard character to match one or more characters. Figure 2–19 displays the Object Find dialog.

Figure 2–19 Object Find Dialog



Managing Metadata

Warehouse Builder stores all metadata for the target system you design in the Warehouse Builder design repository. Before you begin using Warehouse Builder you must already have an installed repository. When you open Warehouse Builder, the navigation tree displays the contents of the design repository. To manage your metadata, Warehouse Builder enables you to run reports on the metadata including lineage and impact analysis, export metadata from one repository and import it into another, and exchange metadata between different tools.

Metadata Reporting

Access reports on all of the metadata stored in your design repository using the Warehouse Builder Design Browser. You can access metadata reports from within the Warehouse Builder client, or through Oracle Portal without even installing the client. For more information, see Chapter 17, "Metadata Browsing and Reporting".

Metadata Import and Export

Use the Metadata Loader (MDL) to populate a new repository as well as transfer, update, or restore a backup of existing repository metadata. You can import and export metadata for any type of object on the navigation tree using the MDL utility. Access MDL through the Warehouse Builder client or through the OMB Plus scripting interface. For more information, see "Importing and Exporting Metadata Using the Metadata Loader" on page 15-1.

Metadata Exchange

Create collections and use the Warehouse Builder Transfer Wizard to exchange metadata stored in the Warehouse Builder design repository.

Defining Collections

Collections are groups of objects within projects that you can define to organize objects in Warehouse Builder. When you create a collection, you create shortcuts pointing to objects already existing in the project. These shortcuts provide quick access to the base object and allow you to make fast changes to it.

Using the Warehouse Builder Transfer Wizard

Use the Warehouse Builder Transfer Wizard to synchronize, integrate, and use metadata stored in the design repository in a variety of business intelligence tools. You can import and export metadata using a bridge. You can also exchange metadata with OMG files, Oracle Discoverer, Oracle Express, ERwin, Predesignate, and Oracle Database OLAP Server. For instructions for using the Transfer Wizard, and for additional information on this topic, see Chapter 22, "Integrating Warehouse Builder Metadata with Other BI Products".

Part I

Defining Source and Target Objects

This part contains the following chapters:

3 Defining Oracle Data Objects

After you finish gathering the requirements for your data warehouse or data mart, you are ready to design your target system using Warehouse Builder. Most of your target schema modelling takes place within the Oracle warehouse module. This chapter shows you how to create an Oracle warehouse module and define the data objects within that module.

This chapter contains the following topics:

Creating Warehouse Modules

Warehouse Builder stores the definitions for your target schema in warehouse modules. These definitions can be created using the Warehouse Builder wizards or by importing them from external sources. This section shows you how to create a warehouse module. The following sections show you how to create and import definitions for data objects in the warehouse module.

To create a warehouse module:

1. From the Warehouse Builder navigation tree, expand the Databases node.
2. To create a source module for Oracle data sources, right-click the Oracle node and select **Create Oracle Module.**
3. At the welcome page for the New Module Wizard, click **Next.**

   The wizard displays the Name page.
4. Specify the following information:

Name of the module.

The status of the module: Development, Quality Assurance, or Production. This is for descriptive purposes only.

Warehouse Target as the Module Type.

An optional description.

Click **Next.**

5. At the Connection Information page, you have the option to provide the database link information necessary to import metadata into the warehouse module. You can choose to skip this page and provide this information when you import the metadata.

   First select your metadata source:

   Oracle Data Dictionary: Select to import metadata from an Oracle database.

   Oracle Designer Repository: Select to import metadata from an Oracle Designer repository.

   From the Database Link field, select from a list of previously created database links. Or create a new database link by provide the following information:

   Owner: The source database user who is creating this database link.

   Username: Name of a user with access to the source database using the database link.

   Connect String: Name of the system where the source database resides.

   Schema: Name of the schema where the source database resides.

   Click **Next.**

6. At the Location page, you have the option to select a location from the drop-down menu or click **New** to create a new location.

   Locations define information about the database schema or target tool where you will be deploying objects. Locations are specific to a type of module such as Oracle Database, SAP, or flat file. For more information, see "Defining Locations" on page 3-2.

   This step is optional. You can choose to create a location for this module later when you are deploying the object.

   Click **Next.**

7. At the Finish page, which summarizes the information you provided on each of the wizard pages, check the checkbox if you want to directly start the Import Metadata Wizard.

   Click **Finish.**

   The wizard creates the warehouse module and inserts its name in the project navigation tree. If you checked the check box, Warehouse Builder starts the Import Metadata Wizard. See "Importing Metadata into Target Modules" on page 3-50.

Defining Locations

Locations represent specific database schemas and target tools. They are specific to types of modules such as an Oracle or non-Oracle Database, SAP, or file system and are organized into the navigation tree under these modules. When you create a location, a logical definition containing location type and version is stored. When the location is registered, the physical connection information is requested and stored in the Runtime Repository.

Each location defined within a project can be registered separately within each Runtime Repository, and each registration can reference different physical information. Using this approach, you can design and configure a target system one time, and deploy it many times with different physical characteristics. This is useful if you need to create multiple versions of the same system such as development, test, and production.

When you create modules, you must specify a location for each module. You can assign the same location to multiple modules, but you must create a location for each distinct database schema or tool to which you want to deploy. For Oracle database modules, you must also define connectors to any modules it is referencing. These connectors, when deployed, may generate database links as necessary for designed data movement. Within an Oracle database module, a connector referencing a file system can also be created. This kind of connector will be generated as a database directory, which are used by external tables.

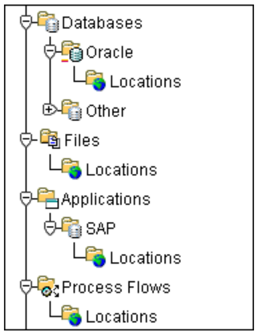
Creating Locations

To create a new location:

1. Select a project, and expand the navigation tree to display Locations.

   There are four main types of locations: database, file system, applications, and process flow, as shown in Figure 3–1.

Figure 3–1 Locations on the Navigation Tree



2. Select a Locations node from under the type of module you want to create a location.
3. From the Object menu, select Create Location or right-click Locations and select **Create Location**.
4. At the New Location Wizard Welcome Page, click **Next** to continue.
5. In the Name page, define the following information for the location:

   Name: Provide a name for the location. Maximum Length: 30 Characters.

   Optional Description: Provide an optional description for the location. Maximum Length: 400 Characters.

   There are no pre-assigned default values for a new location.

   Click **Next** to continue.
6. In the Details Page, define the following information for the location:

Location Type: Select a type from the drop-down list for the location you are creating. The default is determined by the type of module for which the location is being created.

Version: Select a version for the type of location you are creating from the drop-down list.

Click **Next** to continue.

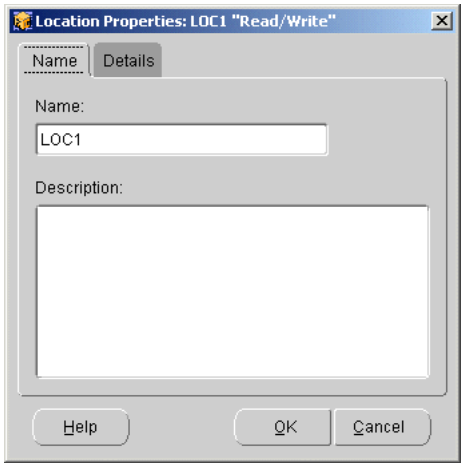
7. At the Finish Page, verify the definition of the new location. This page lists the name, type, and version.

   Click **Finish** to create the location as defined. The location is created and added under the appropriate Locations node.

Editing Locations

To edit a location, right-click the location from the navigation tree and select **Properties.** Warehouse Builder displays the properties window, as shown in Figure 3–2.

Figure 3–2 Location Name Tab



Name:
LOC1
Description:

This page displays the properties of the selected location. Use the following two tabs to view and edit the properties. Click **OK** to save changes or **Cancel** to close the window.

Name Tab

The name tab display the following information:

- **Name:** Displays the location name. Maximum Length: 30 Characters.
- **Description:** Displays an optional description for the location. Maximum Length: 4000 Characters.

Details Tab

The details tab displays the following information:

- **Location Type:** Displays the type of location you are creating. The default is determined by the type of module for which the location is being created.
- **Version:** Displays the version for the type of location you are creating.

Defining Connectors

Connectors define connections between Oracle database module locations and other defined module locations in the navigation tree. They are only located under the Oracle database Locations node in the navigation tree. Connectors indicate that there is a path to transport data from one location to another location.Connectors are owned by their containing Oracle database module, and they reference one other location.

When you create a connector in the navigation tree, a logical definition is stored in the Warehouse Builder repository. When objects in a location with a defined connector are deployed, a database link or directory object may be referenced if necessary. Only one connector can be defined in each direction between any two specific locations. Connectors are also created implicitly in the Mapping Editor as sources and targets are placed on the canvas. As you place sources on the canvas, a connector is created automatically between the mapping location and the source location if one does not exist already. This connector indicates that there is a path to transport data from the source location to the mapping location.If you use a flat file as a source in a mapping, no connector is created automatically; you must define a connector from the flat file source to the relational target yourself.

Note: Connectors that reference a file system are realized as database directories. To create and drop directories, you must be granted the "create any directory" and "drop any directory" privileges. These are high privileges and are not granted behind the scenes.

Creating Connectors

To create a connector:

1. Select a project and expand the navigation tree to display all Oracle database modules.
2. Expand the Locations node.
3. Right-click the location for which you want to create a connector.
4. From the Object menu select **Create Connector** or right-click the location and select **Create Connector.**
5. At the New Connector Wizard Welcome Page, click **Next** to continue.
6. In the Name Page, define the following information for the connector:

   Name: Provide a name for the connector. Maximum Length: 30 Characters.

   Optional Description: Provide an optional description for the connector. Maximum Length: 400 Characters.

   There are no pre-assigned default values for a new connector.

   Click **Next** to continue.
7. In the Details Page, specify the referenced location for the connector:

   Database: Select this option to link to a database location. Then select the specific location name from the drop-down list.

   File System: Select this option to link to a file location. Then select the specific location name from the drop-down list.

   Application: Select this option to link to a application location. Then select the specific location name from the drop-down list.

Unspecified: Select this option to link to an unspecified type of location.

Click **Next** to continue.

8. In the Finish Page, verify the definition of the new connector. This page lists the name and referenced location.

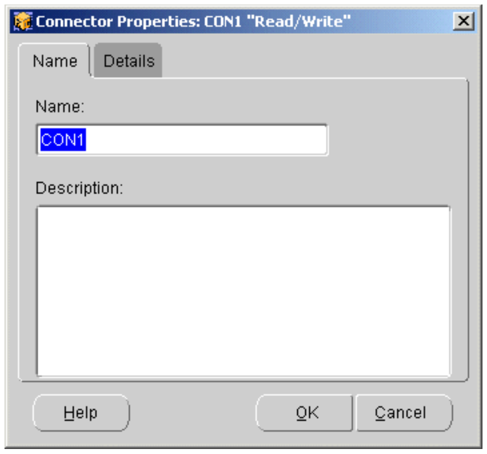
   Click **Finish** to create the connector as defined.

   The connector is created and added under the location. Continue creating connectors for each location as necessary.

Editing Connectors

To edit a connector, right-click the connector from the navigation tree and select **Properties.** Warehouse Builder displays the properties window, as shown in Figure 3–3.

Figure 3–3 Connector Name Tab


Connector Properties: CON1 "Read/Write"

Name: CON1

Description:

This page displays the properties of the selected connector. Use the following two tabs to view and edit the properties. Click **OK** to save changes or **Cancel** to close the window.

Name Tab

- **Name:** Displays the location name. Maximum Length: 30 Characters.
- **Description:** Displays an optional description for the location. Maximum Length: 400 Characters.

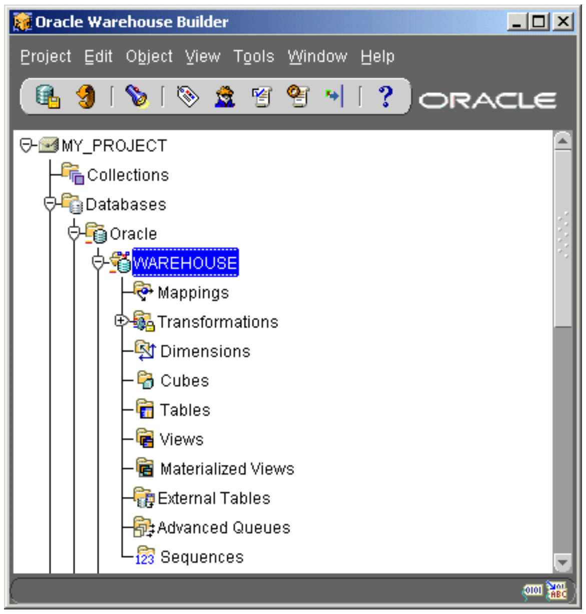
Details Tab

- **Database:** Select this option to link to a database location. Then select the specific location name from the drop-down list.
- **File System:** Select this option to link to a file location. Then select the specific location name from the drop-down list.
- **Application:** Select this option to link to a application location. Then select the specific location name from the drop-down list.
- **Unspecified:** Select this option to link to an unspecified type of location.

About Data Objects

After you create a warehouse module, you can locate it by expanding the Databases node, then the Oracle node in the navigation tree. Now expand the warehouse module to view all the data objects supported by Warehouse Builder, as shown in Figure 3–4.

Figure 3–4 Data Objects within a Warehouse Module



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

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

Table 3–1 describes the types of data objects you can use in Warehouse Builder.

Table 3–1 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 and triggers can also be defined for a table. See Using Tables on page 3-8.
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 3-18.

Table 3–1 (Cont.) Data Objects in Warehouse Builder

Data Object	Type	Description
Views	Relational	A view is a custom-tailored presentation of data in one or more tables. Views do not actually contain or store data; they derive their data from the tables on which they are based. Like tables, views can be queried, updated, inserted into, and deleted from, with some restrictions. All operations performed on a view affect the base tables of the view. Use views to simplify the presentation of data or to restrict access to data. See Using Views on page 3-25.
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 3-28.
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 3-31.
Advanced Queues	Relational	Advanced Queues enable message management and communication required for application integration. See Using Advanced Queues on page 3-33.
Dimensions	Dimensional	A general term for any characteristic that is used to specify the members of a data set. The 3 most common dimensions in sales-oriented data warehouses are time, geography, and product. Most dimensions have hierarchies. See Using Dimensions on page 3-39.
Cubes	Dimensional	Cubes contain measures and links to one or more dimension tables. They are also known as Facts. See Using Cubes on page 3-47.

Using Tables

In Warehouse Builder, tables are metadata representations of relational storage objects. They can be tables from a database system such as Oracle tables or even tables from an SAP system. The following sections provide information about creating and using tables in Warehouse Builder:

- Creating a Table Definition on page 3-8
- Using the Table Editor on page 3-10
- Editing Table Definitions on page 3-11

Creating a Table Definition

The table you create in Warehouse Builder captures the metadata used to model your target schema. This table definition specifies the table constraints, indexes, and metadata about the columns and data types used in the table. This information is stored in the Warehouse Builder repository. You can later use these definitions to generate `.ddl` scripts in Warehouse Builder that can be deployed to create physical tables in your target database. These tables can then be loaded with data from chosen source tables.

Use the Table Wizard to create a table definition. This section describes the main pages of the New Table Wizard.

Note: You can also create a table from the Mapping Editor.

Creating a Table with the New Table Wizard

To create a table:

1. From the Warehouse Builder navigation tree, 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 **Create Table.**

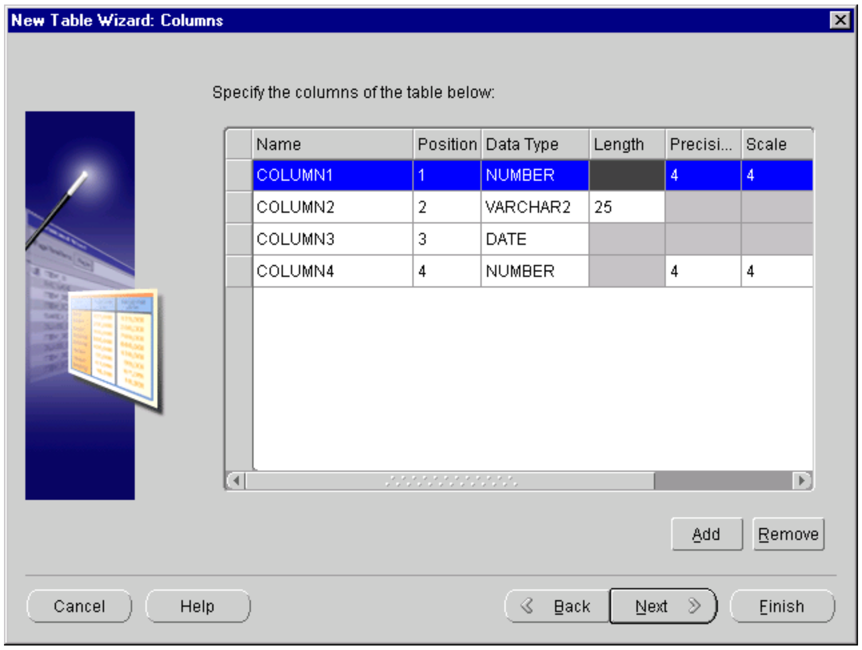
   The welcome page for the New Table Wizard displays the steps involved in creating a table.
4. Click **Next** to continue to name the table.
5. Type a name for the table.

   In physical naming mode, you must type a unique name between 1 and 28 valid characters. Spaces are not allowed. In logical mode, you can type a name up to 4000 characters long. Spaces are allowed. The name must be unique within the module. For more information, see "Naming Preferences" on page 2-17.

   After you name the table, you can continue using the wizard to define the table properties, or you can click Finish to create the table and set-up or edit these properties later using the Table Editor.
6. Type in a description for the table.

   The description can be a maximum of 4000 characters. This is optional.
7. Click **Next** to continue and define the columns, as shown in Figure 3–5.

Figure 3–5 Table Wizard Columns Page



Specify the columns of the table below:

Name	Position	Data Type	Length	Precisi...	Scale
COLUMN1	1	NUMBER		4	4
COLUMN2	2	VARCHAR2	25		
COLUMN3	3	DATE			
COLUMN4	4	NUMBER		4	4

8. Click **Add** to add a column.

Type in a column name. 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 3-14.

9. Select a data type for the column from the drop-down list under Data Type. For details on data types, see"Adding Columns" on page 3-12. Warehouse Builder supports the following Oracle Database data types:

   CHAR

   DATE

   FLOAT

   NUMBER

   VARCHAR

   VARCHAR2

10. Type the precision, length, or scale as appropriate.

    Warehouse Builder enables you to enter values relevant to the data type you select. For example, for CHAR data types you must specify length. For NUMBER data types you must specify precision and scale.

11. Specify NULL or NOT NULL. By default, all columns in a table allow nulls. A NOT NULL constraint prohibits a column of a table from containing null values.
12. Type a description of the column in the Description field (optional).
13. Repeat these steps 8 through 13 for each column.
14. Click **Next** to continue and define constraints.
15. In the Constraints page, you have the option to define constraints if required. For more information on adding constraints, see "Editing Constraints" on page 3-15.
16. Click **Next.**

    Warehouse Builder displays all of the table properties defined in the New Table Wizard. Verify the name, descriptions and various table properties.

17. Click **Finish.**

    The New Table Wizard creates and stores a definition for the table in the repository and inserts the new table name in the navigation tree.

Using the Table Editor

After you create a table in Warehouse Builder, you can use the Table Editor to view the table structure, table columns, and any related tables. To open the Table Editor, right-click the name of a table and select **Editor** the name of the table. Figure 3–6 shows the Table Editor.

Figure 3–6 Table Editor

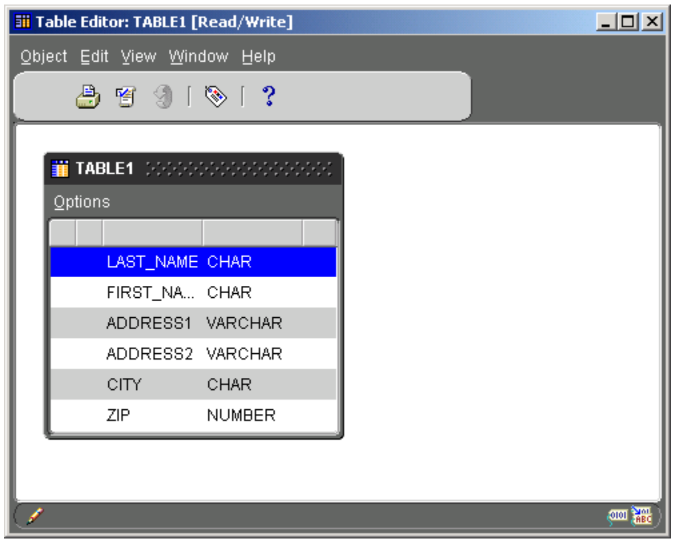


Table Editor: TABLE1 [Read/Write]
Object Edit View Window Help
TABLE1
Options

LAST_NAME	CHAR
FIRST_NA...	CHAR
ADDRESS1	VARCHAR
ADDRESS2	VARCHAR
CITY	CHAR
ZIP	NUMBER

Use the Table Editor Menu, the Table Editor Navigation bar, or the right-click pop up options to edit properties, print the table diagram, validate table definitions, synchronize the table definitions, and invoke Warehouse Builder Browser to run reports on the selected table.

Displaying Related Tables

When you open the Table Editor, it displays a diagram of the selected table and the columns defined for that table. The columns in this table may reference columns in another table through foreign key relationships. You can display those referenced or related tables by selecting **View,** then **Show Related Objects.** The Table Editor displays the related tables and the linking lines show the relationships. You can rearrange the displayed tables by selecting a table and moving it around the canvas.

Validating Table Definitions

Before you generate DDL scripts to create a table, you can validate your table definitions from the Table Editor. Select **Object,** then **Validate.** Warehouse Builder displays the Validation Results dialog indicating whether your definitions were valid or not. If your definitions are valid, you can generate scripts to create that object in the your target database. For details on validating definitions and code generation, refer to Chapter 12, "Validating Objects".

Viewing Reports

To view metadata reports on any table, select **View,** then **Reports.** You can also view Lineage and Impact Analysis reports on a table to show the lineage for the data in the table or the impact of any changes made to the table. To view one of these reports, select **View,** then **Lineage** or **Impact Analysis.** For information on installing and configuring the Warehouse Builder Browser, refer to Chapter 17, "Metadata Browsing and Reporting".

Editing Table Definitions

From the Table Editor, you can access the Table Properties window to edit the name, description, columns, constraints, and attribute sets in a table.

To edit table properties, select **Object** then **Properties** from the Table Editor menu or right-click the name of a table from the navigation tree and select **Properties.** The Table properties window displays four tabs: Name, Columns, Constraints, and Attribute Sets.

Click these tabs to perform the following tasks:

- Rename a Table
- Add and Remove Columns
- Reorder Columns
- Add and Remove Constraints
- Edit Constraints
- Delete Constraints
- Add Attribute Sets

Renaming a Table

You can rename a table without editing its description by right-clicking the name of the table on the Warehouse Builder navigation tree and selecting **Rename.**

To edit the name and description of a table:

1. Right-click the name of a table and select **Properties.**
2. In the Table Properties window, select the Name tab and edit the following properties.

   Name: Type a new name for the table. In physical naming mode, you must type a unique name containing between 1 and 30 valid characters. Spaces are not allowed. In logical naming mode, you can type a name up to 200 characters long. Spaces are allowed. The name must always be unique within the module.

   Description: Type or modify the table description within this field. The description can be a maximum of 4000 characters. This field is optional.

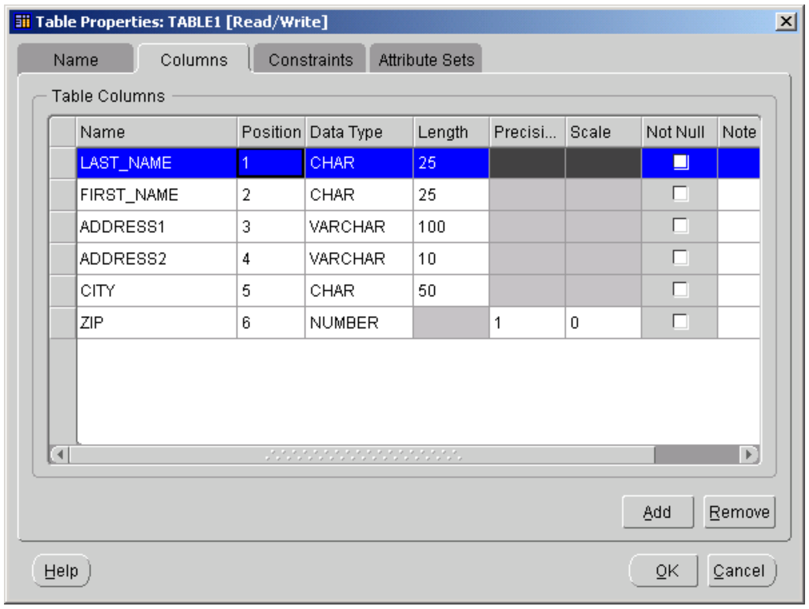
Adding Columns

To add a new column:

1. From the navigation tree, right-click the name of an object and select **Properties.**

   The Table Properties window displays.
2. Select the Columns tab, as shown in Figure 3–7.

Figure 3–7 Table Properties Columns Page



Name	Position	Data Type	Length	Precisi...	Scale	Not Null	Note
LAST_NAME	1	CHAR	25			☐	
FIRST_NAME	2	CHAR	25			☐	
ADDRESS1	3	VARCHAR	100			☐	
ADDRESS2	4	VARCHAR	10			☐	
CITY	5	CHAR	50			☐	
ZIP	6	NUMBER		1	0	☐	

3. Click **Add.**

   A blank row displays in the columns field.

4. Type the following information to define the new column:

   Name: Type a name for the column. 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 column name must be unique within the table. Spaces are allowed.

   Position: By default, the columns are sorted in the order they are created. However, Warehouse Builder enables you to reorder the columns. See "Reordering Columns in a Table" on page 3-14.

   Data Type: You must select the data type of the column from the drop-down list. This is a required field. The following data types are available in Warehouse Builder:

   Length: Define the length of the column. Length is defined only for character data types. Depending on the data type you selected, this field may be required, non-editable, or optional.

   Precision: Define the total number of digits allowed for the column. Precision is defined only for numeric data types. Depending on the data type you selected, this field may be required, non-editable, or optional.

   Scale: Define the total number of digits to the right of the decimal point. Scale is defined only for numeric data types. Depending on the data type you selected, this field may be required, non-editable, or optional.

   Not Null: By default, all columns in a table allow nulls. Null means the absence of a value. A check mark against this field indicates that the column cannot contain a NULL or empty value. For example, you can define a NOT NULL constraint to require that a value be input in the last_name column for every row of the employees table. This is an optional field.

   Note: Type a description for the column up to 4000 characters long. This is an optional field.

Available Data Types

The following data types can be used to create columns:

- CHAR: Select the CHAR data type to store fixed-length character data. For length, specify a maximum size up to 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.
- DATE: Select the DATE data type to store 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: Select the FLOAT data type when the data is 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.
- NUMBER: Select the NUMBER data type to store 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.
- VARCHAR: A VARCHAR field is a length-value data type. It consists 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 datafile. In that case, the length is in characters.
- VARCHAR2: Select VARCHAR2 data type to store 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.

Reordering Columns in a Table

By default, columns in a table are displayed in the order they are created. This order is also propagated to the DDL script generated by Warehouse Builder to create the table. If this default ordering does not suit your application needs, or if you want to further optimize query performance, you can reorder the columns.

To change the position of a column:

1. From the navigation tree, right-click a table name and select **Properties.**

   The Properties window displays.

2. Select the Columns tab.

   The Columns page displays all the columns defined for the object.

3. Select the gray square located to the left of the column name.

   The entire row is highlighted. Wait until the cursor appears as crosshairs.

4. Drag the row up or down and drop it into its new position.

   The position of the column is now updated.

5. Click **OK.**

Editing Constraints

Constraints are used to enforce the business rules you want to associate with the information in a database and to prevent the entry of invalid data into tables. 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.

The following constraints can be defined on tables:

- **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 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.

Defining Constraints in Warehouse Builder

You can define UK, PK, FK, and Check constraints in Warehouse Builder. Keep in mind the following rules:

- When you create a UK constraint, you can later change the constraint type to PK using the Table Properties window.
- When you create a PK constraint, you can later change the constraint type to UK using the Table Properties window.
- When you create a FK constraint, you cannot change the constraint type. You must first drop the constraint and then create a new one using the Table Properties window.
- When you create a Check constraint, you cannot change the constraint type. You must first drop the constraint and then create a new one using the Table Properties window.
- If you want to change a Check or PK constraint to a FK or Check, you must first drop the constraint and then create a new one using the Table Properties window.

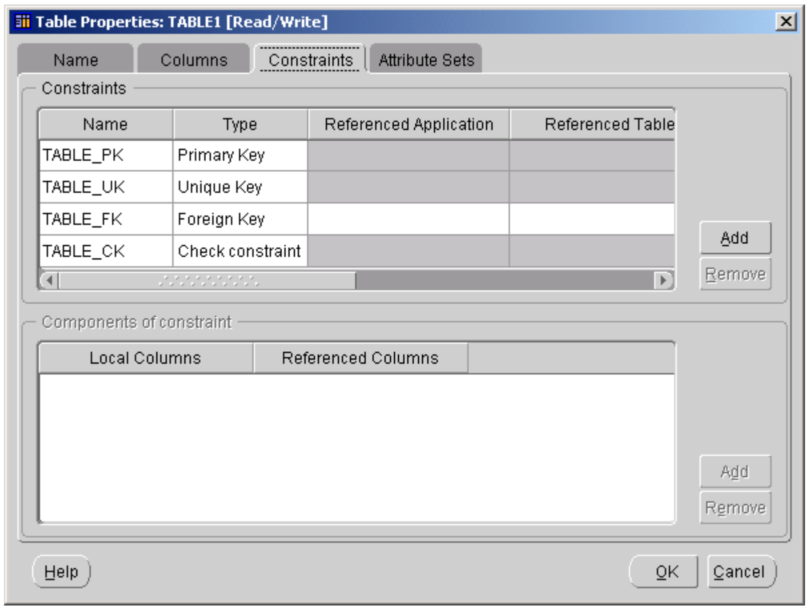
To add constraints to a table:

1. Open the Properties window for an object by right-clicking its name and selecting **Properties.**

   The Properties window displays.

2. Select the Constraints tab, as shown in Figure 3–8.

Figure 3–8 Table Properties Constraints Page



Name	Type	Referenced Application	Referenced Table
TABLE_PK	Primary Key		
TABLE_UK	Unique Key		
TABLE_FK	Foreign Key		
TABLE_CK	Check constraint		

3. Click the **Add** button next to the Constraint field.

   A blank row displays in the Constraints field.

4. Provide the following information to define a constraint:

   Name: Type a name for the constraint. 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 name up to 4000 characters long. Spaces are allowed. The constraint name must always be unique within the module.

   Type: Select the type of constraint from the drop-down list: check constraint, foreign key, unique key, or primary key.

5. Specify the Referenced Application.

   If you are creating a foreign key constraint, you must select the name of the referenced module that contains the referenced table from the drop-down list. This can be a different module from your current location. This field is required for FKs and not allowed for other constraints.

6. Specify the Referenced Table.

   If you are creating a foreign key constraint, you can select the name of the referenced table that contains the referenced key, from the drop-down list. This field is required for FKs and not allowed for other constraints.

7. Specify the Referenced Key.

   If you are creating a foreign key constraint, you select the name of the referenced key, from the drop-down list. This field is required for FKs and not allowed for other constraints.

8. Specify the Check Condition.

   If you are creating a check constraint, you must type a condition or rule in this field. For example, Status = Active. If you leave this field blank, an error is generated during validation and you cannot generate valid code for this constraint.

   he 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 Runtime Audit Browser for details.

9. Click **OK** to close the Properties window.

Deleting Constraints

To delete a constraint:

1. Right-click a table name and select **Properties.**

   The Table Properties window displays.

2. Select the Constraints tab.

   All the constraints defined for the table are displayed.

3. Select the Constraint you want to delete.

4. Click **Remove.**

   The Delete Confirmation dialog displays.

5. Click **OK** to remove the constraint.

Adding Attribute Sets

Warehouse Builder enables you to define attribute sets, or groups of columns, for every table. 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.

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

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-type: 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 navigation tree, right-click the table name and select **Properties.**

   The Properties window displays.

2. Select the Attribute Sets tab.

   Warehouse Builder displays the attribute sets defined for the table.

3. Click **Add.**

A blank row displays in the Attribute sets of the entity field.

4. Provide the following information to define the attribute set:

   Name: Type 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.

   Type: Select the type of attribute set from the drop-down list, USER_DEFINED or BRIDGE_ TYPE.

   Description: Type a description for the attribute set. This is an optional field.

5. In the **Attributes of the selected attribute set,** click **Include** for each attribute you want to include. The order in which you select the columns determines their initial order in the attribute set.

   You can click **Select All** to include all the columns in the attribute set or **Deselect All** to exclude all the columns from the attribute set. To remove a column from the attribute set, click the check box again to remove the check mark.

   Tip: To change the position of an attribute in the attribute set, click the gray square located to the left of the attribute, drag it up or down, and drop it in its new location. When you re-open the Table Properties Attribute Set tab, it will display the order of the selected attributes followed by that of the non-selected attributes.

6. If you selected BRIDGE-TYPE, click **Advanced.**

   Warehouse Builder displays the Advanced Attribute Set Properties dialog.

7. For each attribute in the bridge-type attribute set, specify the following properties. These properties determine how the objects appear and display in Oracle Discoverer.

   Hidden: Click this check box to hide unused or obsolete columns when the table is viewed in another application. In the Discoverer Administration Edition, hidden columns are grayed out. In the Discoverer Plus Edition, hidden columns are not displayed.

   Aggregation: Select an aggregation for numerical attributes SUM, MIN, MAX, AVG, COUNT, or DETAIL for no aggregation. The default is SUM.

   Position: Select the default attribute position: DATA POINT, PAGE, SIDE, TOP, or TOP/SIDE. The default is DATA POINT.

   Item Class: Check for TRUE or uncheck for FALSE. The default is FALSE.

   Heading: Type the heading text.

   Format: Type the text for the format field.

8. Click **OK** to close the Advanced Attribute Set Properties dialog.

9. Click **OK** to close the Properties window.

Using External Tables

External tables are database objects in the Oracle Database database. You cannot use external tables with any other database type or any Oracle database previous to the Oracle Database release.

External tables are tables that represent data from flat files in a relational format. They are read-only tables that behave similarly to regular source tables in Warehouse Builder. When you create and define an external table, the metadata for the external table is saved in the Warehouse Builder repository. You can use these external table definitions in mappings to design how you want to move and transform data from flat file sources to your targetS.

The following sections provide information about external tables:

- About External Tables
- Creating a New External Table Definition
- Using the External Table Editor
- Reconciling an External Table Definition with a Record in a File
- Editing External Table Definitions

For related information, see the following topics:

- About Flat File Sources and Targets on page 4-20
- Oracle Database Database Utilities Manual

About External Tables

An external table is a read-only table that is associated with a single record type in external data such as a flat file. External tables represent data from non-relational source in a relational table format. When you use an external table in a mapping, columns properties are based on the SQL properties you defined when importing the flat file. For more information on SQL properties for flat files, see "SQL Properties" on page 4-33.

When you use an external table as a source in a mapping, you can use it as you would a regular 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: External tables are source tables only. If you connect an External Table operator as a target in the Mapping Editor, Warehouse Builder returns a validation error when you run the mapping.

External Tables 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, Warehouse Builder generates SQL code. If the data is to be joined with other tables or requires complex transformations, use an external table.

When you use a flat file operator, Warehouse Builder generates SQL*Loader code. In cases where large volumes of data are to be extracted and little transformation is required, you can use the flat file operator. From the flat file operator, you could load the data to a staging table, add indexes, and perform transformations as necessary. The transformations you can perform on data introduced by a flat file operator are limited to SQL*Loader transformations only.

In Warehouse Builder, you can use an external table to combine the loading and transformation within a single set-based SQL DML statement. You do not have to stage the data temporarily before inserting it into the target table.

For more information on differences between external tables and SQL*Loader (flat file operators in Warehouse Builder), see the *Oracle Database Database Utilities Manual.*

Creating a New External Table Definition

After you use the Flat File Sample Wizard to import metadata, you can create an external table based on a single flat file record type. For information on importing flat file data, see "Using the Flat File Sample Wizard" on page 4-23.

To create a new external table definition:

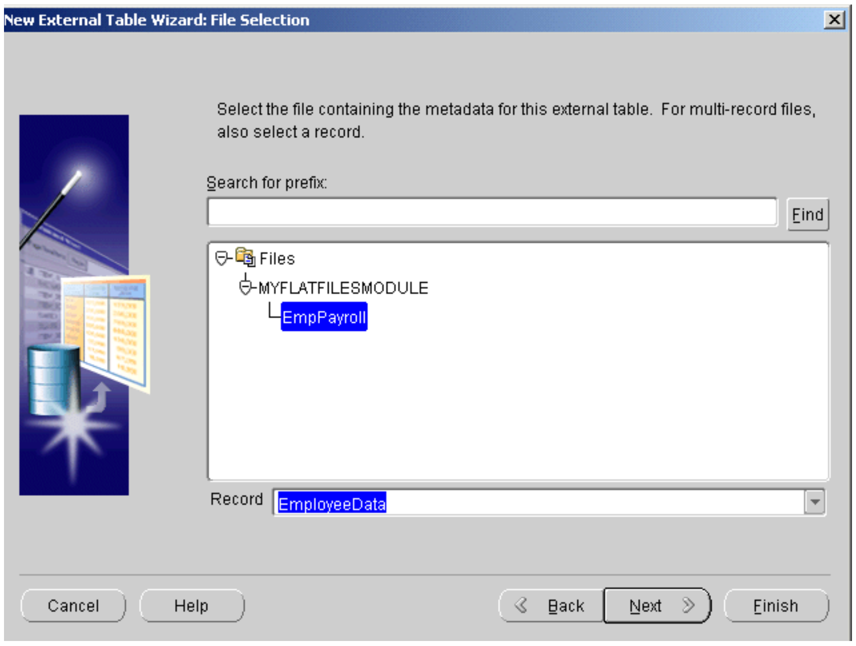
1. From the Warehouse Builder navigation tree expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the external table.
3. Right-click the External Tables node and select **Create External Table.**

   Warehouse Builder displays the Welcome page.
4. Click **Next.**

   The wizard displays the Name page.
5. Type a name and optional description for the external table.
6. Click **Next.**

   The wizard displays the File Selection page, as shown in Figure 3–9.

Figure 3–9 File Selection Page in External Table Wizard


New External Table Wizard: File Selection

Select the file containing the metadata for this external table. For multi-record files, also select a record.

Search for prefix:

Files
MYFLATFILESMODULE
EmpPayroll

Record: EmployeeData

7. The wizard lists all the flat files available in the repository. Select a file upon which to base the external table. To search through long lists of files, type the first few letters of the file name and click **Find.**

If you cannot find a file, make sure you imported the metadata for the file. For information on importing flat files, see "Using the Flat File Sample Wizard" on page 4-23.

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.

8. Click **Next.**

   The wizard displays the Locations page.

9. You can select a location from the drop down box which lists the locations associated with flat files. Alternatively, you can leave the location unspecified. If you do not specify a location in the wizard, you can later specify a location on the external table properties sheet. For more information, see "Defining Locations" on page 3-2.

   > **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.

10. Click **Next.**

    The wizard displays the Finish page. This page summarizes the information you entered on each of the wizard pages. Verify the information.

11. Click **Finish.**

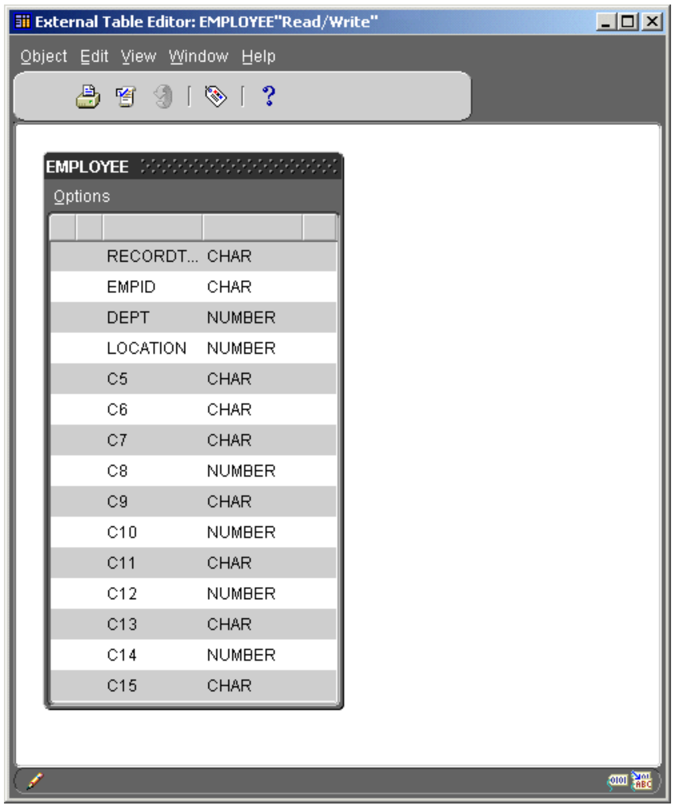
    The wizard creates the external table and inserts its name in the navigation tree.

Using the External Table Editor

Warehouse Builder provides an editor dialog and right-click pop up options similar to regular tables.

Figure 3–10 shows the External Table Editor.

Figure 3–10 External Table Editor


External Table Editor: EMPLOYEE"Read/Write"
Object Edit View Window Help
EMPLOYEE
Options

RECORDT...	CHAR
EMPID	CHAR
DEPT	NUMBER
LOCATION	NUMBER
C5	CHAR
C6	CHAR
C7	CHAR
C8	NUMBER
C9	CHAR
C10	NUMBER
C11	CHAR
C12	NUMBER
C13	CHAR
C14	NUMBER
C15	CHAR

You can print the external table diagram, validate external table definitions, synchronize the external table definitions, and invoke Warehouse Builder Browser to run reports on the selected external table. For more information on options available in both the table editor and external table editor, refer to "Using the Table Editor" on page 3-10.

Reconciling an External Table Definition with a Record in a File

One option available in the external table editor and not the table editor is the reconcile option. Use **Reconcile** to update the external table definition with metadata changes made to the file associated with the external table.

To reconcile an external table definition with a record in a file:

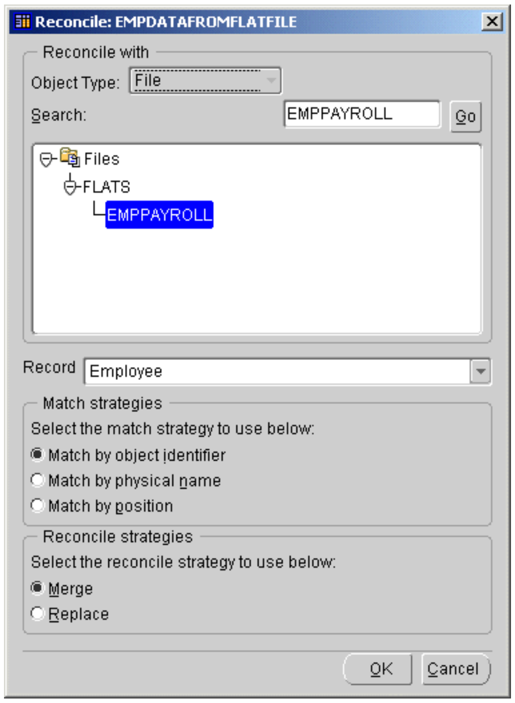
1. Launch the external table editor by right-clicking on an external table from the navigation tree and selecting **Editor.**

   Warehouse Builder displays the external table editor.

2. From the menu, select **Object** then **Reconcile.**

   The external table editor displays the reconcile dialog as shown in Figure 3–11. Use the reconcile dialog to specify a record in a flat file.

Figure 3–11 Reconcile Dialog for External Tables


Reconcile: EMPDATAFROMFLATFILE

Reconcile with

Object Type: File

Search: EMPPAYROLL

Files
FLATS
EMPPAYROLL

Record: Employee

Match strategies

Select the match strategy to use below:

(●) Match by object identifier
( ) Match by physical name
( ) Match by position

Reconcile strategies

Select the reconcile strategy to use below:

(●) Merge
( ) Replace

3. In **Search,** type in a file name and select **Go** to find a specific flat file. Or scroll through the list and select a flat file.
4. For files with multiple record types, select a record from **Record.**

   You must select a record. **Match strategies, Reconcile strategies,** and **OK** are not available until you specify a record to reconcile against.
5. Set the **Match strategies.** Based on your selection, Warehouse Builder searches for matches and updates the external table with the information from the flat file.

   Matching by Object Identifier: This strategy compares the field IDs of that the external table columns references with the field IDs in the flat file.

   Matching by Physical Name: This strategy compares the physical names in the external table with the physical names in the flat file.

   Matching by 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. If the external table has more attributes than the flat file, the excess attributes are removed from the external table.
6. Set the **Reconcile Strategies.** Use these settings to indicate how Warehouse Builder handles differences in metadata between the existing external table definition and the record you specified.

   Merge: Warehouse Builder combines the metadata from the existing external table definition and the record you specified.

   Replace: Warehouse Builder deletes metadata from the external table definition if it does not match the metadata from the record you specified. The resulting reconciled external table contains metadata that matches the file record metadata.

Editing External Table Definitions

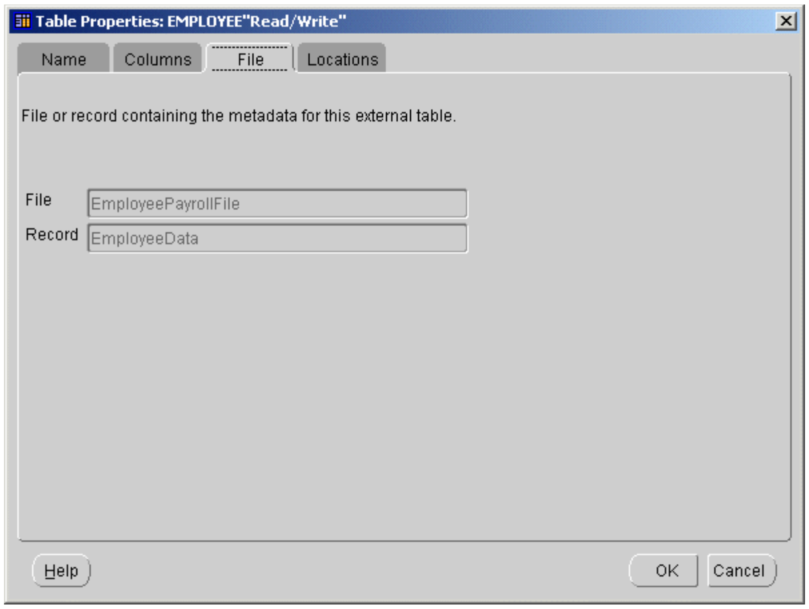
From the External Table Editor, you can access the External Table Properties window to edit the name, description, columns, file and location.

To edit external table properties, select **Object** then **Properties** from the External Table Editor menu or right-click the name of a table from the navigation tree and select **Properties**. The External Table properties window displays. The tabs and properties that you can edit depend on how you defined the external table in the repository.

In most cases, the External Table Properties window displays as shown in Figure 3–12 with the following four tabs:

- **Name:** Use the Name tab to rename the external table. The same rules for renaming tables apply to external tables. For more information, see "Renaming a Table" on page 3-12.
- **Columns:** 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 3-11.
- **File:** 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 "Reconciling an External Table Definition with a Record in a File" on page 3-22.
- **Locations:** Use the Location tab to view or change the flat file location, as shown in Figure 3–12.

Figure 3–12 External Table Properties Window



File or record containing the metadata for this external table.

File: EmployeePayrollFile
Record: EmployeeData

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.

Access Parameters

Access parameters define how to read from the flat file. In some cases, the External Table Properties window displays the Access Parameters tab instead of the File tab.

The tab for the access parameters displays under the following conditions:

- You imported an external table from another repository. In this case, you can view and edit the access parameters.
- You created an external table in an Oracle database and imported its definition into Warehouse Builder. In this case, you can view and edit the access parameters.
- You used the New 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 datafile are represented in the external table as columns. For example, if the datafile contained a field `emp_id` with a datatype of INTEGER(2), the access parameters could indicate that field be converted to a character string column in the external table.

Although you can make changes to the access parameters that affect how Warehouse Builder generates and deploys the external table, it is not recommended. Warehouse Builder does not validate the changes. For more information on the access parameters clause, see *Oracle Database Database Utilities Manual.*

Using Views

In Warehouse Builder, you can define views and materialized views. This section describes views. For information on materialized views, see "Using Materialized Views" on page 3-28.

The following sections provide information about using views:

- About Views on page 3-25
- Creating a View Definition on page 3-25
- Editing a View Definition on page 3-28

About Views

Views are used to simplify the presentation of data or restrict access to data. Often the data that users are interested in is stored across multiple tables with many columns. When you create a view, you create a query stored to retrieve only the relevant data or only data that the user has permission to access.

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

Creating a View Definition

When you create a view with the New View Wizard, the wizard prompts you to type the following information:

- View name and description
- Aliases for columns

- Queries that define the view (optional)
- Logical constraints (optional)

To create a view definition:

1. From the Warehouse Builder navigation tree expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the view.
3. Right-click **View** and select **Create View.**

   Warehouse Builder displays the welcome page for the wizard.

   Note: You can also define a View from the Mapping Editor and model your own query.

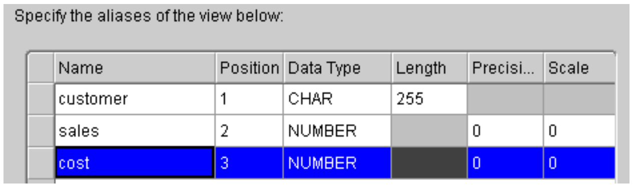
   ---

4. Click **Next.**

   The wizard displays the Name page. Type the a name and description for the view. The description is optional.
5. Click **Next.**

   The wizard displays the Columns page, as shown in Figure 3–13.

Figure 3–13 Columns Page



Specify the aliases of the view below:

Name	Position	Data Type	Length	Precisi...	Scale
customer	1	CHAR	255		
sales	2	NUMBER		0	0
cost	3	NUMBER		0	0

6. To define a column, click **Add.**

   Type the column name and select the data type.

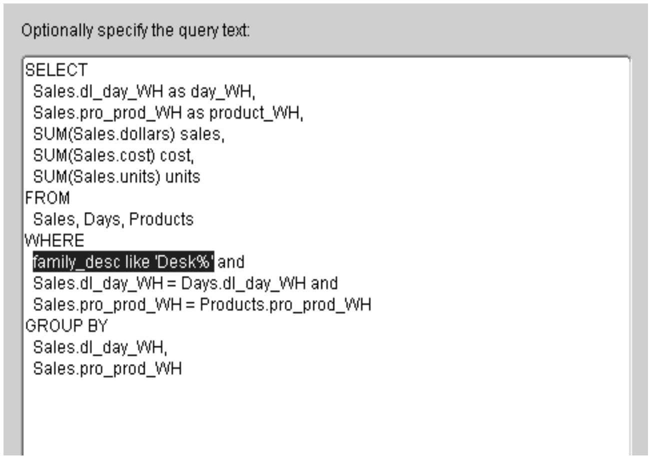
   Repeat this procedure for each column.
7. Click **Next.**

   The wizard displays the Query Text page. You can either type the query definition or click **Next** to continue.

   If you type a query definition into the Query Text page, be sure to type a valid statement. Warehouse Builder does not validate the text in the Query Text page and will attempt to deploy a view even if the syntax is invalid.

   Figure 3–14 shows a sample query text.

Figure 3–14 Query Text


Optionally specify the query text:

```
SELECT
 Sales.dl_day_WH as day_WH,
 Sales.pro_prod_WH as product_WH,
 SUM(Sales.dollars) sales,
 SUM(Sales.cost) cost,
 SUM(Sales.units) units
FROM
 Sales, Days, Products
WHERE
 family_desc like 'Desk%' and
 Sales.dl_day_WH = Days.dl_day_WH and
 Sales.pro_prod_WH = Products.pro_prod_WH
GROUP BY
 Sales.dl_day_WH,
 Sales.pro_prod_WH
```

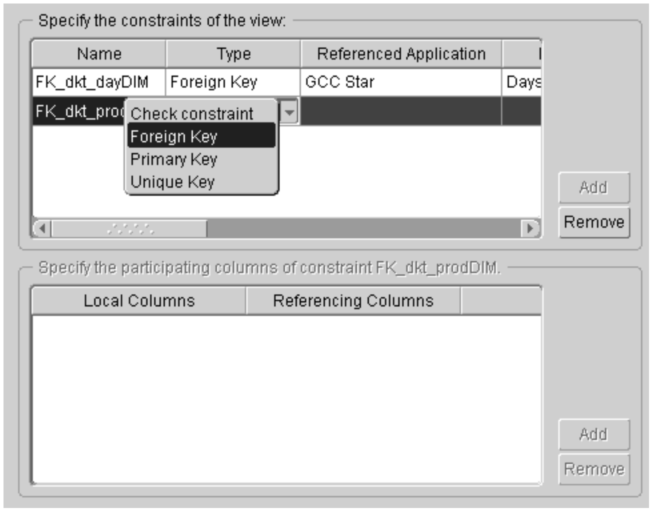
8. Click **Next.**

The wizard displays the Define Constraints page, as shown in Figure 3–15.

As an option, use this page to define logical constraints for a view. Although Warehouse Builder does not use these constraints when enumerating DDL for the view, these constraints can be useful when the view serves as a data source in a mapping. The Mapping Editor can use the logical foreign key constraints to include the referenced dimensions as secondary sources in the mapping.

For general information about constraints, see "Editing Constraints" on page 3-15.

Figure 3–15 Define Constraints Page


Specify the constraints of the view:

Name	Type	Referenced Application	I
FK_dkt_dayDIM	Foreign Key	GCC Star	Days
FK_dkt_proc			

Specify the participating columns of constraint FK_dkt_prodDIM.

Local Columns	Referencing Columns

9. Click **Next.**

Warehouse Builder displays the Finish page. Verify the description. If you need to modify the definition, click **Back.**

10. Click **Finish.**

The wizard creates a definition for the view, stores this definition in the repository, and inserts its name in the navigation tree.

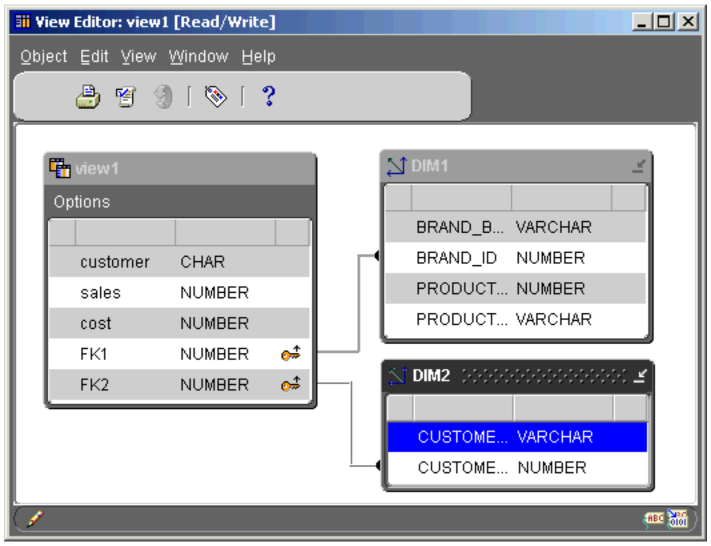
Editing a View Definition

To rename a view, right-click the view name and select **Rename.** Type the new name over the highlighted object name.

You can display the view in the View Editor. Use the property sheet to edit the view.

To open the View Editor, right-click the view and select **Edit.** The editor diagrams the view and its references, as shown in Figure 3–16.

Figure 3–16 View Editor



customer	CHAR
sales	NUMBER
cost	NUMBER
FK1	NUMBER
FK2	NUMBER

BRAND_B...	VARCHAR
BRAND_ID	NUMBER
PRODUCT...	NUMBER
PRODUCT...	VARCHAR

CUSTOME...	VARCHAR
CUSTOME...	NUMBER

To open the View properties sheet, right-click the view and select **Properties.** You can modify the view definition by editing the property sheet. For examples on editing a property sheet, see "Editing Table Definitions" on page 3-11.

Using Materialized Views

In Warehouse Builder, you can define views and materialized views. This section discusses materialized views. For information on conventional views, see "Using Dimensions" on page 3-39.

The following sections provide information about using materialized views:

- About Materialized Views on page 3-28
- Creating a Materialized View Definition on page 3-29
- Editing a Materialized View Definition on page 3-31
- Configuring Materialized Views on page 5-17

About Materialized Views

In Warehouse Builder, you can create materialized views to improve query performance.When you create a materialized view, you create a set of query commands that aggregate or join data from multiple tables. Materialized views provide precalculated data that can be reused or replicated to remote data marts. For example, data about company sales is widely sought throughout an organization.

When you create a materialized view in Warehouse Builder, you can configure it to take advantage of the query rewrite and fast refresh features available in Oracle

Database. For information on query rewrite and fast refresh, "Fast Refresh for Materialized Views" on page 5-18.

Creating a Materialized View Definition

When you create a materialized view with the New Materialized View Wizard, the wizard prompts you to type the following information:

- Materialized view name
- Materialized view description
- Column names
- Queries that define the materialized view
- Logical constraints (optional)

To create a materialized view definition:

1. From the Warehouse Builder navigation tree 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 **Create Materialized View.**

   Warehouse Builder displays the welcome page for the New Materialized View wizard.

   Note: You can also define a Materialized View from the Mapping Editor.

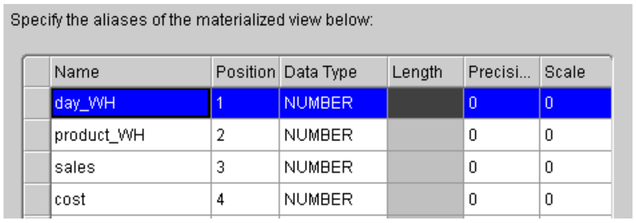
4. Click **Next.**

   The wizard displays the Name page.

5. Type the name and description for the view. The description is optional.
6. Click **Next.**

   The wizard displays the Columns page, as shown in Figure 3–17.

Figure 3–17 Columns Page



Specify the aliases of the materialized view below:

Name	Position	Data Type	Length	Precisi...	Scale
day_WH	1	NUMBER		0	0
product_WH	2	NUMBER		0	0
sales	3	NUMBER		0	0
cost	4	NUMBER		0	0

7. To define a column, click **Add.**

   Type the column name and select the data type. Repeat this procedure for each column.

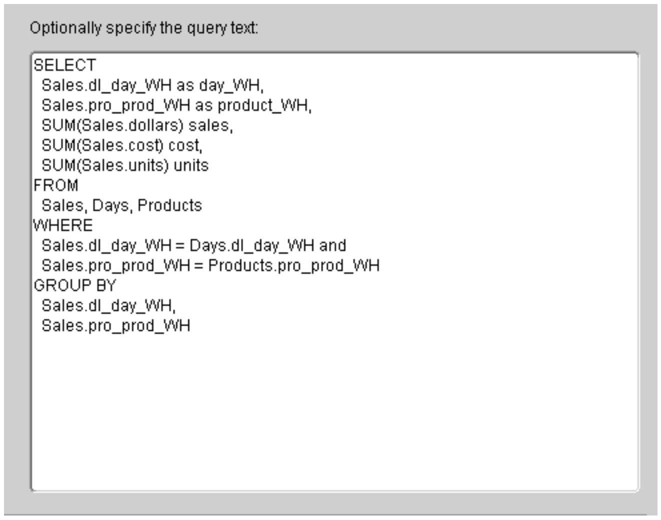
8. Click **Next.**

   The wizard displays the Query Text page. You can either type the query definition or click **Next** to proceed to the Define Constraints page.

If you type a query definition into the Query Text page, be sure to type a valid statement. 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 on the Query Text page. Warehouse Builder does not validate the text in the Query Text page and will attempt to deploy a view even if the syntax is invalid.

Figure 3–18 shows sample query text.

Figure 3–18 Sample Query Text



Optionally specify the query text:

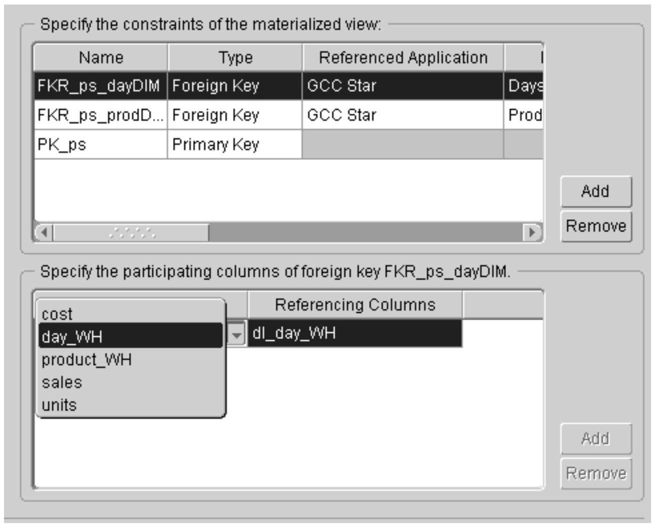
```
SELECT
 Sales.dl_day_WH as day_WH,
 Sales.pro_prod_WH as product_WH,
 SUM(Sales.dollars) sales,
 SUM(Sales.cost) cost,
 SUM(Sales.units) units
FROM
 Sales, Days, Products
WHERE
 Sales.dl_day_WH = Days.dl_day_WH and
 Sales.pro_prod_WH = Products.pro_prod_WH
GROUP BY
 Sales.dl_day_WH,
 Sales.pro_prod_WH
```

9. Click **Next.**

   The wizard displays the Define Constraints page. As an option, you can define constraints for the materialized view, as shown in Figure 3–19.

   These constraints are for logical design purposes only and Warehouse Builder does not use these constraints when enumerating DDL for the view. For general information about constraints, see "Editing Constraints" on page 3-15.

Figure 3–19 Define Constraints Page



Specify the constraints of the materialized view:

Name	Type	Referenced Application	
FKR_ps_dayDIM	Foreign Key	GCC Star	Days
FKR_ps_prodD...	Foreign Key	GCC Star	Prod
PK_ps	Primary Key		

Specify the participating columns of foreign key FKR_ps_dayDIM.

	Referencing Columns
day_WH	dl_day_WH

10. Click **Next.**

    The wizard displays the Finish page. Verify the summary, and if you need to modify the definition, click **Back.**

11. Click **Finish.**

    The wizard creates a definition for the materialized view, stores this definition in the database module, and inserts its name in the warehouse module navigation tree.

Editing a Materialized View Definition

To rename a materialized view, right-click the view name and select **Rename.** Type the new name over the highlighted object name.

You can view the materialized view in the Materialized View Editor. Use the property sheet to edit the materialized view.

To open the Materialized View Editor, right-click the materialized view and select **Editor.** The editor diagrams the materialized view and its references.

To open the Materialized View properties sheet, right-click the materialized view and select **Properties.** You can modify the view definition by editing the property sheet.

For examples on editing a property sheet, see "Editing Table Definitions" on page 3-11.

Using Sequences

A sequence is a database object that generates a serial list of unique numbers. You can use sequences to generate unique primary key values and to coordinate keys across multiple rows or tables. Sequence values are guaranteed to be unique. When you create a sequence in Warehouse Builder, you are creating sequence definitions that are saved in the repository. Sequence definitions can be used in mappings to generate unique numbers while transforming and moving data to your target system.

The following sections provide information about using sequences:

- About Sequences on page 3-31
- Creating a Sequence Definition on page 3-31
- Using the Sequence Editor on page 3-32
- Editing Sequence Definitions on page 3-32

About Sequences

A sequence is referenced in SQL statements with the NEXTVAL and CURRVAL pseudo columns. Each new sequence number is incremented by a reference to the pseudo column NEXTVAL, while the current sequence number is referenced using the pseudo column CURRVAL. When you define a sequence, Warehouse Builder creates these attributes.

In Warehouse Builder, you can also import sequence definitions from existing source systems using the Import Object Wizard. For more information, see "Importing Metadata into Target Modules" on page 3-47.

Creating a Sequence Definition

To create a new sequence:

1. From the Warehouse Builder navigation tree, expand the warehouse module node.
2. Right-click Sequence and select **Create Sequence** from the pop-up menu.

   Warehouse Builder displays the New Sequence Wizard.
3. Enter the following information:

   Name: Type a name for the column. 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 column name must be unique within the table. Spaces are allowed.

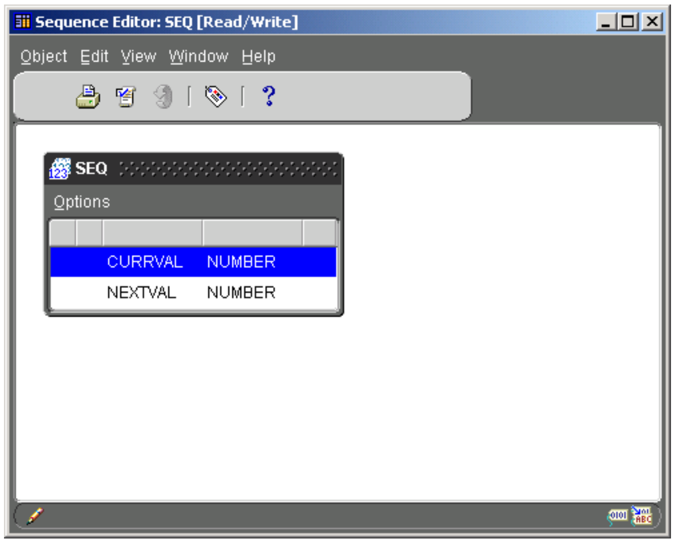
   Description: Optionally type a description for the sequence. The description cannot exceed 4000 characters.
4. Click **OK.**

   Warehouse Builder stores the definition for the sequence and inserts its name in the navigation tree.

Using the Sequence Editor

After you define a sequence in Warehouse Builder, you can use the Sequence Editor to view columns. To open the Sequence Editor, right-click the name of a sequence and select **Editor.** Figure 3–20 shows the Sequence Editor.

Figure 3–20 Sequence Editor


Sequence Editor: SEQ [Read/Write]
Object Edit View Window Help
SEQ
Options

CURRVAL	NUMBER
NEXTVAL	NUMBER

Use the Sequence Editor Menu, the Sequence Editor Navigation bar, or the right-click pop up options to print the sequence diagram, validate sequence definitions, synchronize the sequence definitions, and invoke Warehouse Builder Browser to run reports on the selected sequence.

To view metadata reports on any sequence, select View, then Reports. For information on installing and configuring the Warehouse Builder Browser, refer to the Oracle Warehouse Builder Installation and Configuration Guide.

Editing Sequence Definitions

From the Sequence Editor, you can access the Sequence Properties window to edit the name, description, and column notes of a sequence.

To edit sequence properties, select right-click the name of a sequence from the navigation tree and select **Properties or double-click the name of the sequence.** The Sequence Properties window displays two tabs: General and Columns.

Click these tabs to perform the following tasks:

- Rename a sequence
- Edit sequence columns

Renaming a Sequence

To edit the name and description of a sequence:

1. Right-click the name of a sequence and select **Properties.**

   The Sequence Properties window opens.

2. Select the General tab:

   Name: Type a new name for the sequence. In physical mode, you must type a unique name between 1 and 30 valid characters. Spaces are not allowed. In logical mode, you can type a name up to 200 characters long. Spaces are allowed. The name must be unique within the module.

   Description: Type or modify the sequence description within this field. The description can be a maximum of 4000 characters. This field is optional.

Tip: If you want to rename a sequence without editing its description, right-click the name of the sequence from the navigation tree and select **Rename.**

Editing Sequence Column Descriptions

To edit the column descriptions of a sequence:

1. Right-click the name of a sequence and select **Properties.**

   The Sequence Properties window opens.

2. Select the Columns tab.
3. Scroll to the **Note** field and type or modify the description for the selected column.

Using Advanced Queues

Oracle Advanced Queues (AQs) provide a database-integrated message queuing system that plays a central role in enterprise data integration. Warehouse Builder enables you to import Advanced Queue definitions and use AQs to move data from your sources to the target system.

For more information on Oracle Advanced Queuing, see *Oracle Database Application Developer's Guide- Advanced Queuing.*

The integration of Oracle Advanced Queues (AQs) in Warehouse Builder enables the following:

- Use of AQs as data sources or targets in your data warehouse design.
- Integration of MQ Series and TIBCO applications as data sources.
- Use of AQs to propagate source data changes.

This section includes the following topics:

For related information, see:

- Configuring Advanced Queues on page 5-15
- Mapping Advanced Queue Operator on page 8-19

About Advanced Queues

Web enabled business applications often communicate with one another through message queues. Advanced Queuing leverages the functions of the Oracle database to persistently store these messages, to propagate them between queues on different machines and databases, and to transmit them using Oracle Net Services, HTTP(S), and SMTP. AQs enable message management and communication required for application integration. Warehouse Builder supports AQs as data sources for your warehouse design.

The following types of AQs are supported in Warehouse Builder:

- Multiconsumer AQ. The associated queue table must be created as a multiple consumer queue table in the database.
- Normal AQs (Exception AQs are not supported).
- Persistent AQs (non-persistent AQs are not supported).

This section discusses the following key concepts related to AQs.

Payloads

Payload is the data stored in a queue. It can be unstructured, such as, the data type RAW, or structured. Payloads can be structured by using Oracle object types, also known as ADTs or user defined types. Complex object types including embedded attributes, collection attributes, and XMLType and SYS.AnyData attributes are not currently supported in Warehouse Builder.

AQs require that the payload type be either an Oracle object type or RAW. To be used in Warehouse Builder, the payload must be defined as an object type containing attributes with scalar data types supported by Warehouse Builder. RAW data types are not supported.

Messages

A message is the smallest unit of information entered into or retrieved from a queue. Messages contain control information and payload data. The control information contains message properties or metadata used by AQs to manage messages. Payload data is the information stored in the queue. A message can reside in only one queue.

Creating Definitions for Advanced Queues

Currently, AQs can only be imported into Warehouse Builder from an Oracle source schema. For information on how to create an AQ in your source schema, see "Creating Advanced Queues Using SQL" on page 3-38.

Importing AQ Definitions

The imported AQ metadata contains the AQ name, description, the payload object type name, payload structure, the schema where object type is defined, and the name of the associated queue table. These definitions are imported from the agent AQ and not from the source AQ. The payload object type must reside in the same schema as the AQ. If the payload object type is not defined in the same schema as the AQ, then the AQ is imported but the payload type is not imported. In this case, the AQ will not be associated with any payload type and will be invalid. You will receive an error when you validate this AQ. Object types can be created or deleted through scripting and public API.

To import AQs into Warehouse Builder:

1. In the navigation tree, right click the target module and select **Import.**

   The Database Link Information dialog displays if you have not defined a database link.

2. Create a new database link to the system from where you are importing the AQ definitions or select a previously created database link.

3. Click **OK.**

   The Import Metadata Wizard welcome page displays.

4. Click **Next.**

   The Filter Information page displays.

5. Select **Advanced Queue** as the type of object you want to import.

6. Limit the search of the data dictionary in one of the following methods:

   Type a search pattern. For example, you can type a warehouse project name followed by a percentage sign (%) to import objects that begin with that project name. Use a percentage sign (%) as a wild card match for multiple characters and an underscore (_) as a wild card match for a single character.

   Type the maximum number of objects to retrieve.

7. Click **Next.**

   Warehouse Builder displays the names of objects that meet the filter conditions and displays the Object Selection page.

8. From the Available Objects list, expand the ADVANCED QUEUE node and select the queues you want to import.

9. Click the arrow buttons to move the selected queues to the Selected Objects list. Warehouse Builder only supports the import of persistent multiconsumer queues.

   If you are re-importing definitions, previously imported objects appear in bold.

10. Click **Next.**

    The Summary and Import page displays. This page summarizes your selections in a spreadsheet listing the name, type of object, and whether the object will be reconciled or created. Verify the contents of this page and optionally add descriptions for each of the objects.

11. Click **Finish.**

    Warehouse Builder imports the definitions for the selected queues and displays the results in the Import Results dialog. When you import a queue, its payload is also imported. You can expand the queue node to view the attributes of the payload for a queue.

The payload object type must reside in the same schema as the AQ. If the payload object type is not defined in the same schema as the AQ, then the AQ is imported but the payload type is not imported. In this case, the AQ will not be associated with any payload type and will be invalid. You will receive an error when you validate this AQ

12. Verify that the definitions have been created and click **OK** to complete the import.

    Click **Undo** if you do not want to import the definitions. Or click **Save** to save the definitions to a local drive. Warehouse Builder stores the imported definitions under the AQ node in the warehouse module navigation tree.

Note: At runtime, a temporary table must be available to use the AQ as a source. The create table script is generated when you deploy the AQ.

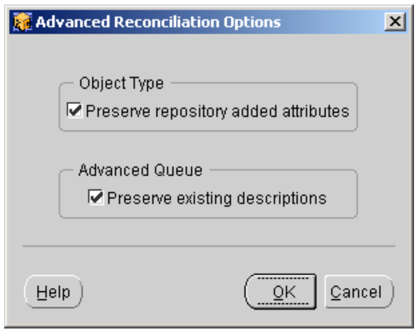
Reimporting Advanced Queue Definitions

Two advanced queues can share the same payload type. If you import a queue (Q2) whose payload has already been imported with another queue (Q1), then the attributes of Q2 are reconciled with the previously imported attributes.

To reimport advanced queues:

1. Follow the preceding steps using the Import Metadata Wizard. In the Summary and Import page, the Advanced Reconcile Options button is now enabled.

Figure 3–21 AQ Advanced Reconcile Options


Advanced Reconciliation Options

Object Type

☑ Preserve repository added attributes

Advanced Queue

☑ Preserve existing descriptions

2. Click **Advanced Reconcile Options** to select a reconciliation option.

   The Advanced Reconciliation Options dialog displays, as shown in Figure 3–21. Use this dialog to reconcile the reimported metadata with the existing definitions in the Warehouse Builder repository. You can choose to preserve object type attributes and AQ descriptions in the repository.

   Select options for reconciling the object type:

   Preserve repository added attributes: Check this option to preserve attributes added to the repository that are not present in the object to be imported.

   Select options for reconciling the AQ:

   Preserve existing descriptions: Check this option to preserve previously imported AQ definitions in the repository.

3. Click **OK** after selecting your options.

> **Note:** By default, all options are checked. Uncheck the boxes to have these repository objects replaced and not preserved.

4. Click **OK** to accept the changes. Click **Undo** to cancel the import.

   Warehouse Builder stores the imported definitions under the AQ node in the warehouse module navigation tree.

For related information, see the following sections:

- Mapping Advanced Queue Operator on page 8-19
- Configuring Advanced Queues on page 5-15

Viewing Advanced Queue Properties

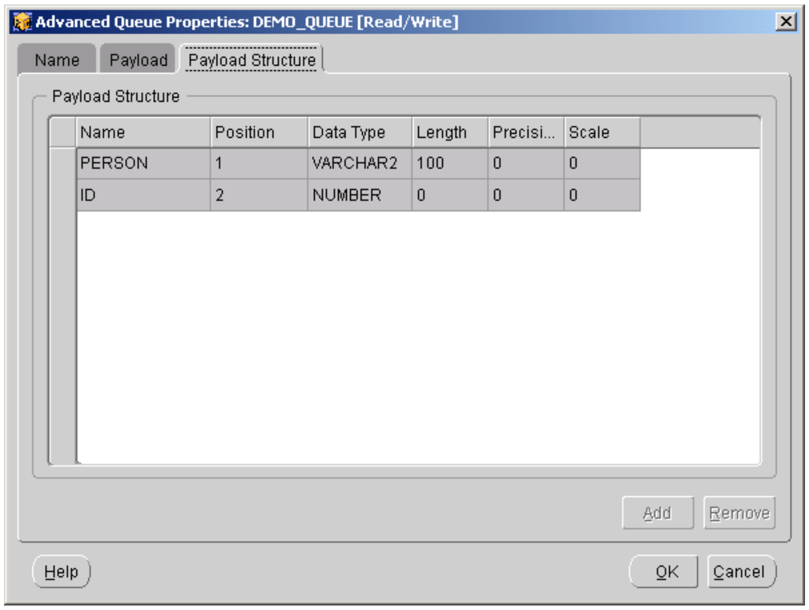
After you import AQs into Warehouse Builder, you can view their properties by right-clicking the AQ name and selecting **Properties** from the pop-up menu. The Advanced Queue Properties window displays three tabs: Name, Payload, and Payload Structure. The imported AQ properties cannot be edited.

> **Note:** The object types imported with an AQ cannot be deleted from the client UI. They can only be deleted through scripting.

You can view the following properties by clicking each of the tabs:

- **Name:** Name and description of the imported AQ.
- **Payload:** Name and description of the payload imported with the AQ.
- **Payload Structure:** Attributes of the payload imported with the AQ, such as, Name, Position, Data Type, and Length as shown in Figure 3–22.

Figure 3–22 AQ Properties Payload Structure Tab



Name	Position	Data Type	Length	Precisi...	Scale
PERSON	1	VARCHAR2	100	0	0
ID	2	NUMBER	0	0	0

Creating Advanced Queues Using SQL

The following steps outline how to create AQs in your Oracle source system using SQL:

```
SQL> grant aq_administrator_role, aq_user_role to scott;
```

(Grants the required roles to the user who will administer or use the AQ. These roles must be granted by the system administrator or the user with SYSDBA privileges, SYS or SYSTEM)

```
SQL> connect scott/tiger;
```

(Connect with the user name and password of the user who has been granted the privileges, for example, scott/tiger)

```
SQL> create type employee as object (empno number, ename
varchar2(30));
```

(Create the object structure on which the AQ will be based, for example, employee)

```
SQL> execute dbms_aqadm.create_queue_table(queue_table =>
'EMPLOYEE_QUEUE_TBL', multiple_consumers=>true queue_payload_
type => 'EMPLOYEE');
```

(Create the AQ table. Because Warehouse Builder currently supports only multiconsumer queues, multiple_consumers => true)

```
SQL> execute dbms_aqadm.create_queue('EMPLOYEE_QUEUE',
'EMPLOYEE_QUEUE_TBL');
```

(Create the AQ)

```
SQL> execute dbms_aqadm.start_queue('EMPLOYEE_QUEUE');
```

(Start the AQ)

Creating Advanced Queues Using Oracle Enterprise Manager

The following steps outline how to create AQs in your Oracle source system using Oracle Enterprise Manager:

1. Grant the AQ_ADMINISTRATOR_ROLE and AQ_USER_ROLE to the user who will administer or use the AQ. Expand the navigation tree by expanding the **Database** node, then **Security,** then **Users,** and then **Role** (must have SYSDBA privileges).
2. Create the object type. From the **Object** menu, select **Create,** and then **Object Type.** Specify the attributes and create the object type.
3. Create the AQ table. Expand the navigation tree by expanding the **Databases** node, then **Database Name,** then **Schema,** then Advanced Queues, and then double click **Queue Tables** to create the AQ table.
4. Create the AQ by selecting the **Object** menu, then **Create,** and then **Queue.**
5. Start the AQ by expanding the **Databases** node, then **Database Name,** then **Schema,** then **Advanced Queues,** then **Queue Tables,** then **Schema Name,** then **Queue Table Name**. Highlight the queue name then from the **Object** menu select **Queue Start.**

   For more details, see the *Oracle Enterprise Manager User's Guide.*

Using Dimensions

Dimensions are the primary organizational unit of data in a star schema. Warehouse Builder uses dimensions to organize and index data for cubes. Examples of some commonly used dimensions are Customer, Product, and Time.

When you define a dimension, you need to define its hierarchies, levels, and level relationships. The levels represent the level of aggregation and the hierarchies describe parent-child relationships among a set of levels. Dimension hierarchies are logical structures that use ordered levels as a means of organizing data.

Query performance can be improved using Dimensions because users often analyze data by drilling down on known hierarchies. An examples of a hierarchy is the time hierarchy of Year, Quarter, Month, Day. Oracle Database makes use of defined hierarchies by rewriting queries to retrieve data from summary rather than detail tables. The rewritten queries have improved performance.

Typical 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 defined by a Create Dimension statement maximize the number of query rewrites by the Oracle Database server.

Rules for Dimension Objects

A dimension definition includes a dimension object definition and a dimension table definition. This section provides information about the dimension object.

Table 3–2 summarizes the rules for dimensions.

Table 3–2 Warehouse Builder Rules for Dimension Objects

Rule	Description
Denormalized	A generated dimension object is defined on a single table. Warehouse Builder does not currently support the definition of a dimension object on a set of normalized tables.
Functional Dependence	Child values must uniquely determine their parent value. For example, a city rolls up to one and only one state, a month determines a quarter, and a product determines a brand. These relationships must obtain in the physical data else queries can return incorrect result sets.
Unique Key Generation	Warehouse Builder implements a unique key constraint only on the lowest level of a hierarchy.
Foreign Key References	A table can reference only the lowest level of a hierarchy (where the unique key constraint is always defined).

About Levels and Hierarchies

Dimension objects consist of a set of levels and a set of hierarchies defined over those levels. The levels represent levels of aggregation. Hierarchies describe parent-child relationships among a set of levels.

For example, a typical calendar dimension could contain five levels. Two hierarchies can be defined on these levels:

H1: YearL > QuarterL > MonthL > WeekL

H2: YearL > QuarterL > WeekL > DayL

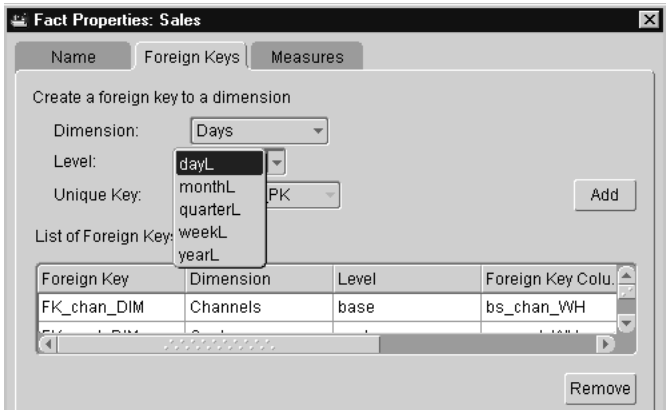
The hierarchies are described from parent to child, so that Year is the parent of Quarter, Quarter the parent of Month, and so forth.

About Unique Key Constraints

When you create a definition for a hierarchy, Warehouse Builder creates an identifier for each level of the hierarchy and a unique key constraint on the lowest level. Warehouse Builder uses the identifiers during the generation phase to build a DDL script to create the dimension object.

When you create a foreign key reference on a cube that points to a dimension, Warehouse Builder shows the unique key constraint and the other identifier keys as candidates for the referenced column. A cube can reference only the lowest level of a hierarchy because it contains a unique key constraint. If you select any other level, the definition is invalid.

Figure 3–23 Cube Properties Dialog Showing the Foreign Keys Tab



Create a foreign key to a dimension

Dimension: Days

Level: dayL

monthL

quarterL

weekL

yearL

Unique Key: PK

List of Foreign Keys:

Foreign Key	Dimension	Level	Foreign Key Colu.
FK_chan_DIM	Channels	base	bs_chan_WH

About Mixed Levels of Aggregation

An application can require two hierarchies that start at different levels of aggregation. For example, you can have the following hierarchies:

H1: YearL > QuarterL > MonthL > DayL

H2: YearL > WeekL > DayL

H3: YearL > QuarterL > MonthLowL

To model this mixed case using Warehouse Builder:

- The dimension table must contain the additional column MonthLow.
- The MonthLow column must be populated with unique values.
- A separate MonthLow level must be defined for the dimension.

For this set of hierarchies, Warehouse Builder generates six level identifiers and two unique key constraints. One unique constraint is defined on the Days column and the other on the MonthLow column. Because DayL and MonthLowL are at the bottoms of their respective hierarchies, they can serve as targets of foreign key references.

Warehouse Builder generates a dimension as a single denormalized table with a set of levels and hierarchies defined on that table. Each level can have any number of attributes.

Creating a Dimension Definition

To create definitions for a dimension, use the New Dimension Wizard. You name the dimension and define a primary key constraint. When you define each column, Oracle recommends setting the constraint to NOT NULL to prevent inconsistent result sets and to maximize the number of query rewrites.

You also define the dimension hierarchy and its levels of aggregation. Table 3–3 provides an example of a dimension table with each level of aggregation, a prefix for each level, and the attributes defined on each level. The levels occur in parent to child order: class is the parent of family and family is the parent of product.

Table 3–3 Example of a Dimension Object

Level	Prefix	Attribute	Data Type	Description
class	cl	class_id	number	Level identifier or key
		class_desc	varchar(20)	Description of product class
family	fa	family_id	number	Level identifier or key
		family_desc	varchar(20)	Description of product family
product	pd	prod_WH	number	Base level or warehouse key
		item_desc	varchar(35)	Description of the product
		product_upc	varchar(11)	Universal product code (natural key)
		item_source	varchar(30)	Supplier for product
		packaging	varchar(20)	Packaging for the product

To create a dimension definition:

1. From the Warehouse Builder navigation tree 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** and select **Create Dimension.**

   Warehouse Builder displays the welcome page for the New Dimension Wizard.
4. Click **Next.**

   The wizard displays the Name page, as shown in Figure 3–24.
5. Type the following:

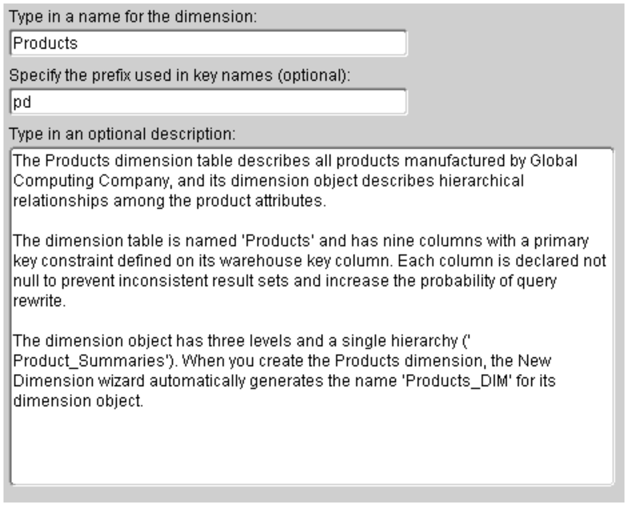
   A name for the dimension.

   A prefix.

   The prefix is used to generate a unique name for the unique key constraint on the base level key column. If the prefix is blank, the dimension name is used.

   A description of the dimension (optional).

Figure 3–24 Name Page



Type in a name for the dimension:

Products

Specify the prefix used in key names (optional):

pd

Type in an optional description:

The Products dimension table describes all products manufactured by Global Computing Company, and its dimension object describes hierarchical relationships among the product attributes.

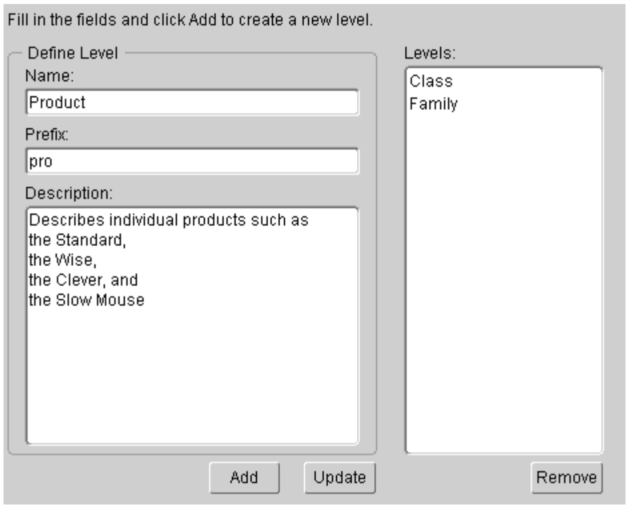
The dimension table is named 'Products' and has nine columns with a primary key constraint defined on its warehouse key column. Each column is declared not null to prevent inconsistent result sets and increase the probability of query rewrite.

The dimension object has three levels and a single hierarchy (' Product_Summaries'). When you create the Products dimension, the New Dimension wizard automatically generates the name 'Products_DIM' for its dimension object.

6. Click **Next.**

The wizard displays the Levels page. Dimensions contain at least one level. You can define a default level to satisfy this requirement and include additional levels as required. Figure 3–25 shows the Levels page.

Figure 3–25 Levels Page



Fill in the fields and click Add to create a new level.

Define Level

Name:

Product

Prefix:

pro

Description:

Describes individual products such as
the Standard,
the Wise,
the Clever, and
the Slow Mouse

Levels:

Class
Family

7. Define levels of aggregation in the dimension. Enter the following:

The name of the level.

A prefix for the level. The default prefix is the name of the level.

A description of the level.

8. Click **Add** to add the level. Continue this process until you have defined each level of aggregation.

Prefixes are useful because they:

Reduce the number of attributes you must enter manually. The wizard generates an ID attribute for each level and assigns it the name levelprefix_ID.

Allow you to reuse attribute names. This is a common practice when you build dimensions for higher levels of aggregation.

Note: The dimension prefix is used to form the names of level unique keys. After a unique key name is generated, changing the dimension prefix does not change the names of existing levels because cube foreign keys may already refer to the generated level. Unique key names generated after you edit the prefix use the new prefix.

---

9. Click **Next.**

   The wizard displays the Level Attributes page. A level can have one or more attributes. The wizard generates an ID attribute for each level.

   The ID attribute for a level identifies the level. The attribute is the key column for the level. This attribute is used in the create dimension statement to define the level, and the defined level is used in the determines clause to specify other columns within that level (dependent columns). See the *Oracle Database SQL Reference and the Oracle Database Data Warehousing Guide* for more information.

10. Select a level of aggregation from the drop-down list.
11. Type a name for the attribute.
12. Select a data type for the attribute from the drop-down list under Data Type. Warehouse Builder supports the following Oracle Database data types:

    CHAR

    DATE

    FLOAT

    NUMBER

    VARCHAR

    VARCHAR2

13. Enter the precision, length, or scale as appropriate for the data type.
14. Type a description of the attribute.
15. Click **Add.**

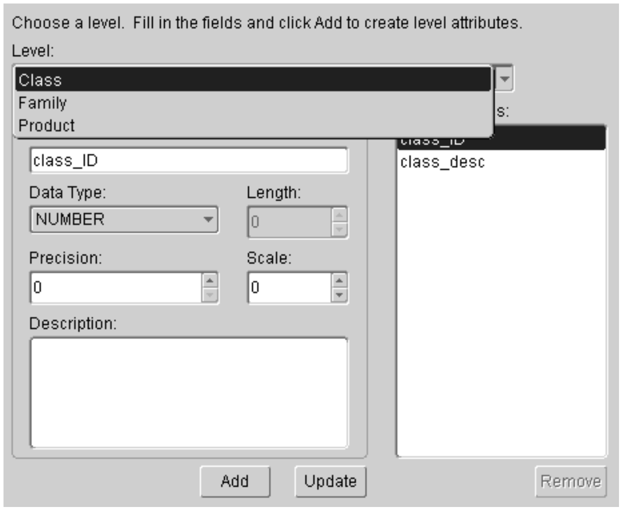
    You can define another attribute for the selected level or select another level and define its attributes. Continue this process until you have defined all the attributes for each level, as shown in Figure 3–26.

    If you want to rename the ID column, select **ID** in the Level Attributes text box.

    Type a new name in the Name text box.

    Click **Update.**

Figure 3–26 Level Attributes Page



Choose a level. Fill in the fields and click Add to create level attributes.

Level:

Class
Family
Product

class_ID

Data Type: NUMBER

Length: 0

Precision: 0

Scale: 0

Description:

class_ID
class_desc

16. Click **Next.**

    The wizard displays the Hierarchies page.

17. **Define the hierarchy**:

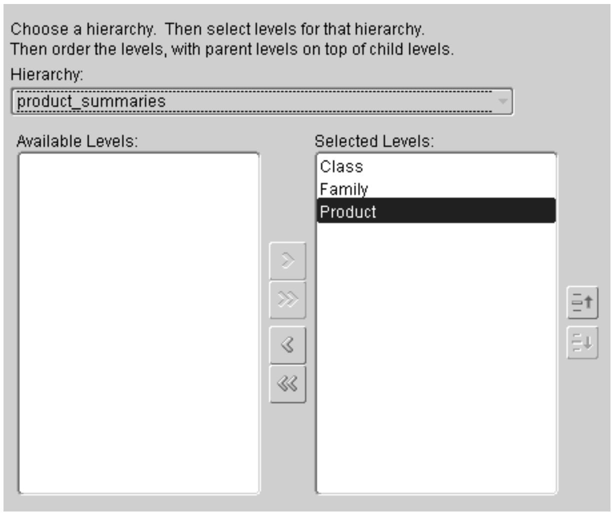
    Type a name and prefix for each hierarchy.

    Type a description of the hierarchy.

18. Click **Next.**

    The wizard displays the Level Relationships page, as shown in Figure 3–27.

Figure 3–27 Level Relationships Page



Choose a hierarchy. Then select levels for that hierarchy.
Then order the levels, with parent levels on top of child levels.

Hierarchy: product_summaries

Available Levels:

Selected Levels:
Class
Family
Product

19. **Define the levels within a hierarchy**:

    Select a hierarchy from the drop-down list.

    Move the names of levels for a selected hierarchy from Available Levels to Selected Levels.

    Arrange the levels so that they show the parent to child order.

20. Click **Next.**

    The wizard displays the Finish page. Verify the description.

21. Click **Finish.**

    The wizard creates a definition for the dimension.

    The wizard generates a unique key (UK) constraint for a dimension table on the ID column that represents the base level of aggregation for the dimension. Dimensional designs often call for a primary key (PK) rather than a UK constraint. After you complete a definition for a dimension, you can change the UK to a PK constraint.

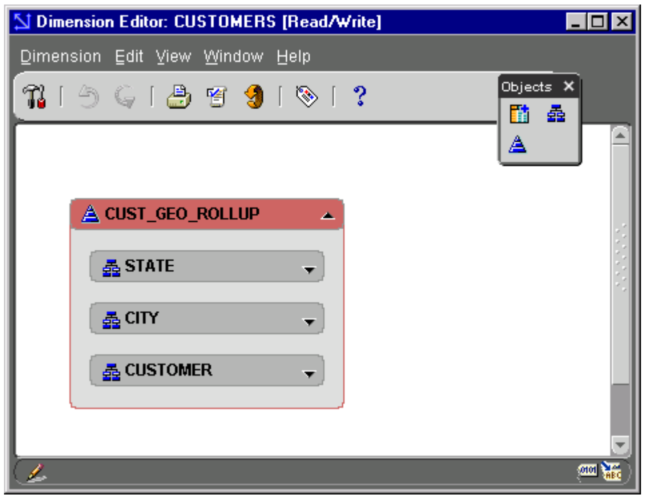
Editing Dimension Definitions

You can edit the definition for a dimension object with the Dimension Editor or by editing entries in the dimension property sheet.

Using the Dimension Editor

To display the Dimension Editor, right-click a dimension in the navigation tree and select **Editor.** Figure 3–28 shows the Dimension Editor.

Figure 3–28 Dimension Editor


Dimension Editor: CUSTOMERS [Read/Write]
Dimension Edit View Window Help
Objects
CUST_GEO_ROLLUP
STATE
CITY
CUSTOMER

The Dimension Editor displays a toolbox and the dimension object.

To add an element to the dimension object, drop an icon from the toolbox onto a dimension element.

To add an attribute to a level:

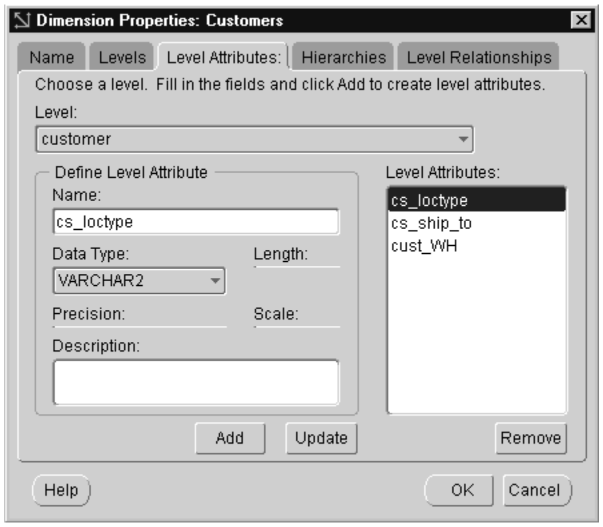
1. Fully expand the Level where you want to add the new attribute.
2. Drop the Attribute icon on the Level.

   Warehouse Builder adds an attribute (attribute1) in the level with the number data type.
3. Enter a name for the attribute.
4. To change the data type, double-click the attribute name.

   The Dimension Editor displays the dimension property sheet.

5. Select the **Level Attributes** tab, as shown in Figure 3–29.
6. Select a data type from the drop-down list.
7. Change the length, scale, or precision depending on the data type selected.
8. Click **Update.**

Figure 3–29 Level Attributes Tab of Dimension Properties Sheet



Choose a level. Fill in the fields and click Add to create level attributes.

Level: customer

Define Level Attribute

Name: cs_loctype

Data Type: VARCHAR2

Length:

Precision:

Scale:

Description:

Level Attributes:
cs_loctype
cs_ship_to
cust_WH

To print the diagram, click the Print icon on the Dimension Editor toolbar.

Using the Property Sheets

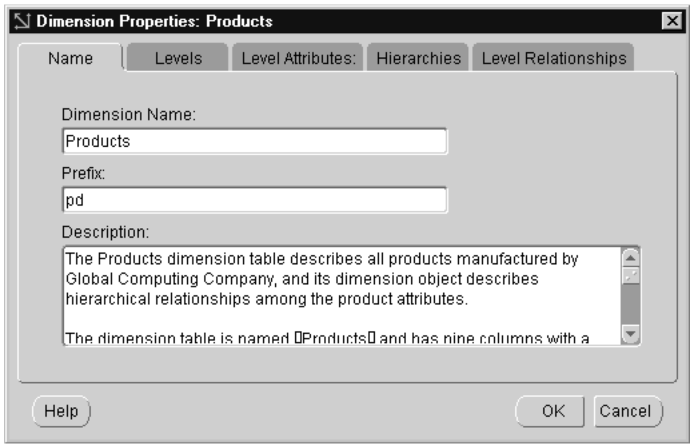
The dimension object and the dimension table both have property sheets. In the dimension object property sheet, you edit the levels and hierarchies. In the dimension table property sheet, you edit the columns and constraints.

To display the dimension object property sheet:

- In the Dimension Editor, from the **Edit** menu, select **Properties** or click the **Properties** icon.
- From the Warehouse Builder navigation tree, right-click the dimension and select **Properties.**

Warehouse Builder displays the property sheet for the dimension object, as shown in Figure 3–30.

Figure 3–30 Dimension Object Properties Sheet



Dimension Name: Products

Prefix: pd

Description: The Products dimension table describes all products manufactured by Global Computing Company, and its dimension object describes hierarchical relationships among the product attributes.

The dimension table is named Products and has nine columns with a

The dimension object property sheet has the following tabs:

- Name
- Levels
- Level Attributes
- Hierarchies
- Level Relationships

To display the dimension table property sheet:

1. Open the Dimension Editor.
2. Select **Table Properties** from the Edit menu.

   Warehouse Builder displays the dimension table property sheet. For information about the table property sheet, see "Editing Table Definitions" on page 3-11.

Using Cubes

Cubes, also known as facts, contain measures and link to one or more dimensions. Most cube measures are additive. Common additive measures include sales, units, and cost.

Cubes are linked to dimension tables over foreign key constraints. These constraints are critical in a data warehousing environment where data integrity is paramount. The constraints enforce referential integrity during the daily operation of the data warehouse.

When dimensions are designed with warehouse keys, the cube row length is usually reduced because warehouse keys are shorter than their natural counterparts. The result is less storage space wasted in the cube.

A typical cube contains:

- A primary key defined on the 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.

When you create a definition for a cube, you must define its measures and its foreign key references. To define a foreign key reference, you include the name of the referenced dimension and its primary key column.

Creating Cube Definitions

This section describes how to create and update a definition for a cube. You create a definition for a cube using the New Cube Wizard, and you update the definition by editing its property sheet. You can also import definitions for tables from another database source or an Oracle Designer Repository.

Use the New Cube Wizard to create definitions for a cube. This information includes details regarding foreign key references, measures, and the data types of all the table columns.

To create a cube definition:

1. From the Warehouse Builder navigation tree 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** and select **Create Cube.**

   Warehouse Builder displays the Welcome page for the New Cube Wizard.
4. Click **Next.**

   The wizard displays the Name page.
5. Enter the following:

   The name of the cube

   A description of the cube (optional)
6. Click **Next.**

   The wizard displays the Define Foreign Keys page.
7. Select the name of a dimension from the Dimension drop-down list.
8. Select the base key level from the Level drop-down list.
9. Select the primary key column constraint defined on the dimension from the Unique Key drop-down list.
10. Click **Add.**

    The wizard inserts the foreign key reference constraint in the text box that lists the foreign keys.
11. Repeat these steps for each foreign key constraint. Select the lowest level of aggregation for the foreign key reference target.

    You can change the name of the generated foreign key by selecting the name and typing over it. The name must be unique within the project.

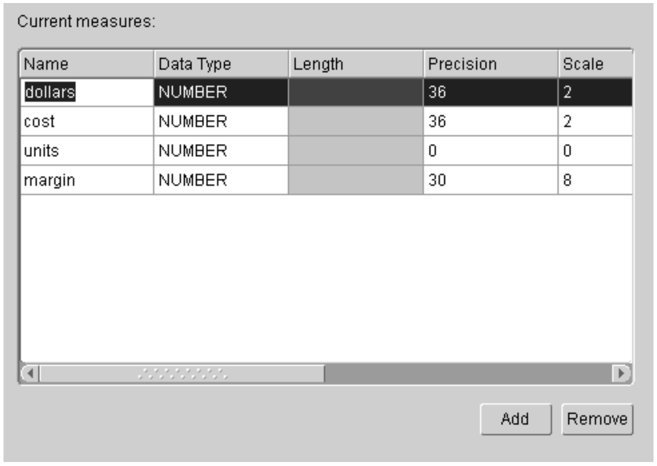
Note:

- The Cube Wizard displays the name for each generated PK constraint on level columns. Only the lowest level PK constraint is an actual physical constraint.
- You cannot modify the name or data type of the foreign key reference columns. You can only do this by editing the definition for the referenced table.
- If you add a column to a PK or UK constraint on a dimension, you must also update the cube foreign key references.

12. Check the box next to **Create segmented unique key from foreign keys.**
13. Click **Next.**

    The wizard displays the Define Measures page, as shown in Figure 3–31.

Figure 3–31 Define Measures Page



Current measures:

Name	Data Type	Length	Precision	Scale
dollars	NUMBER		36	2
cost	NUMBER		36	2
units	NUMBER		0	0
margin	NUMBER		30	8

14. Click **Add.**
15. Type the name of the measure.
16. Select the data type of the measure.
17. Repeat these steps for each measure in the cube.
18. Click **Next.**

    The wizard displays the Finish page. This page summarizes the cube. Click **Back** to modify any of the elements.
19. Click **Finish.**

    The wizard creates a definition for a cube, stores it in the warehouse module, and inserts its name in the navigation tree.

Editing a Cube Definition

A cube object has two property sheets: one for the cube object and another for the table. You can update cube object properties by editing the property sheets. In addition, you can add foreign key references or measures to a cube object using the

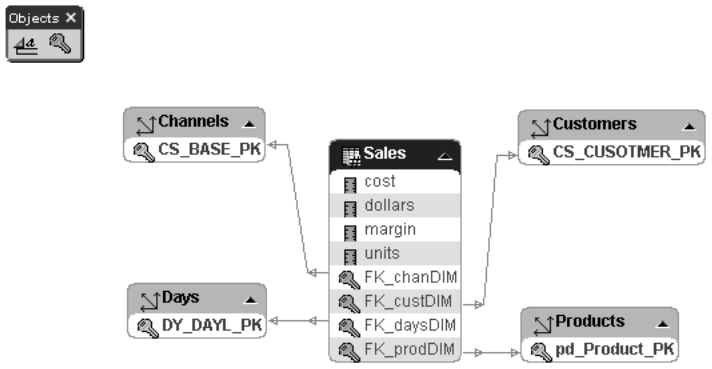
Cube Editor. You can also use the Cube Editor to change cube properties and foreign key relationships with dimensions.

Using the Cube Editor

To open the Cube Editor, right-click a cube name and select **Edit** from the pop-up menu. Warehouse Builder displays the Cube Editor containing a tool palette and a diagram of the cube and the related dimensions.

Figure 3–32 shows the Cube Editor.

Figure 3–32 Cube Editor



To print the diagram, click the printer icon on the Cube Editor toolbar.

To display the cube object property sheet:

- From the **Cube** menu, select **Cube Properties** or click the **Properties** icon.
- Right-click the cube and select **Properties.**

The properties include the object name and description, foreign key references, measures, and attribute sets.

From the Cube Properties sheets, you can:

- Change the name and description of the object.
- Add or Remove a foreign key reference constraint.
- This Foreign Keys sheet shows all the UK constraints defined on a dimension: the base level of aggregation and each higher level of aggregation. Warehouse Builder generates DDL *only for the constraint defined on the base level of aggregation*.
- Change the name of a foreign key reference constraint.
- Add, Remove, or edit a measure (name, data type, and description).

Importing Metadata into Target Modules

Warehouse Builder enables you to import data object definitions into target modules using the Import Metadata Wizard. These definitions assist you in modelling your target system. For Oracle target modules, you can import definitions for tables, views, external tables, sequences, advanced queues, and PL/SQL transformation packages. For other target modules, such as SAP systems, you can import table definitions.

To import object definitions into an Oracle target module:

1. Right-click an Oracle module name and select **Import.**

   The welcome page for the Import Metadata Wizard displays.

2. Click **Next.**

   The Filter Information page displays.

3. Limit the search of the data dictionary in one of the following methods:

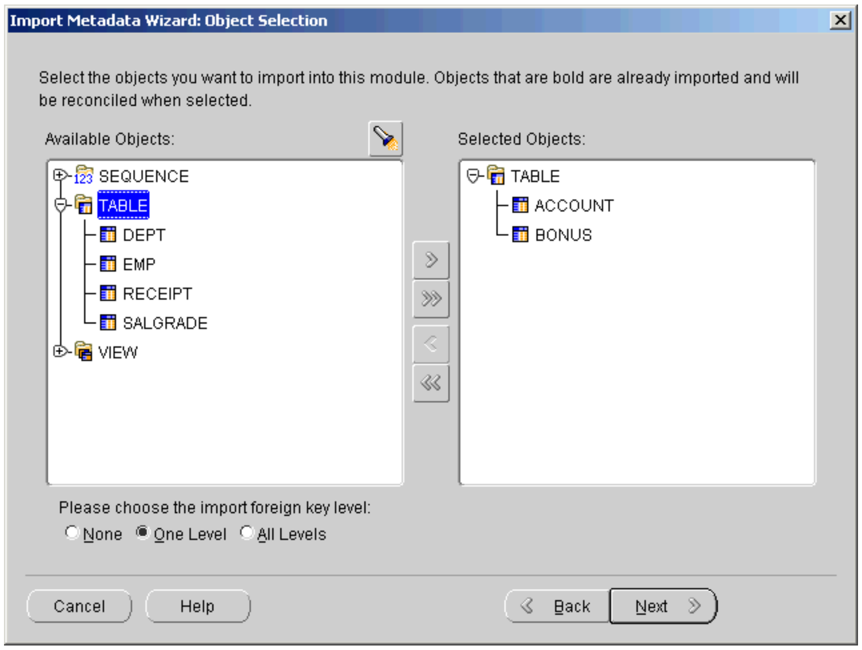
   Select tables, views, external tables, sequences, advanced queues, or PL/SQL transformation packages.

   Type a search pattern. For example, you can type a warehouse project name followed by a % to import objects that begin with that project name. Use % as a wild card match for multiple characters and _ as a wild card match for a single character.

4. Click **Next.**

   Warehouse Builder retrieves names that meet the filter conditions from the data dictionary and displays the Object Selection page as shown in Figure 3–33.

Figure 3–33 Import Metadata Wizard Object Selection Page


Import Metadata Wizard: Object Selection

Select the objects you want to import into this module. Objects that are bold are already imported and will be reconciled when selected.

Available Objects:
- SEQUENCE
- TABLE
  - DEPT
  - EMP
  - RECEIPT
  - SALGRADE
- VIEW

Selected Objects:
- TABLE
  - ACCOUNT
  - BONUS

Please choose the import foreign key level:
None [✓ One Level] All Levels

5. Select items to import from the Available Objects list and click the arrow to move them to the Selected Objects list.

   To move all items to the Selected Objects list, click the double arrow.

   To move an object and the objects it references, select the name of the object and check **One Level.**

   To move a single object and all the objects it references directly or indirectly, select the name of the object and check **All Levels.**

   If you are re-importing definitions, previously imported objects appear in bold.

6. Click **Next.**

The Summary and Import page displays. This page summarizes your selections in a spreadsheet listing the name, type of object, and whether the object will be reconciled or created. Verify the contents of this page and add descriptions for each of the objects.

7. Click **Finish.**

   The Import Results page displays.

8. Click **OK** to accept the changes. Click **Undo** to cancel the import.

   Warehouse Builder stores the definitions in the target module.

After you have created or imported object definitions to model your target system, you can configure these objects for deployment. For more information, see Chapter 5, "Configuring Data Objects". You can also define how to extract, transform, and load the data from your sources to the target systems. For more information, see Chapter 6, "Designing Mappings".

4 Importing Data Definitions

This chapter describes how to create source modules in Warehouse Builder. This chapter also shows you how to import definitions from different data sources into the source modules.

This chapter includes the following topics:

- Creating Source Modules on page 4-1
- About Database Sources on page 4-5
- Using Oracle Designer 6i/9i Sources on page 4-15
- About Flat File Modules on page 4-19
- About Flat File Sources and Targets on page 4-20

Creating Source Modules

In Warehouse Builder, you create a source modules to store definitions from different source systems. You can create source modules for:

- Oracle databases
- Non-Oracle databases
- Flat files
- SAP Applications

Warehouse Builder uses software integrators to read data definitions and extract data from source systems. Table 4–1 summarizes application types and their corresponding integrators.

Table 4–1 Applications and Corresponding Software Integrators

Type of Source	Application Version or System Type	Integrator
Oracle Database	■ Oracle7 Release 7.3 ■ Oracle8 Release 8.0 ■ Oracle8*i* Release 8.1 ■ Oracle9*i* Release 1 (9.0.1) ■ Oracle9*i* Release 9.2	OWB Integrator for Oracle DB & Apps 3.0
Non-Oracle Database (including Sybase, Informix, ODBC)	Oracle Generic Gateway Connectivity	OWB Integrator for Oracle DB & Apps 3.0

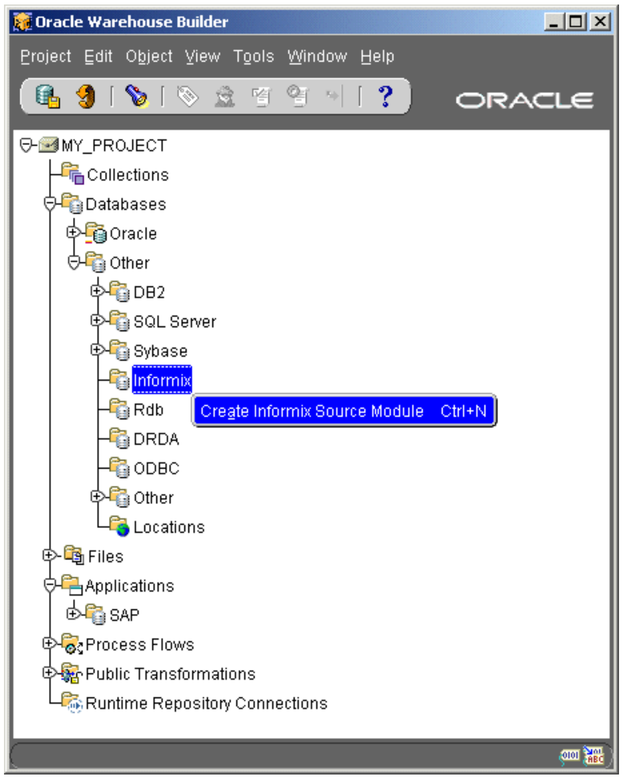
Table 4–1 (Cont.) Applications and Corresponding Software Integrators

Type of Source	Application Version or System Type	Integrator
SAP R/3 3.x, 4.x	SAP	OWB Integrator for SAP Applications 3.0
Flat File System	File System	OWB Integrator for Flat Files

Selecting the Type of Source Module to Create

When you create source modules in Warehouse Builder, the New Module wizard determines the correct integrator to use based upon the source type you select. This section shows you how to choose the type of source module you want to create from the Warehouse Builder navigation tree, as shown in Figure 4–1:

Figure 4–1 Creating Source Modules



- **Oracle Source Module:** Expand the Databases node, right-click the Oracle node and select **Create Oracle Module.** For details, see "Creating a Database Source Module" on page 4-5.
- **SAP Source Module:** Expand the Applications node, then right-click the SAP node and select **Create SAP R/3 Source Module.** See "Creating SAP Module Definitions" on page 21-3.
- **Flat File Module:** Right-click the Files node and select **Create Flat File Module.** See "Creating Flat File Modules" on page 4-19.
- **DB2 Source Module:** Expand the Databases node, then expand the Others node, right-click the DB2 node and select **Create DB2 Source Module.**

- **SQL Server Source Module:** Expand the Databases node, then expand the Others node, right-click the SQL Server node and select **Create SQL Server Source Module.**
- **Sybase Source Module:** Expand the Databases node, then expand the Others node, right-click the Sybase node and select **Create Sybase Source Module.**
- **Informix Source Module:** Expand the Databases node, then expand the Others node, right-click the Informix node and select **Create Informix Source Module.**
- **RDB Source Module:** Expand the Databases node, then expand the Others node, right-click the RDB node and select **Create RDB Source Module.**
- **DRDA Source Module:** Expand the Databases node, then expand the Others node, right-click the DRDA node and select **Create DRDA Source Module.**
- **ODBC Source Module:** Expand the Databases node, then expand the Others node, right-click the ODBC node and select **Create ODBC Source Module.**
- **Other Gateway Source Module:** Expand the Databases node, then expand the Others node, right-click the Other node and select **Create Other Gateway Source Module.** See the section on "Oracle Heterogeneous Services" in the following section.

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 system. 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 database server. You must still create and customize an initialization file for your generic connectivity agent.

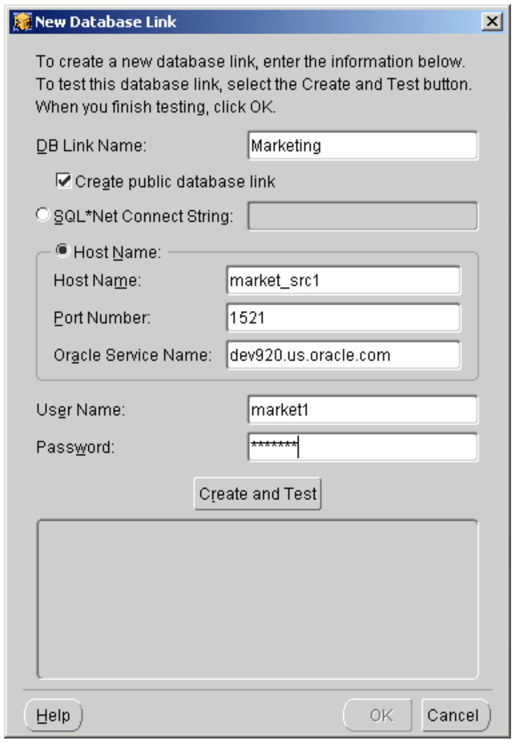
For additional information on distributed processing systems, see *Oracle Database Distributed Database Systems.*

Configuring Connections for Database Sources

When you create a source module for a database source, you create or select a database link in the Warehouse Builder repository that points to the source system. Warehouse Builder uses this link to access the data dictionary of the source.

You can specify the database link from the Connections page of the New Module Wizard. Select an existing database link from the drop-down list and verify the link owner, user name, and connect string or create a new database link using the New Database Link dialog as shown in Figure 4–2.

Figure 4–2 New Database Link Dialog



New Database Link

To create a new database link, enter the information below.
To test this database link, select the Create and Test button.
When you finish testing, click OK.

DB Link Name: Marketing

☑ Create public database link

○ SQL*Net Connect String:

● Host Name:

Host Name: market_src1

Port Number: 1521

Oracle Service Name: dev920.us.oracle.com

User Name: market1

Password: *******

You can create either private or public database links. Private database links can only be used by the user that created them. Public database links are available to all repository users on the same database. By default, if you have multiple users for the same repository, only the owner is given the CREATE PUBLIC DATABASE LINK privilege. If other users want to create public database links they must have the privilege granted.

To create a new database link:

1. Specify a name for the database link.

   A database link name can be a maximum of 128 bytes and can include periods (.) and the "at" sign (@).

2. Check the **Create public database link** check box if you want the database link created in the PUBLIC schema and made available to other users.

   By default, the check box is unchecked.

3. Select either the **SQL*Net Connect String** or **Host Name** radio button and provide the following connection information.

   SQL Connect String: Specify the connect string for the database system as it exists in your TNSNAMES.ORA file. If the system is a non-oracle system, specify this by including `(HS-OK)' within the connect_data clause.

   Host Name: If the connect string does not exist in your TNSNAMES.ORA file, enter the Host Name, Port Number, and Oracle Service Name.

4. Enter a user name and password with access to the database you are connecting to and click **Create and Test.**

   Case sensitive names and passwords need to be double-quoted. For BIS Applications, enter your APPS account user name.

The connection information you provided is tested, and a message will display.

5. Click **OK** when you are done.

   Warehouse Builder stores database link properties in the repository. After you create a database link, you can edit the link information in the module property sheet. For more information on database links and connect strings, see Oracle Database *Distributed Database Systems* and Oracle Database *SQL Reference.*

About Database Sources

This section describes how to create a source module that connects with an application based on a database system. The source module is a container for data definitions imported from the database system.

The following sections contain information about working with Database Source systems:

- Creating a Database Source Module on page 4-5
- Importing Definitions from a Database on page 4-8
- Re-Importing Definitions from an Oracle Database on page 4-10
- Updating Oracle Database Source Definitions on page 4-13

For related information, see the following sections:

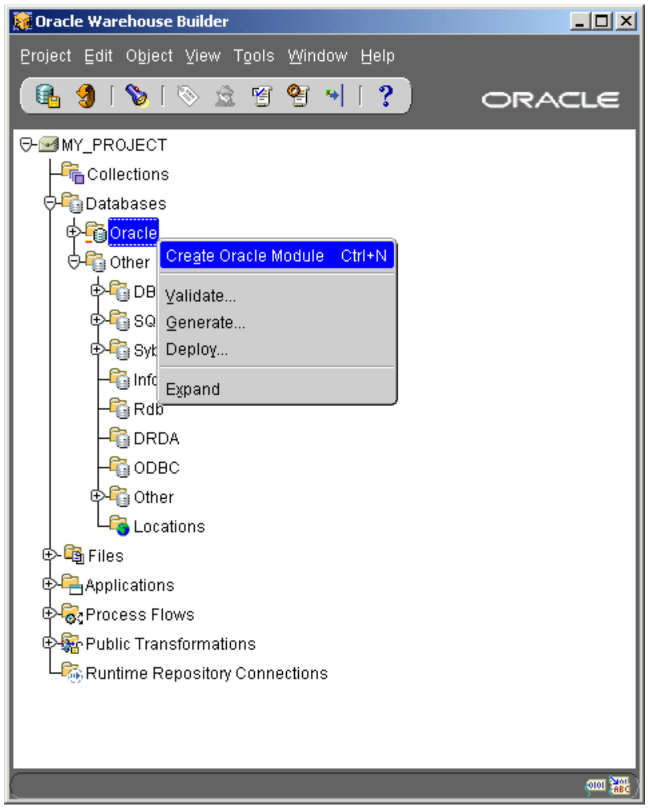
- Configuring Connections for Database Sources on page 4-3
- Using Oracle Designer 6i/9i Sources on page 4-15

Creating a Database Source Module

To create a database source module:

1. From the Warehouse Builder navigation tree, expand the Databases node.
2. To create a source module for Oracle data sources, right-click the Oracle node and select **Create Oracle Module** as shown in Figure 4–3.

Figure 4–3 Creating an Oracle Source Module



The New Module Wizard determines the source type from the navigation tree node from where you launch the wizard. For example, to create an Informix source module, right-click Informix and select **Create Informix Source Module.** The wizard will automatically determine the correct Warehouse Builder integrator to use for this data source.

Warehouse Builder displays the welcome page for the New Module Wizard.

3. Click **Next.**

   The wizard displays the Name page.

4. On the Name page, type:

   Name of the module

   Status of the module

   Specify the status as Development, Quality Assurance, or Production. This status is for descriptive purposes only.

   Data Source as the Module Type

   Description (optional)

5. Click **Next.**

   The wizard displays the Data Source Information page.

6. Select the following:

   Generic Oracle Database Application for the Application

   An Application Version or System Type

The wizard determines the correct integrator based on your selections. For a non-Oracle Database, select Oracle Generic Gateway Connectivity as the Database Version.

7. Click **Next.**

   The wizard displays the Connection Information page.

8. Select Oracle Data Dictionary or Oracle Designer Repository as the metadata source.

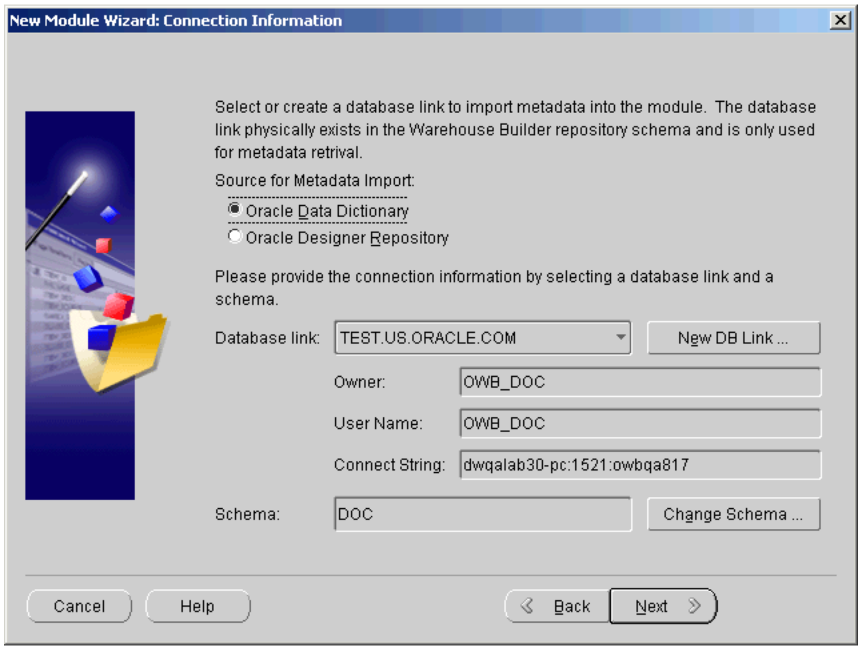
9. Select the name of the database link from the list.

   Click **New DB Link** to create the link if it not does not exist in the Warehouse Builder repository. For more information, see "Configuring Connections for Database Sources" on page 4-3.

   When the database link points to a heterogeneous services agent, the page displays an additional field for the gateway agent if you are importing from non-Oracle databases including systems accessed through ODBC, OLE DB or an Oracle Transparent Gateway.

   The wizard displays the link information and the name of the schema owner as shown in Figure 4–4.

Figure 4–4 Connection Information Page


New Module Wizard: Connection Information

Select or create a database link to import metadata into the module. The database link physically exists in the Warehouse Builder repository schema and is only used for metadata retrival.

Source for Metadata Import:
- [✓ Oracle Data Dictionary]
- Oracle Designer Repository

Please provide the connection information by selecting a database link and a schema.

Database link: TEST.US.ORACLE.COM

Owner: OWB_DOC

User Name: OWB_DOC

Connect String: dwqalab30-pc:1521:owbqa817

Schema: DOC

10. To change the schema name, click **Change Schema.** The wizard displays a list of users.

11. Select a schema and then click **OK.**

    The wizard updates the schema owner name in the Connection page.

12. Click **Next.**

    The Logical Location page displays.

    Locations define information about the database schema or target tool where you will be deploying objects. Locations are specific to a type of module such as Oracle

Database, SAP, or flat file. When you use create a location, a logical definition containing location type and version will be stored. When the location is deployed, the physical connection information is obtained from the Runtime Instance it is deployed to and that information is stored in the Runtime Repository.

13. Use this page to specify a location for the module. Click **New** to create a new location or choose from a list of previously specified locations in the drop-down menu. This step is optional. You can choose to create a location for this module later when you are deploying the object.
14. Click **Next.**

    The Finish page summarizes the information you provided on each of the wizard pages. Check the checkbox if you want to directly start the Import Metadata Wizard.
15. Click **Finish.**

    The wizard creates the source module and inserts its name in the project navigation tree. If you checked the check box, Warehouse Builder starts the Import Metadata Wizard.

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.

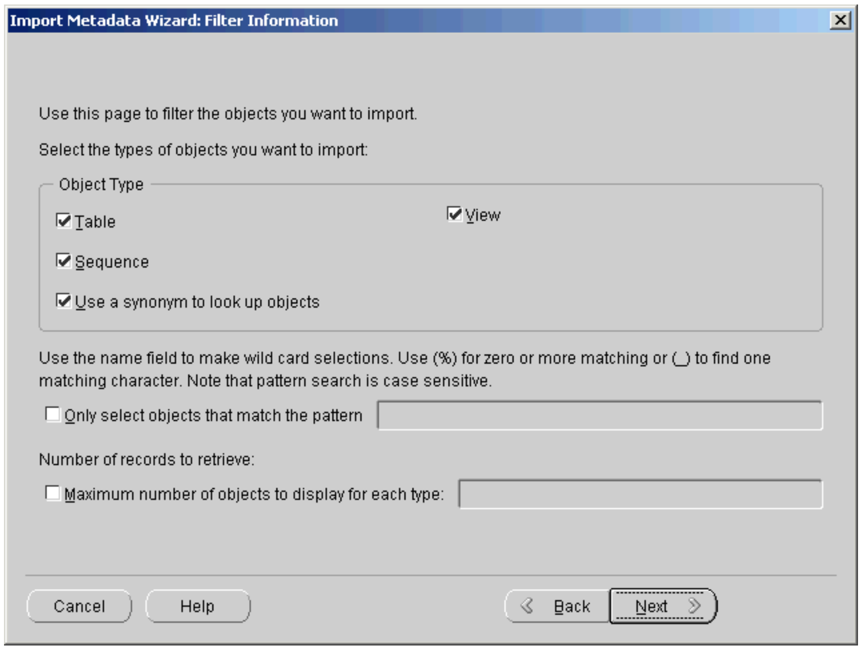
To import definitions from an Oracle Data Dictionary:

1. Right-click a data source module name and select **Import.**

   The welcome page for the Import Metadata Wizard displays.
2. Click **Next.**

   The Filter Information page displays, as shown in Figure 4–5.

Figure 4–5 Filter Information Page


Import Metadata Wizard: Filter Information

Use this page to filter the objects you want to import.

Select the types of objects you want to import:

Object Type

☑ Table ☑ View

☑ Sequence

☑ Use a synonym to look up objects

Use the name field to make wild card selections. Use (%) for zero or more matching or (_) to find one matching character. Note that pattern search is case sensitive.

☐ Only select objects that match the pattern

Number of records to retrieve:

☐ Maximum number of objects to display for each type:

3. Limit the search of the data dictionary in one of the following methods:

   Select tables, views, external tables, sequences, advanced queues, or PL/SQL transformation packages.

   Type a search pattern. For example, you can type a warehouse project name followed by a % to import objects that begin with that project name. Use % as a wild card match for multiple characters and _ as a wild card match for a single character.

4. Click **Next.**

   Warehouse Builder retrieves names that meet the filter conditions from the data dictionary and displays the Object Selection page, as shown in Figure 4–6.

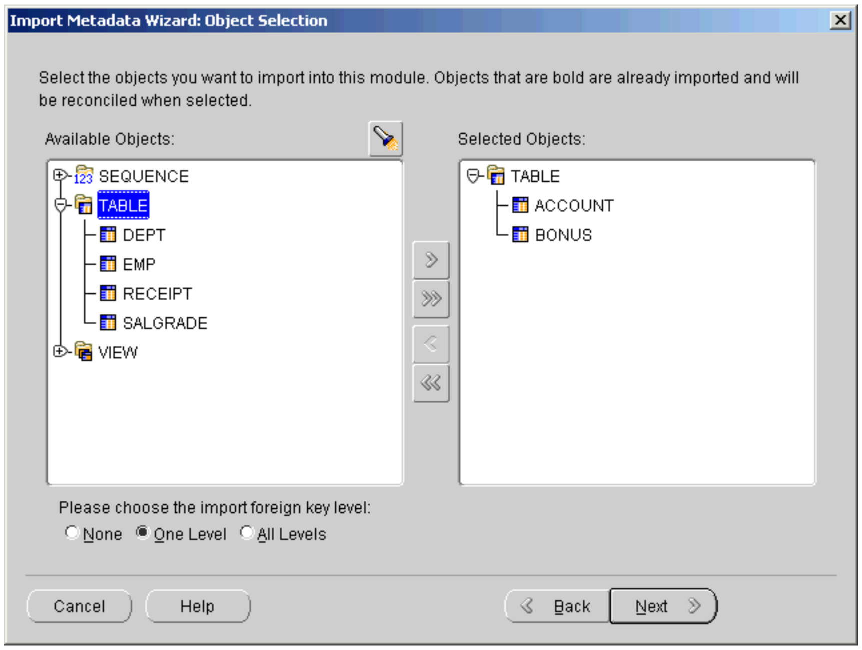
5. Select items to import from the Available Objects list and click the arrow to move them to the Selected Objects list.

   To move all items to the Selected Objects list, click the double arrow.

   To move an object and the objects it references, select the name of the object and check **One Level.**

   To move a single object and all the objects it references directly or indirectly, select the name of the object and check **All Levels.**

Figure 4–6 Import Metadata Wizard Object Selection Page



Import Metadata Wizard: Object Selection

Select the objects you want to import into this module. Objects that are bold are already imported and will be reconciled when selected.

Available Objects:

```
SEQUENCE
TABLE
    DEPT
    EMP
    RECEIPT
    SALGRADE
VIEW
```

Selected Objects:

```
TABLE
    ACCOUNT
    BONUS
```

Please choose the import foreign key level:

None [✓ One Level] All Levels

   If you are re-importing definitions, previously imported objects appear in bold.

6. Click **Next.**

   The Summary and Import page displays. This page summarizes your selections in a spreadsheet listing the name, type of object, and whether the object will be reconciled or created. Verify the contents of this page and add descriptions for each of the objects.

7. Click **Finish.**

   The Import Results page displays.

8. Click **OK** to accept the changes. Click **Undo** to cancel the import.

   Warehouse Builder stores the definitions in the source module.

For related information, see the following sections:

- Re-Importing Definitions from an Oracle Database on page 4-10
- Creating a Database Source Module on page 4-5
- Using Oracle Designer 6i/9i Sources on page 4-15

Re-Importing Definitions from an Oracle Database

Re-importing your source database definitions enables you to import changes made to your source metadata since your previous import. You do not have to remove the original definitions from the repository. Warehouse Builder provides you with options that also enable you to preserve any changes you may have made to the definitions since the previous import. This includes any new objects, foreign keys, relationships, and descriptions you may have created in Warehouse Builder.

To re-import definitions:

1. Right-click a data source module name and select **Import.**

   The welcome page for the Import Metadata Wizard displays.

2. Click **Next.**

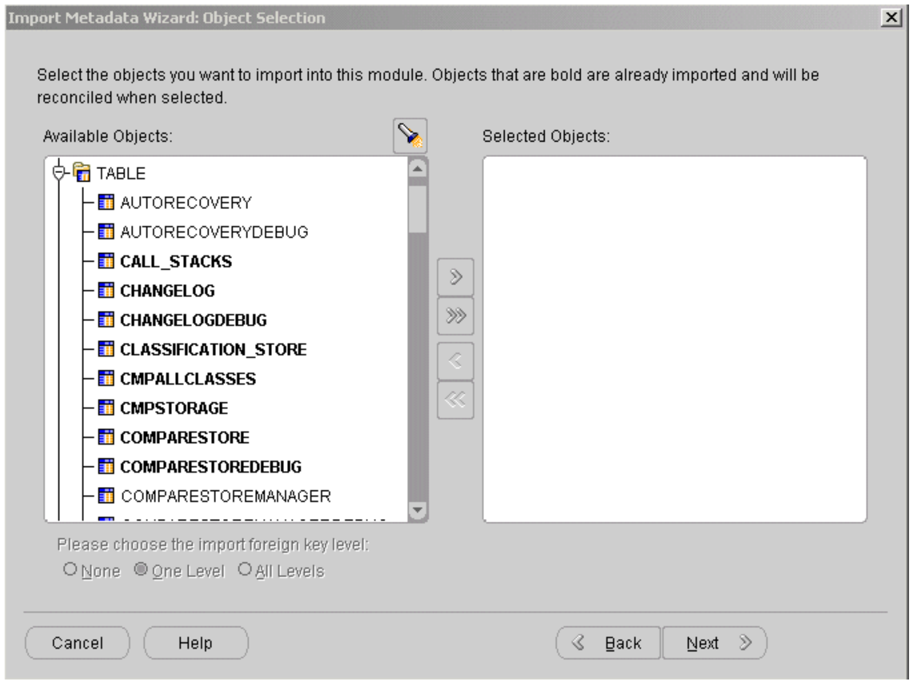
   The Filter Information page displays.

3. Select the object types you want to re-import. You must select the same settings used in the original import to ensure that the same objects are re-imported.

4. Click **Next.**

   The Object Selection page displays. The objects that were originally imported display in bold, as shown in Figure 4–7.

Figure 4–7 Reimport Metadata Object Selection Dialog


Import Metadata Wizard: Object Selection

Select the objects you want to import into this module. Objects that are bold are already imported and will be reconciled when selected.

Available Objects:

- TABLE
  - AUTORECOVERY
  - AUTORECOVERYDEBUG
  - **CALL_STACKS**
  - **CHANGELOG**
  - **CHANGELOGDEBUG**
  - **CLASSIFICATION_STORE**
  - **CMPALLCLASSES**
  - **CMPSTORAGE**
  - **COMPARESTORE**
  - **COMPARESTOREDEBUG**
  - COMPARESTOREMANAGER

Selected Objects:

Please choose the import foreign key level:

None [✓ One Level] All Levels

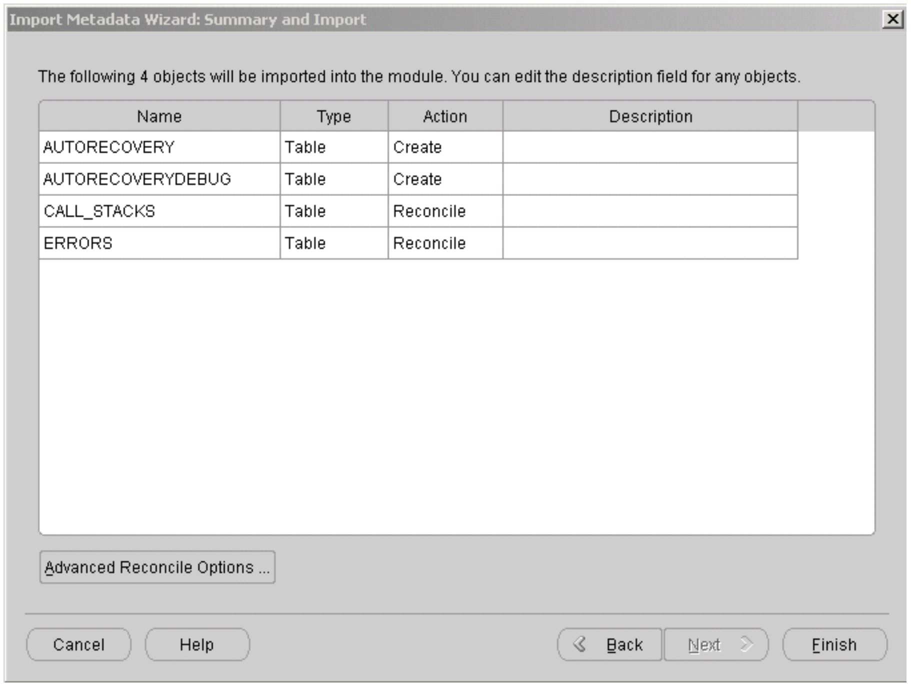
5. Select the objects that you originally imported, and click the arrow to move them to the Selected Objects list.

6. Click **Next.**

   The Summary and Import page displays as shown in Figure 4–8. The Reconcile action is displayed for the objects you are re-importing.

   If the source contains new objects related to the object you are re-importing, the wizard requires that you import the new objects at the same time. The Create action displays for these objects.

Figure 4–8 Summary and Import Page Showing Reconcile Action



Import Metadata Wizard: Summary and Import

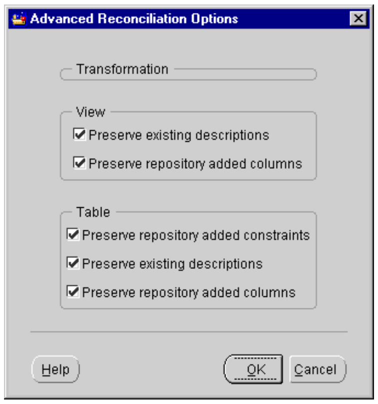
The following 4 objects will be imported into the module. You can edit the description field for any objects.

Name	Type	Action	Description
AUTORECOVERY	Table	Create	
AUTORECOVERYDEBUG	Table	Create	
CALL_STACKS	Table	Reconcile	
ERRORS	Table	Reconcile	

7. Click **Advanced Reconcile Options** to select advanced reconciliation options.

   The Advanced Reconciliation Options dialog displays, as shown in Figure 4–9.

Figure 4–9 Advanced Reconciliation Options Dialog



Advanced Reconciliation Options

Transformation

View
- [✓] Preserve existing descriptions
- [✓] Preserve repository added columns

Table
- [✓] Preserve repository added constraints
- [✓] Preserve existing descriptions
- [✓] Preserve repository added columns

This dialog enables you to preserve any edits and additions made to the object definitions in the Warehouse Builder repository.

Select these options for reconciling views:

Preserve existing descriptions: The descriptions stored in the repository are preserved.

Preserve repository added columns: The columns you added in Warehouse Builder are preserved.

Select these options for reconciling tables:

Preserve repository added constraints: The constraints you added to the table in Warehouse Builder are preserved.

Preserve existing descriptions: The descriptions stored the repository are preserved.

Preserve repository added columns: The columns you added in Warehouse Builder are preserved.

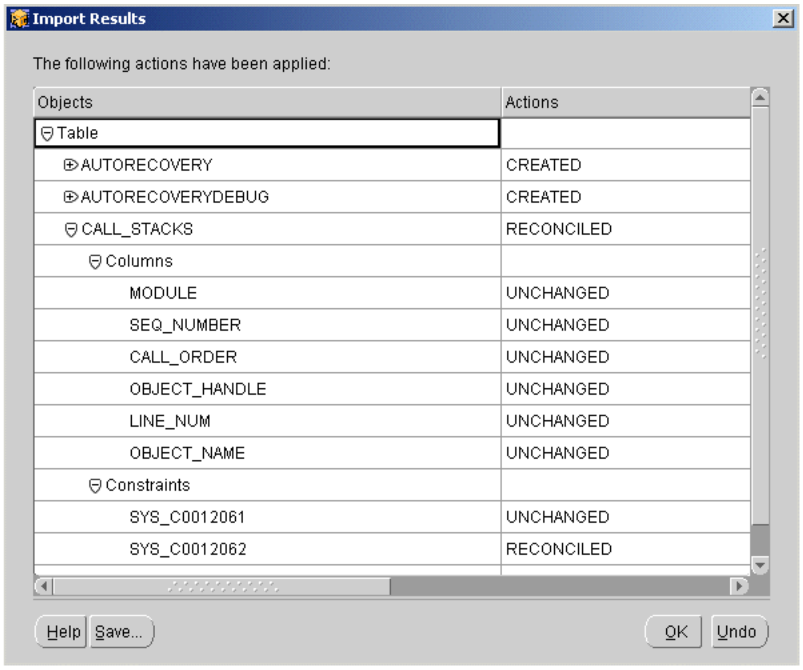
By default, all options are checked. Clear boxes to have these repository objects replaced and not preserved.

For example, after importing tables or views for the first time, you manually add descriptions to the table or view definitions. If you want to make sure that these descriptions are not overwritten while reimporting the table or view definitions, you must select the Preserve Existing Definitions option. This ensures that your descriptions are not overwritten.

8. Click **OK** after selecting your options.
9. Click **Finish.**

   Warehouse Builder reconciles and creates objects. When this is complete, the Import Results dialog displays, as shown in Figure 4–10.

Figure 4–10 Import Results Dialog



The following actions have been applied:

Objects	Actions
Table	
AUTORECOVERY	CREATED
AUTORECOVERYDEBUG	CREATED
CALL_STACKS	RECONCILED
Columns	
MODULE	UNCHANGED
SEQ_NUMBER	UNCHANGED
CALL_ORDER	UNCHANGED
OBJECT_HANDLE	UNCHANGED
LINE_NUM	UNCHANGED
OBJECT_NAME	UNCHANGED
Constraints	
SYS_C0012061	UNCHANGED
SYS_C0012062	RECONCILED

The report lists the actions performed by Warehouse Builder for each object.

Click **Save** to save the report. You should use a naming convention that is specific to the re-import.

10. Click **OK** to proceed.

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

Updating Oracle Database Source Definitions

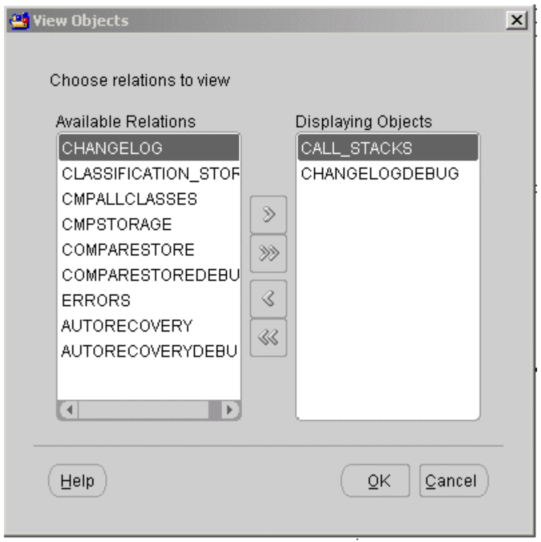
The Oracle Source Module Editor enables you to view and print schema diagrams of objects in the Oracle Source Module and run Lineage and Impact Analysis reports on them.

To display the Oracle Source Module Editor:

1. Double-click the name of the source module.

   The View Objects dialog displays as shown in Figure 4–11.

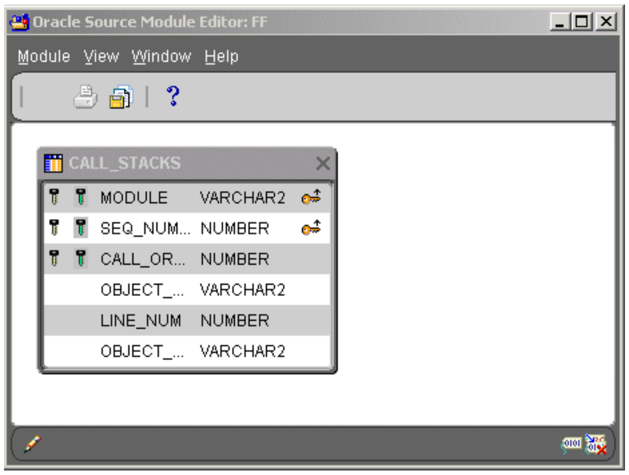
Figure 4–11 View Objects Dialog



2. Choose the objects you want to graphically display on the Oracle Source Module Editor canvas.
3. Click **OK.**

   Warehouse Builder displays the Oracle Source Module Editor, as shown in Figure 4–12.

Figure 4–12 Source Module Editor



MODULE	VARCHAR2
SEQ_NUM...	NUMBER
CALL_OR...	NUMBER
OBJECT_...	VARCHAR2
LINE_NUM	NUMBER
OBJECT_...	VARCHAR2

To print a diagram of a source definition, click the Print icon on the editor toolbar.

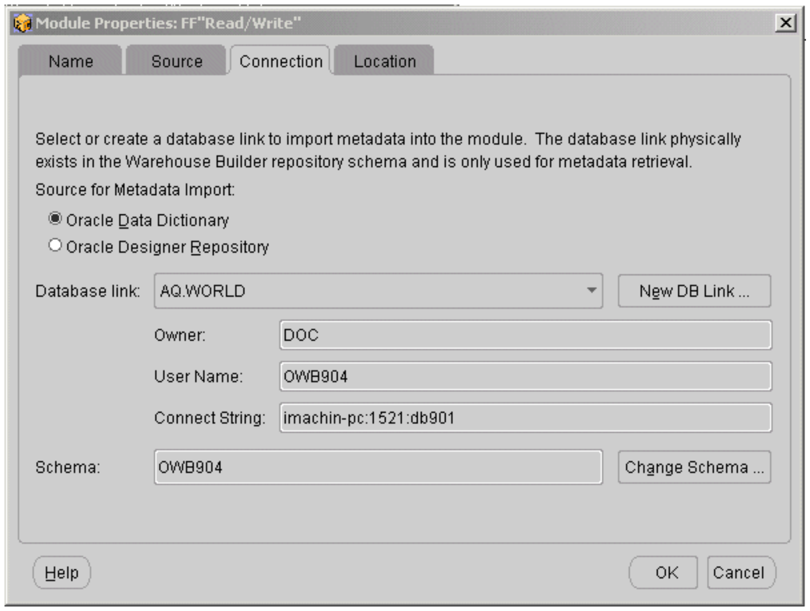
Updating a Source Definition

You can update a the properties of a source definition by editing its Property Sheet. To display the property sheet, right-click the module name from the navigation tree and select **Properties** from the pop-up menu.

Updating the Connection

Select the Connection tab from the Module Properties window to update the connection information for a data source. You can select another database link from the drop-down list, as shown in Figure 4–13.

Figure 4–13 Connection Tab of the Module Properties Sheet



Select or create a database link to import metadata into the module. The database link physically exists in the Warehouse Builder repository schema and is only used for metadata retrieval.

Source for Metadata Import:

(●) Oracle Data Dictionary
( ) Oracle Designer Repository

Database link: AQ.WORLD

Owner: DOC

User Name: OWB904

Connect String: imachin-pc:1521:db901

Schema: OWB904

When you change the connection information, Warehouse Builder displays a warning message that you may compromise the existing definitions in the source module. To change the connection, click **OK.**

Updating the Location

Select the Location tab from the Module Properties window to update the location of a data source. If you have defined multiple locations for the source module, you can select a different location from the drop-down list.

For more on Locations, see "Defining Runtime Repository Connections" on page 13-2.

Using Oracle Designer 6*i*/9*i* Sources

In Warehouse Builder, you can create a source module that connects with an Oracle Designer repository. When the definitions for an application are stored and managed in an Oracle Designer repository, you can reduce the amount of time you need to connect with the application.

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 Application Systems within workareas. An Application System contains definitions for namespace and ownership of objects and enables you to view objects even though they are owned by a different user. Because Designer 6*i*/9*i*

Application Systems are controlled by workareas, they have version control. See the Designer 6*i*/9*i* documentation for more information about workareas and application systems.

All visible objects of a workarea or an Application System in Designer 6*i*/9*i* are available for use as data sources in Warehouse Builder. To select Designer 6*i*/9*i* objects as Warehouse Builder sources:

- Specify a workarea or,
- Specify the application system in the workarea

The New Module Wizard detects the Designer version available in a database link. If it finds Designer 6*i*/9*i*, the Connection Information page changes to show the Workarea and the Application System fields along with a change button for each.

After you click **Change,** the New Module Wizard displays a selection list from which you choose either a workarea or an application system. 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 application system. The list displays objects owned by the specified application system and other objects shared by the specified application system, but not owned by it.

To import definitions from a Designer 6*i*/9*i* source, you must follow the steps outlined for importing definitions from database sources on page 4-8.

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

To create a Designer6i/9i source module:

1. From the Warehouse Builder navigation tree, expand the Databases node.
2. Right-click the Oracle node and select **Create Oracle Module.**

   Warehouse Builder displays the welcome page for the New Module Wizard.
3. Click **Next.**

   The New Module Wizard displays the Name page.
4. On the Name page, enter:

   Name of the module

   Status of the module

   Specify the status as Development, Quality Assurance, or Production. This status is for descriptive purposes only.

   Data Source as the Module Type

   Description (optional)
5. Click **Next.**

   The wizard displays the Data Source Information page. The wizard determines the correct integrator based on the navigation tree node from where you launch the wizard.
6. Click **Next.**

The wizard displays the Connection Information page, as shown in Figure 4–14.

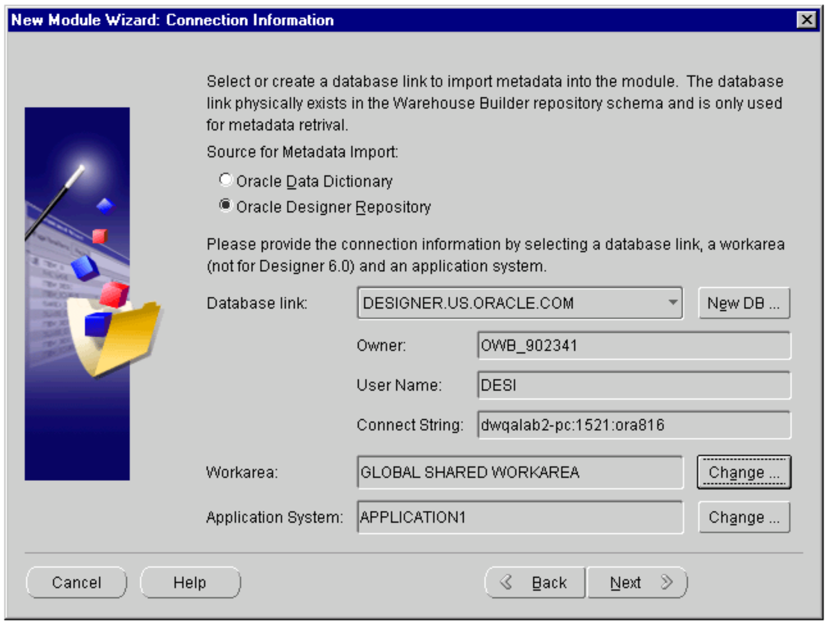
7. Select **Oracle Designer Repository** as the source for metadata import.

8. Select the name of the database link to a Designer 6*i* repository from the list.

   Click **New DB Link** to create the link if it not does not exist in the Warehouse Builder repository. For more information, see "Configuring Connections for Database Sources" on page 4-3.

   When you connect to a Designer 6*i*/9*i* repository, you must connect as the Designer Repository owner.

Figure 4–14 Connection Information Page



New Module Wizard: Connection Information

Select or create a database link to import metadata into the module. The database link physically exists in the Warehouse Builder repository schema and is only used for metadata retrival.

Source for Metadata Import:

○ Oracle Data Dictionary
● Oracle Designer Repository

Please provide the connection information by selecting a database link, a workarea (not for Designer 6.0) and an application system.

Database link: DESIGNER.US.ORACLE.COM

Owner: OWB_902341

User Name: DESI

Connect String: dwqalab2-pc:1521:ora816

Workarea: GLOBAL SHARED WORKAREA

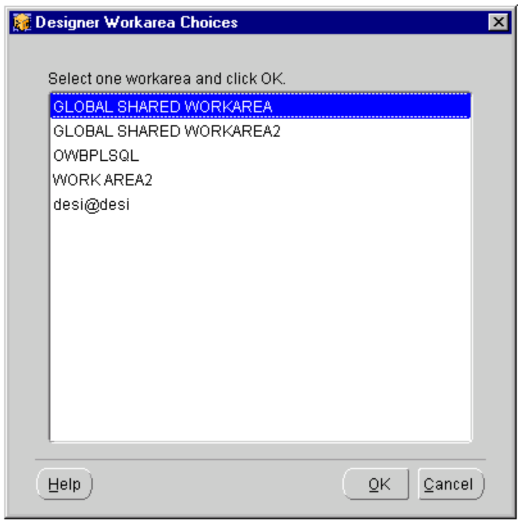
Application System: APPLICATION1

If the New Module Wizard detects a Designer 6*i* repository, the Workarea field, Application System field, and Change button are enabled.

You must specify a workarea before Warehouse Builder can select objects in the Designer 6*i* repository.

9. To specify a Workarea, click **Change** and choose one from the selection list, as shown in Figure 4–15.

Figure 4–15 Workarea Selection List

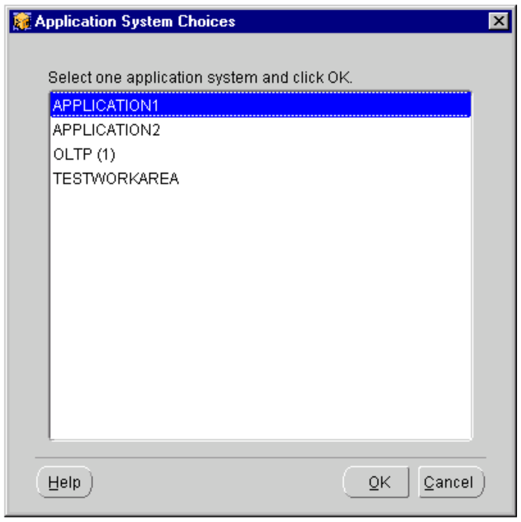


Select one workarea and click OK.

GLOBAL SHARED WORKAREA
GLOBAL SHARED WORKAREA2
OWBPLSQL
WORK AREA2
desi@desi

10. To specify the Application System, click **Change** and choose one from the selection list, as shown in Figure 4–16.

Figure 4–16 Application Systems Selection List



Select one application system and click OK.

APPLICATION1
APPLICATION2
OLTP (1)
TESTWORKAREA

11. Click **Next.**

    The wizard displays the Finish page. This page summarizes the information you entered on each of the wizard pages. Verify the information.

12. Click **Finish.**

    The wizard creates the source module and inserts its name in the project navigation tree.

    To import definitions into this source module from a Designer 6*i* source, you must follow the steps outlined for importing definitions from database sources on page 4-8.

For related information, see the following sections:

- Importing Definitions from a Database on page 4-8

- Re-Importing Definitions from an Oracle Database on page 4-10

About Flat File Modules

A project may need to extract data from or write data to flat files. You can access the flat file directly or by using the External Table operator. This section focuses on the basics of creating a flat file module. This information also serves as the basis for creating an external table from Warehouse Builder. For additional information on external tables, see "Using External Tables" on page 3-18.

When you use a flat file, understanding both flat files and external tables will help you determine which feature to use. 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 that you can apply to an external table. Refer to "External Tables versus Flat File Operators" on page 3-19 for more information.

If you are accessing the data files directly, and if the Warehouse Builder client and the data files reside on different types of operating systems, contact your system administrator to establish the required connectivity through NFS or other network protocol. If the Warehouse Builder client and data files reside on a Windows operating system, store the data files on any drive locally accessed from the Warehouse Builder client machine.

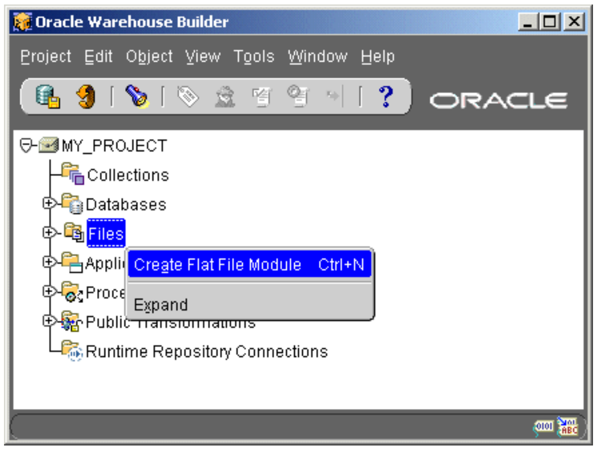
Creating Flat File Modules

Once you create a flat file module, you can import flat file definitions into Warehouse Builder. For more information on importing flat files into Warehouse Builder, see "About Flat File Sources and Targets" on page 4-20.

To create a flat file module:

1. Right-click the **Files** node in the Warehouse Builder Console and select **Create Flat File Module**, as shown in Figure 4–17.

Figure 4–17 Creating a Flat File Module



Warehouse Builder displays the welcome page for the New Module Wizard.

2. Click **Next.**

   The New Module Wizard displays the Name page.

3. On the Name page, type a name for the module and indicate the status of the module. Specify the status as Development, Quality Assurance, or Production. This status is for descriptive purposes only. Optionally provide a description for the module.
4. Click **Next.**

   The wizard displays the Connection Information page.
5. Enter the fully qualified directory, including the drive letter, if appropriate, that contains the file.
6. Click **Next.**

   The Logical Location page displays.

   Locations define information about the database schema or target tool where you will be deploying objects. Locations are specific to a type of module such as Oracle Database, SAP, or flat file. When you create a location, a logical definition containing location type and version will be stored. When the location is deployed, the physical connection information is obtained from the Runtime Instance it is deployed to and that information is stored in the Runtime Repository.
7. Use this page to specify a location for the module. Click **New** to create a new location or choose from a list of previously specified locations in the drop-down menu. This step is optional. You can choose to create a location for this module later when you are deploying the object.
8. Click **Next.**

   The Finish page summarizes the information you provided on each of the wizard pages. Check the checkbox if you want to immediately start the Import Metadata Wizard.
9. Click **Finish.**

   The wizard creates the flat file module and inserts its name in the project navigation tree. If you checked the checkbox, Warehouse Builder starts the Import Metadata Wizard.

About Flat File Sources and Targets

Before you can create a definition for a flat file, you must first create a file module and a file location. To create a definition for a flat file, use the following wizards:

- **The Import Metadata Wizard.** Use the Import Metadata Wizard to select the flat files you wish to import.
- **The Flat File Sample Wizard.** When you finish selecting flat files in the Import Metadata Wizard, the Import Metadata Wizard launches the Flat File Sample Wizard. Use the Flat File Sample Wizard to view a sample of the flat file and define record organization and file properties.

You can update a file definition after you import it into Warehouse Builder. You can also use data from a flat file as a source or a target after you import its structure into Warehouse Builder.

This section includes:

- Using the Import Metadata Wizard Flat Files on page 4-21
- Using the Flat File Sample Wizard on page 4-23

For related information, see the following sections:

- Creating Flat File Modules on page 4-19
- Updating a File Definition on page 4-34
- Mapping Flat File Operator on page 8-22
- Reconciling Operators and Repository Objects on page 6-28

Using the Import Metadata Wizard Flat Files

The following section describes how to create a definition for a flat file using the Import Metadata Wizard.

To import flat files:

1. Right-click a file module and select **Import.**

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

2. Click **Next.**

   The wizard displays the Filter Information page. This page gives you the option to filter file names.

   All Data Files: This option returns all the data files available for the directory you specified for the flat file module.

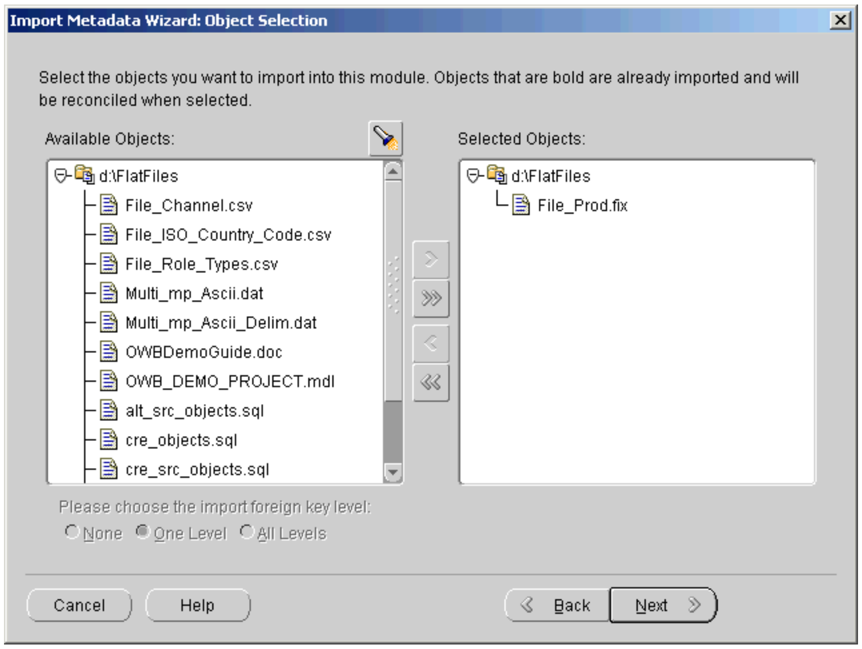
   Data files matching this pattern: Use this option to select only data files that match the pattern you type. For example, if you select this option and enter (*.dat), only files with .dat file extensions will be displayed on the next wizard page. If you type % as part of a filter string, it is interpreted as a wild card match for multiple characters. If you type '_' as part of a filter string, it is interpreted as a wild card match for a single character.

3. Click **Next.**

   The wizard displays the Object Selection page, as shown in Figure 4–18.

   Because Warehouse Builder does not provide inbound reconciliation for flat files, the available objects will never appear in bold like other objects when they are reimported. When you reimport flat files, you always need to resample the flat file objects again

Figure 4–18 Object Selection Page


Import Metadata Wizard: Object Selection

Select the objects you want to import into this module. Objects that are bold are already imported and will be reconciled when selected.

Available Objects:
- d:\FlatFiles
  - File_Channel.csv
  - File_ISO_Country_Code.csv
  - File_Role_Types.csv
  - Multi_mp_Ascii.dat
  - Multi_mp_Ascii_Delim.dat
  - OWBDemoGuide.doc
  - OWB_DEMO_PROJECT.mdl
  - alt_src_objects.sql
  - cre_objects.sql
  - cre_src_objects.sql

Selected Objects:
- d:\FlatFiles
  - File_Prod.fix

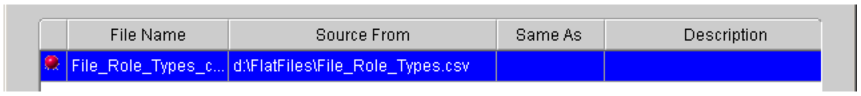
Please choose the import foreign key level:
None [✓ One Level] All Levels

4. Move the name of the files to be described from Available Objects on the left to the Selected Objects window pane on the right.
5. Click **Next.**

   The wizard displays the Summary and Import page. The left-most column of this page contains a status ball which indicates if Warehouse Builder has the metadata for the file, as shown in Figure 4–19.

   If the status ball is red, Warehouse Builder does not have the metadata. Proceed to the next step.
6. For each file on the Summary and Import page, either click **Sample** or select a file with a matching format from the **Same As** list box. If the format for a file matches that of another file on the Summary and Import page, you can select the file name from the **Same As** list box.

Figure 4–19 Summary and Import Page Showing File Status



	File Name	Source From	Same As	Description
●	File_Role_Types_c...	d:\FlatFiles\File_Role_Types.csv		

7. Select a file that has a red status ball and click **Sample** at the bottom of the Summary and Import page.

   The wizard displays the welcome page for the Flat File Sample Wizard. Complete the Flat File Sample Wizard. For more information on the Sample Wizard, see Using the Flat File Sample Wizard.
8. After you complete the Flat File Sample Wizard for one file, Warehouse Builder returns to the Summary and Import page. The file you sampled is now displayed with a green status ball.

9. If the status ball is green for all the files you want to import, click **Finish**. Warehouse Builder creates a definition for file, stores the definition in the source module, and inserts the format name in the source module navigation tree.

Using the Flat File Sample Wizard

Each time you use the Import Metadata Wizard to sample data from existing flat files, the Import Metadata Wizard launches the Flat File Sample Wizard. Use the Flat File Sample Wizard as an aid in defining metadata for flat files. The Flat File Sample Wizard stores the metadata you define in the Warehouse Builder repository.

To utilities the sampled flat file object in a mapping, you must run the Create External Table Wizard and create an external table based on this flat file. This new external table is then available for use in mappings as the external table operator. For information on using external tables versus using flat file operators, see "External Tables versus Flat File Operators" on page 3-19.

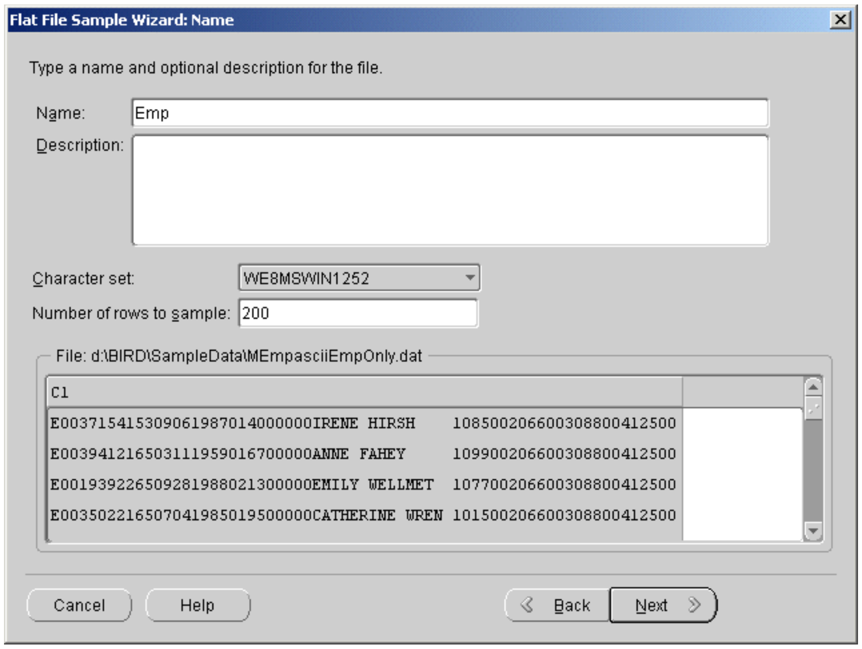
The Flat File Sample Wizard guides you in completing the following steps:

- Describing the Flat File
- Selecting the Record Organization
- Selecting the File Layout
- Selecting the File Format
- Selecting Record Types (Multiple Record Type Files Only)
- Specifying Field Lengths (Fixed-Length Files Only)
- Specifying Field Properties

Describing the Flat File

Use the Name page shown in Figure 4–20 to describe the flat file you are sampling.

Figure 4–20 Name Page for the Flat File Sample Wizard



Type a name and optional description for the file.

Name: Emp

Description:

Character set: WE8MSWIN1252

Number of rows to sample: 200

File: d:\BIRD\SampleData\MEmpasciiEmpOnly.dat

```
C1
E00371541530906198701400000OIRENE HIRSH    1085002066003088004125OO
E003941216503111959016700000ANNE FAHEY     109900206600308800412500
E001939226509281988021300000EMILY WELLMET  107700206600308800412500
E003502216507041985019500000CATHERINE WREN 101500206600308800412500
```

- **Name:** This is the name by which Warehouse Builder will refer to the file after it is imported. By default, the wizard creates a name based on the name of the source file. You can change the file name on this page.

  If you rename the file, do not include a space or any punctuation in the name. You can include an underscore. You can use upper and lower case letters. Do not start the name with a digit. Do not use a Warehouse Builder reserved word. For a list of reserved words, see Appendix B, "Reserved Words".

- **Description:** You can type in an optional description for the file.

- **Character set:** Character sets determine what languages can be represented in database objects and files. In the Warehouse Builder client, the default Globalization Support or National Language Support (NLS) character set matches the character set defined for the machine hosting Warehouse Builder. If the character set differs from that of the source file, the data sample might appear unintelligible. You can display the data sample in the character set native to the source by selecting it from the drop-down list. For complete information on NLS character sets, see the Oracle Database *Globalization Support Guide.*

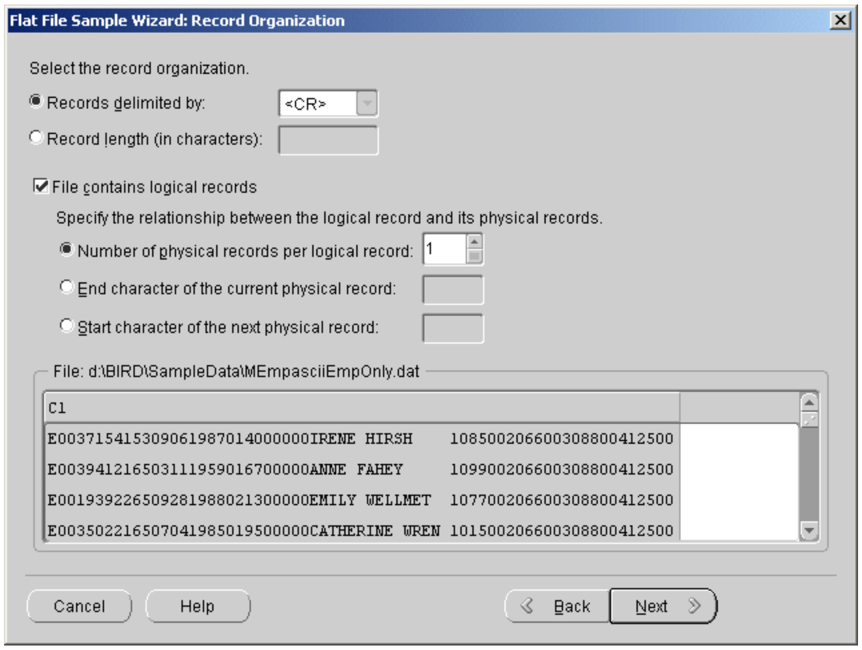
- **Number of rows to sample:** You can indicate the number of rows for the wizard to sample from the data file. By default, the wizard samples the first 200 rows. To determine an optimum value for this field, see "Example: Flat File with Multiple Record Types" on page 4-27.

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

Selecting the Record Organization

Use the Record Organization page shown in Figure 4–21 to indicate how records are organized in the file you are sampling.

Figure 4–21 Record Organization Page for the Flat File Sample Wizard



Flat File Sample Wizard: Record Organization

Select the record organization.

● Records delimited by: <CR>

○ Record length (in characters):

✓ File contains logical records

Specify the relationship between the logical record and its physical records.

● Number of physical records per logical record: 1

○ End character of the current physical record:

○ Start character of the next physical record:

File: d:\BIRD\SampleData\MEmpasciiEmpOnly.dat

```
C1
E00371541530906198701400000OIRENE HIRSH    108500206600308800412500
E00394121650311195901670000OANNE FAHEY     109900206600308800412500
E00193922650928198802130000OEMILY WELLMET  107700206600308800412500
E00350221650704198501950000OCATHERINE WREN 101500206600308800412500
```

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
  ```

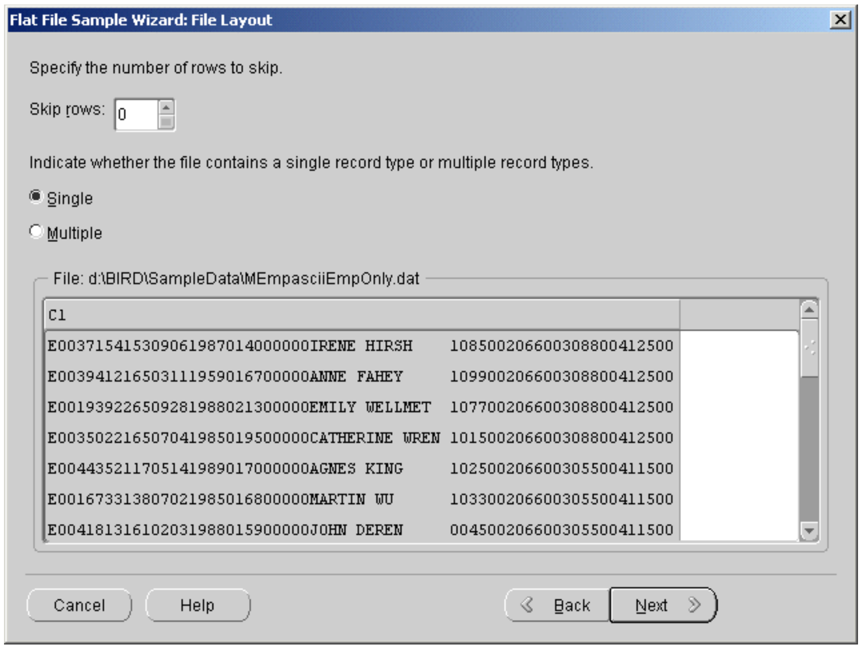
  More than two records can be joined with this technique. The following example shows four physical records for each logical record using continuation at beginning.

```
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 Layout

Use the File Layout page shown in Figure 4–22 to specify the number of rows to skip and to select between a single record type versus multiple record types.

Figure 4–22 File Layout Page for the Flat File Sample Wizard



Specify the number of rows to skip.

Skip rows: 0

Indicate whether the file contains a single record type or multiple record types.

[✓ Single]

Multiple

File: d:\BIRD\SampleData\MEmpasciiEmpOnly.dat

```
C1
E00371541530906198701400000OIRENE HIRSH    108500206600308800412500
```

Indicate the number of records to skip in **Skip rows.**This is useful for skipping over unwanted header information. If one of the records includes field names, skip the preceding header records so that the record containing field names is first in the file. Later in the wizard, on the Field Properties page, you can instruct the wizard to use that record for field names if you are defining a single record file type.

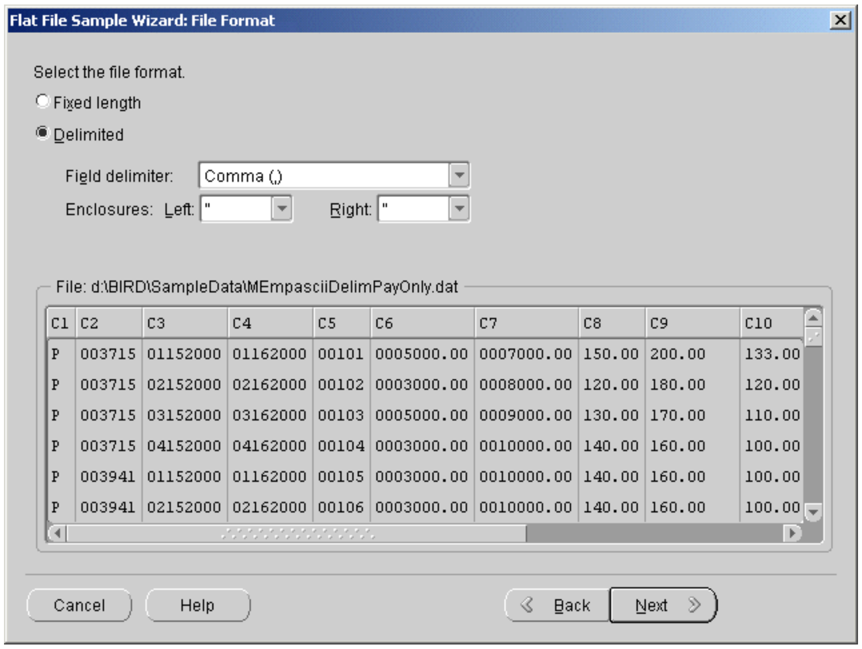
Indicate whether the file contains a single record type or multiple record types.

- If the file contains only one type of record, select **Single.**
- If the file contains more than one record type, select **Multiple.** Later in the wizard you can instruct the wizard to scan the file for the record types. For more information on multiple record types, see "Selecting Record Types (Multiple Record Type Files Only)" on page 4-27.

Selecting the File Format

Use the File Format page shown in Figure 4–23 to select between **Fixed Length** and **Delimited** formats for the file.

Figure 4–23   File Format Page for the Flat File Sample Wizard



Select the file format.

( ) Fixed length
[✓ Delimited]

Field delimiter: Comma (,)

Enclosures: Left: " Right: "

File: d:\BIRD\SampleData\MEmpasciiDelimPayOnly.dat

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
P	003715	01152000	01162000	00101	0005000.00	0007000.00	150.00	200.00	133.00
P	003715	02152000	02162000	00102	0003000.00	0008000.00	120.00	180.00	120.00
P	003715	03152000	03162000	00103	0005000.00	0009000.00	130.00	170.00	110.00
P	003715	04152000	04162000	00104	0003000.00	0010000.00	140.00	160.00	100.00
P	003941	01152000	01162000	00105	0003000.00	0010000.00	140.00	160.00	100.00
P	003941	02152000	02162000	00106	0003000.00	0010000.00	140.00	160.00	100.00

When you select a file format, the wizard updates the sample displayed at the bottom of the wizard page. You can use the scroll bars to navigate the sample data.

When your file is delimited, specify the following properties:

- **Field delimiter:** Field delimiters designate where one field ends and another begins. You can type in a field delimiter or select one from the drop-down list. The drop-down list displays common field delimiters. However, you may type in any character as a delimiter except the ones used for enclosures. The default is the comma (,).
- **Enclosures (Left and Right):** Some delimited files contain enclosures that denote text strings within a field. If the file contains enclosures, enter an enclosure character in the text box or select one from the drop-down list. The drop-down list displays common enclosures. However, you may type in any character. The default for both the left and right enclosure is the double quotation mark (“).

Selecting Record Types (Multiple Record Type Files Only)

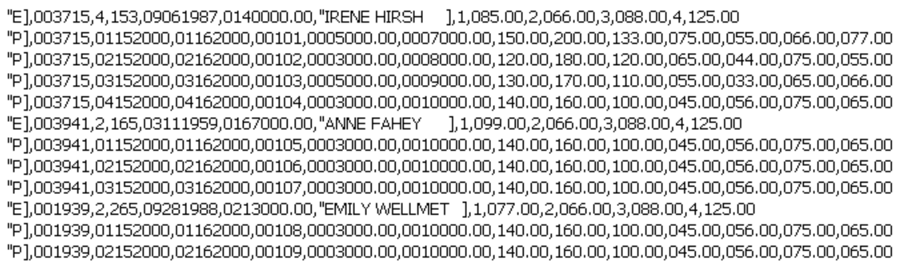
Use the Record Types wizard page to scan the flat file for record types, add or delete record types, and assign type values to the record types.

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 4-30. If the data file has a single record type and delimited file format, proceed to "Specifying Field Properties" on page 4-31.

Example: Flat File with Multiple Record Types

In files with multiple record types, one of the fields distinguishes one record type from the next. Figure 4–24 shows an example of a comma delimited file with two record types, “E” and “P”. When you use the Flat File Sample Wizard, you instruct the wizard to scan a specified field of every record for the record type values. In this case, instruct the wizard to scan the first field. The wizard returns “E” and “P” as the type values.

Figure 4–24 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.

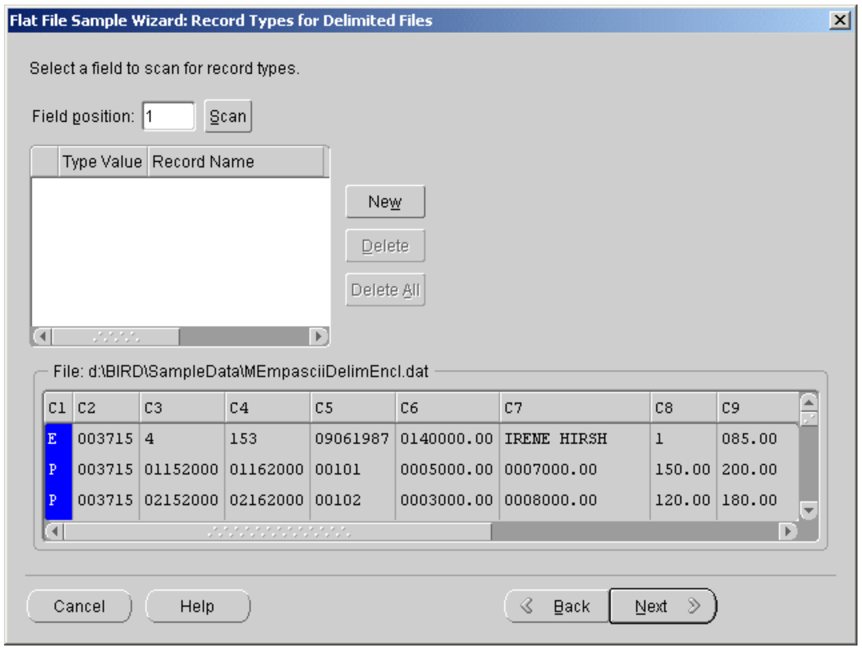
Because sampling cannot be cancelled after it has been started, make sure you pick a "reasonable" number of rows for optimum performance. If all record types do not appear within a reasonable number of rows, you can mock up a sample file with rows selected from different parts of the master file to provide 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 4–25 shows the first field position selected for scanning for multiple record types.

Figure 4–25 Record Types Page for Delimited Files



Flat File Sample Wizard: Record Types for Delimited Files

Select a field to scan for record types.

Field position: 1

Type Value	Record Name

File: d:\BIRD\SampleData\MEmpasciiDelimEncl.dat

C1	C2	C3	C4	C5	C6	C7	C8	C9
E	003715	4	153	09061987	0140000.00	IRENE HIRSH	1	085.00
P	003715	01152000	01162000	00101	0005000.00	0007000.00	150.00	200.00
P	003715	02152000	02162000	00102	0003000.00	0008000.00	120.00	180.00

To complete the Records Type page for a delimited file:

1. Select the one field that identifies the record types in the file.

   The wizard displays all the fields in a sample in the lower panel of the page. Select the field from the sample box. Or, in Field position, you can type in the position as it appears in the sample. Unless you specify otherwise, the wizard defaults to the first field in the file.

   The wizard scans the file for the field and displays the type values. The wizard assigns default record names (RECORD1, RECORD2...) to each type value.

2. You can edit the record names.

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

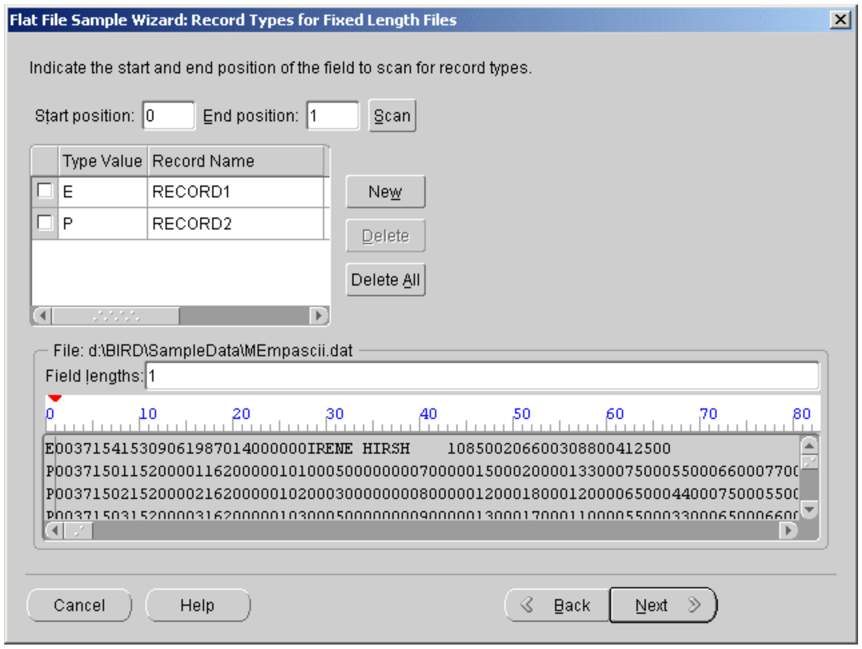
3. Click **Next** to continue with the wizard.

Defining Multiple Record Organization in a Fixed-Length File

When a fixed-length flat file contains several different types of records, you can use the scanning feature within the Flat File Sample Wizard to search for record types and assign a type value to each record type.

Figure 4–26 shows the results from scanning for record types based on the first field position.

Figure 4–26 Record Types Page for Fixed Length Files


Flat File Sample Wizard: Record Types for Fixed Length Files

Indicate the start and end position of the field to scan for record types.

Start position: 0 End position: 1

	Type Value	Record Name
☐	E	RECORD1
☐	P	RECORD2

File: d:\BIRD\SampleData\MEmpascii.dat

Field lengths: 1

```
E00371541530906198701400000IRENE HIRSH    1085002066003088004125OO
P00371501152000011620000010100050000000070000015000200001330007500055000660007700
P00371502152000021620000010200030000000080000012000180001200006500044000750005500
P00371503152000031620000010300050000000090000013000170001100005500033000650006600
```

To complete the Records Type page for a fixed-length file:

1. Specify the one field that identifies the record types in the file. Type in values for the **Start position** and **End position.** If you want to scan for records based on the first field, enter 0 for **Start Position.**

   The wizard indicates the selected field with a red tick mark in the ruler in the file sample in the lower panel of the page.

2. Click **Scan.**

   The wizard scans the file field and displays the type values. The wizard assigns default record names (RECORD1, RECORD2...) to each type value.

3. You can edit the record names.

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

4. Click **Next** to continue with the wizard.

Specifying Field Lengths (Fixed-Length Files Only)

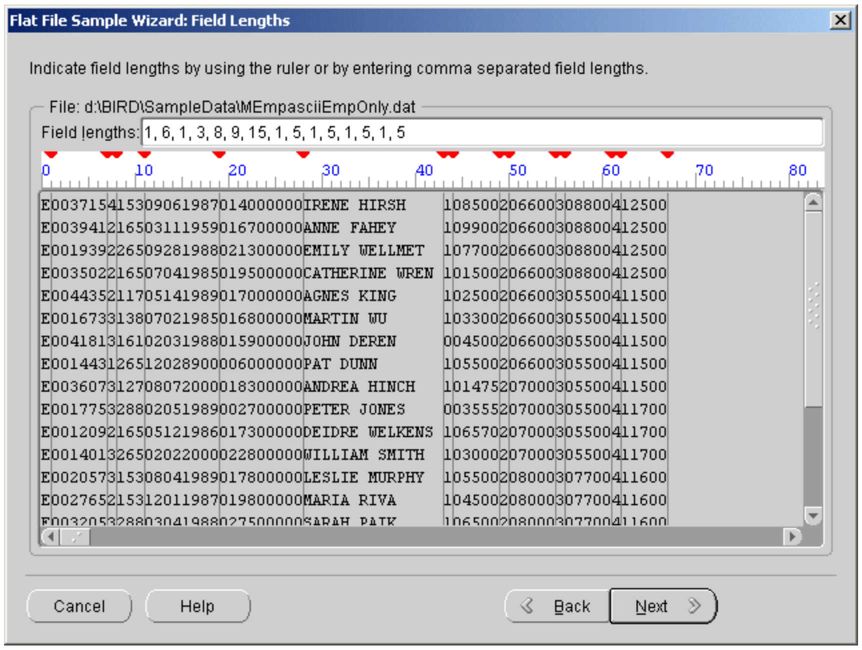
When you use the Flat File Sample Wizard to define a fixed-length flat file, you also need to define the length of each field in the file.

Note: This step is not necessary for delimited files. Proceed to "Specifying Field Properties" on page 4-31.

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

Figure 4–27 displays the Field Lengths page from the Flat File Sample Wizard.

Figure 4–27 Field Lengths Page for the Flat File Sample Wizard


Flat File Sample Wizard: Field Lengths
Indicate field lengths by using the ruler or by entering comma separated field lengths.
0
10
20
30
40
50
60
70
80
Cancel
Help
Back
Next

File: d:\BIRD\SampleData\MEmpasciiEmpOnly.dat

Field lengths: 1, 6, 1, 3, 8, 9, 15, 1, 5, 1, 5, 1, 5, 1, 5

```
E003715415309061987014000000IRENE HIRSH    108500206600308800412500
E003941216503111959016700000ANNE FAHEY     109900206600308800412500
E001939226509281988021300000EMILY WELLMET  107700206600308800412500
E003502216507041985019500000CATHERINE WREN 101500206600308800412500
E004435211705141989017000000AGNES KING     102500206600305500411500
E001673313807021985016800000MARTIN WU      103300206600305500411500
E004181316102031988015900000JOHN DEREN     004500206600305500411500
E001443126512028900006000000PAT DUNN       105500206600305500411500
E003607312708072000018300000ANDREA HINCH   101475207000305500411500
E001775328802051989002700000PETER JONES    003555207000305500411700
E001209216505121986017300000DEIDRE WELKENS 106570207000305500411700
E001401326502022000022800000WILLIAM SMITH  103000207000305500411700
E002057315308041989017800000LESLIE MURPHY  105500208000307700411600
E002765215312011987019800000MARIA RIVA     104500208000307700411600
E003205328803041988027500000SARAH PAIK     106500208000307700411600
```

If you know the length of each field, type in the field length in **Field Lengths.** Separate each length by commas. The wizard displays the changes to the sample at the bottom of the wizard page.

To use the ruler, click any number or hash mark on the ruler. The wizard displays a red tick mark on top of the ruler and marks the boundary with a red line. If you make a mistake, double-click the marker to delete it or move the marker to another position. Use the ruler to create markers for each field in the file.

Note that when you specify a field length using the ruler, your tick markers indicate the starting and ending borders for each field. From this information, Warehouse Builder determines the positions occupied by each field. For example, a three-character field occupying positions 6, 7, and 8 is internally identified with the beginning and ending values of '5,8'.

If you define field positions for a file in OMB Plus and later look at the properties of that same file in the Warehouse Builder user interface Flat File Properties screen, the field definition appears different. The same three-character record that you defined in OMB Plus with the beginning and ending values of '6,8' appears with the positions '5.8' in the Flat File Properties screen.

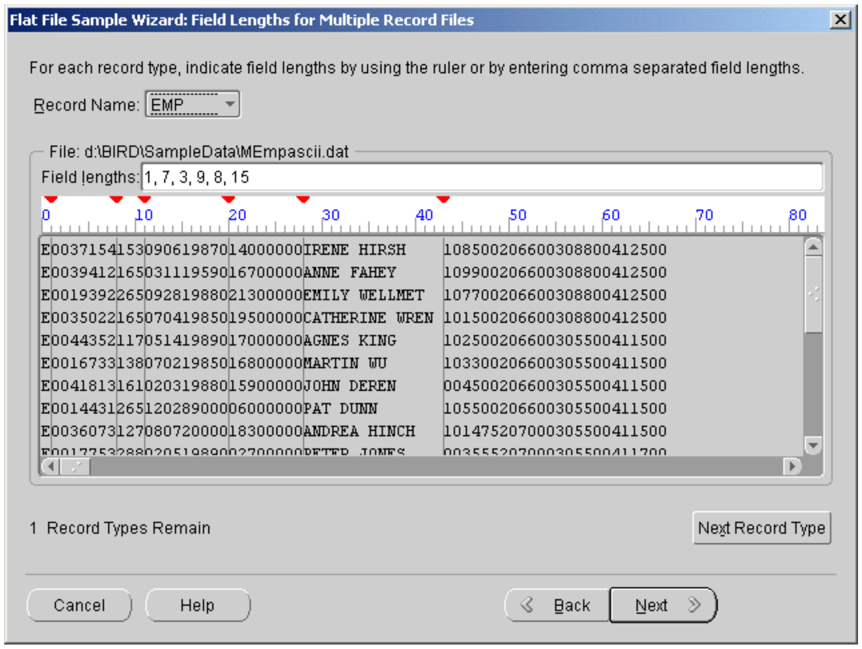
Specifying Field Lengths for Multiple Record Files

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 4–28 shows the Field Lengths page for a fixed length file with multiple record types.

Figure 4–28 Field Lengths for Multiple Record Files page


Flat File Sample Wizard: Field Lengths for Multiple Record Files
0
10
20
30
40
50
60
70
80
Next Record Type
Cancel
Help
Back
Next

For each record type, indicate field lengths by using the ruler or by entering comma separated field lengths.

Record Name: EMP

File: d:\BIRD\SampleData\MEmpascii.dat

Field lengths: 1, 7, 3, 9, 8, 15

```
E0037154153090619870140000000IRENE HIRSH    108500206600308800412500
E0039412165031119590167000000ANNE FAHEY     109900206600308800412500
E0019392265092819880213000000EMILY WELLMET  107700206600308800412500
E0035022165070419850195000000CATHERINE WREN 101500206600308800412500
E0044352117051419890170000000AGNES KING     102500206600305500411500
E0016733138070219850168000000MARTIN WU      103300206600305500411500
E0041813161020319880159000000JOHN DEREN     004500206600305500411500
E0014431265120289000060000000PAT DUNN       105500206600305500411500
E0036073127080720000183000000ANDREA HINCH   101475207000305500411500
E0017753288020519890027000000PETER JONES    003555207000305500411700
```

1 Record Types Remain

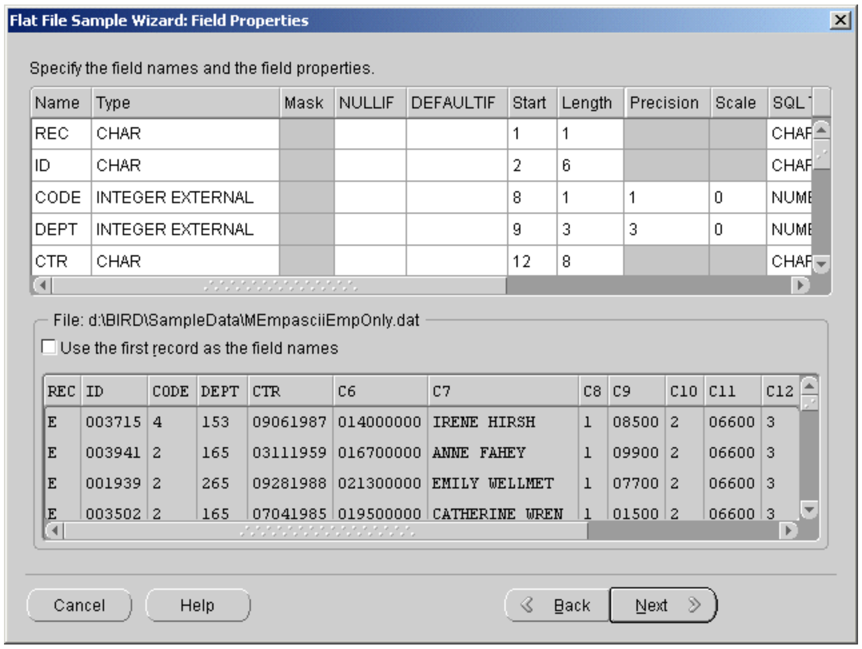
Specifying Field Properties

Use the Properties page in the Flat File Sample Wizard to define properties for each field.

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.

Figure 4–29 shows the Properties page for the Flat File Sample Wizard.

Figure 4–29 Properties Page for the Flat File Sample Wizard



Specify the field names and the field properties.

Name	Type	Mask	NULLIF	DEFAULTIF	Start	Length	Precision	Scale	SQL T
REC	CHAR				1	1			CHAR
ID	CHAR				2	6			CHAR
CODE	INTEGER EXTERNAL				8	1	1	0	NUME
DEPT	INTEGER EXTERNAL				9	3	3	0	NUME
CTR	CHAR				12	8			CHAR

File: d:\BIRD\SampleData\MEmpasciiEmpOnly.dat

☐ Use the first record as the field names

REC	ID	CODE	DEPT	CTR	C6	C7	C8	C9	C10	C11	C12
E	003715	4	153	09061987	014000000	IRENE HIRSH	1	08500	2	06600	3
E	003941	2	165	03111959	016700000	ANNE FAHEY	1	09900	2	06600	3
E	001939	2	265	09281988	021300000	EMILY WELLMET	1	07700	2	06600	3
E	003502	2	165	07041985	019500000	CATHERINE WREN	1	01500	2	06600	3

For each field, the wizard displays the following two sets of properties:

- **SQL*Loader Field Properties:** SQL*Loader properties refer to data types supported in flat files using the SQL*Loader utility. In Warehouse Builder this implies every mapping that uses flat file as a source. The wizard displays this set of properties first. SQL*Loader properties include Name, Type, Mask, NULLIF, DEFAULTIF, Start, and Length. The Precision and Scale fields are reserved for future use.
- **SQL Properties:** SQL Properties refer to relational data type defaults associated with a given flat file field. You can choose a default data type value that will be automatically updated with any changes to the flat file, or the SQL Loader, data type. You can also override these defaults with any value. These values will then become independent of the flat file data type. These properties will be the default column attributes when you map a flat file to an unbound table or when you create an external table. The wizard displays this set of properties second. SQL properties include SQL Type, SQL Length, SQL Precision, and SQL Scale.

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. You can select more files to sample or select **Finish** to begin the import. For information on how to continue sampling files, see Step 8 in "Using the Import Metadata Wizard Flat Files" on page 4-21.

SQL*Loader Properties

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.

You can edit the following SQL* Loader properties:

Name The wizard assigns a name to each field. It assigns 'C1'to the first field, 'C2' to the second, and so on. To rename fields, click a field and type a new name.

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

Type Describes the data type of the field for the SQL*Loader. You can use the Flat File Sample Wizard to import the following data types: CHAR, DATE, DECIMAL EXTERNAL, FLOAT EXTERNAL, INTEGER EXTERNAL, ZONED , AND ZONED EXTERNAL. For complete information on SQL*Loader field and data types, refer to Oracle Database *Utilities*. Currently, only portable datatypes are supported.

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 is: =BLANKS, ='quoted string', =X'ff' (hex value), != (not equal) is also allowed.

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 is: =BLANKS, ='quoted string', =X'ff' (hex value), != (not equal) is also allowed.

Start In fixed length files, indicates the field start 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. This field is reserved for future use.

SQL Properties

The Flat File Sample Wizard assigns default values to these properties based on their corresponding SQL*Loader properties. Warehouse Builder can use the SQL properties in one of the following ways:

- **External table:** After you use the Flat File Sample Wizard to import metadata, you can create an external table based on a single flat file record type. When you use the external table in a mapping, 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 3-18.
- **Populating an Empty Mapping Table.** After you use the Flat File Sample Wizard to import metadata, you can populate an empty relational table with the metadata. The table inherits the SQL properties you defined for the flat file source.

- **Flat file target:** After you use the Flat File Sample Wizard to import metadata, you can use the flat file as a target in a mapping. Flat file targets do not inherit SQL properties, everything defaults to character data type. This option is not used here. For more information about using a flat file operator as a target, see "Mapping Flat File Operator" on page 8-22.

Edit the SQL properties or accept the defaults to specify how the fields with SQL* Loader data types should be mapped to the columns with SQL data types. While default values adjust to SQL Loader values, user specific changes remain constant irrespective of the SQL Loader data type values. These SQL properties are used in mapping, validation, and generation.

SQL Type Describes the SQL data type. Warehouse Builder supports the following set of portable SQL data types:

- CHAR
- DATE
- FLOAT
- NUMBER
- VARCHAR
- VARCHAR2

SQL Length Defines the field length to be used in SQL.

SQL Precision Defines the field precision to be used in SQL for NUMBER and FLOAT datatypes.

SQL Scale Defines the field scale to be used in SQL for NUMBER and FLOAT datatypes.

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 navigation tree.
2. Right-click the file name and select **Properties.**

   Warehouse Builder displays the Flat File property sheet with the following tabs:

   General: Use this tab to edit the name and description of the definition. You can also 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: 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: Use this tab to edit field level attributes, SQL Loader and SQL Properties.

General Tab

Use this tab to edit the name and description of the definition. You can also 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 4–2 shows an example of what the record type values for the two sample records earlier might be:

Table 4–2 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 checkbox 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, 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.

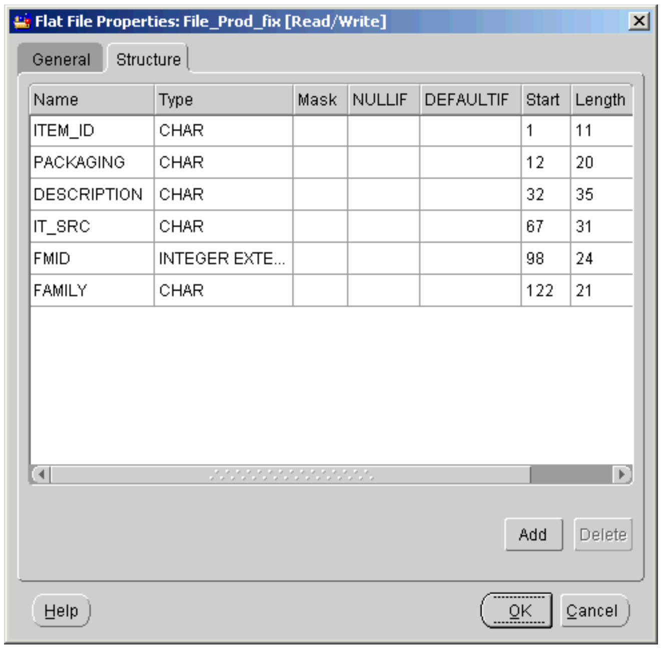
The wizard deactivates properties that do not apply to a given data type. For example, you can edit the length for a CHAR, but precision and scale are not available.

For each field, the wizard displays the following two sets of properties:

- **SQL* Loader Properties:** The wizard displays this set of properties first. SQL*Loader properties include Name, Type, Mask, NULLIF, DEFAULTIF, Start, and Length. The Precision and Scale fields are reserved for future use.
- **SQL Properties:** The wizard displays this set of properties second. SQL properties include SQL Type, SQL Length, SQL Precision, and SQL Scale.

Figure 4–30 shows the Structure tab for a single record type data file.

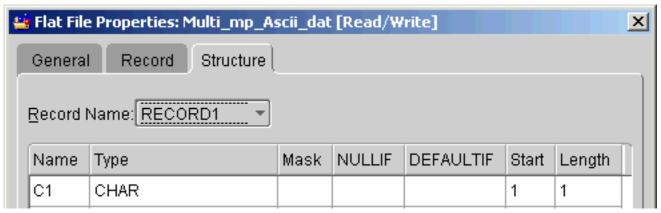
Figure 4–30 Structure Tab for a Single Record Type Data File



Name	Type	Mask	NULLIF	DEFAULTIF	Start	Length
ITEM_ID	CHAR				1	11
PACKAGING	CHAR				12	20
DESCRIPTION	CHAR				32	35
IT_SRC	CHAR				67	31
FMID	INTEGER EXTE...				98	24
FAMILY	CHAR				122	21

Figure 4–31 displays the Structure tab for a multiple record type data file.

Figure 4–31 Structure Tab for a Multiple Record Type Data File



Record Name: RECORD1

Name	Type	Mask	NULLIF	DEFAULTIF	Start	Length
C1	CHAR				1	1

5 Configuring Data Objects

Earlier in the design phase, you defined a logical model for your target system using Warehouse Builder design objects. This chapter discusses how you assign physical properties to those design objects.

This chapter first describes how you create indexes and partitions for design objects such as tables, materialized views, dimensions, and cubes defined in Warehouse Builder. Next, this chapter discusses how you assign physical properties to Warehouse Builder object definitions.

This chapter includes:

Creating Indexes and Partitions

In Warehouse Builder, indexes and partitions can be created for Tables, Materialized Views, Dimensions, and Cubes. Indexes and partitions are created to enhance query performance of your data warehouse.

This section includes:

- Creating Partitions on page 5-5

About Indexes

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

- Index column values are stored presorted.
- Because indexes are stored in a separate area of the database, they can be created or dropped at any time without any effect on the underlying table.
- When data is deleted, added, or updated, the indexes are maintained automatically. They are independent of the data in the table.

B*-Tree and Bitmap indexes are particularly useful in data warehousing. To learn more about indexes and indexing strategies, see the Oracle Database *Data Warehousing Guide.*

In Warehouse Builder, after you define the data objects, such as tables, materialized views, dimensions, and cubes, you can create indexes on them according to the requirements of your target system. After creating the indexes, you need to configure them with physical properties to enable deployment. The following sections show you how to create and configure indexes in Warehouse Builder.

Creating Indexes

You can create and configure indexes on cubes, dimensions, tables, and materialized views.

To create indexes:

1. From the Warehouse Builder navigation tree, right-click the appropriate design object and select **Configure.**

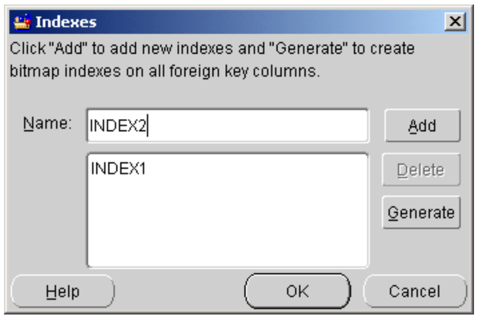
   The Configuration Properties dialog displays.

2. From the Configuration Properties dialog, select the field to the right side of Indexes.

3. Click the **...** button.

   The Indexes dialog displays as shown in Figure 5–1.

Figure 5–1 Indexes Dialog


Indexes

Click "Add" to add new indexes and "Generate" to create bitmap indexes on all foreign key columns.

Name: INDEX2

INDEX1

4. Type a name for the index in the Name field and click **Add.**

   Repeat this step for each index you want to create.

5. Click **OK** if the indexes are correct. If not, click **Cancel.**

   You are now ready to configure your index for deployment.

Creating Bitmap Indexes

Warehouse Builder utilizes the bitmap indexing feature available in the Oracle database to provide pointers to the rows in a table that contain a given key value. In data warehousing, bitmaps are created to enable star query transformations. The star transformation is a cost-based query transformation 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. For more information, see the *Oracle Database Data Warehousing Guide.*

To create bitmap indexes:

1. From the Warehouse Builder navigation tree, right-click the appropriate design object and select **Configure.**

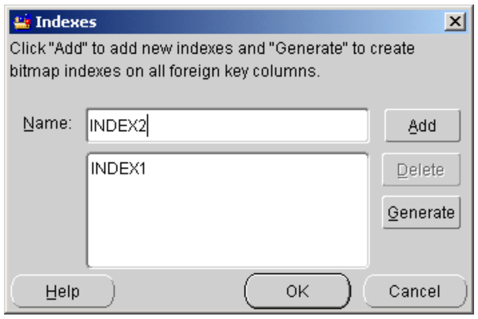
   The Configuration Properties dialog displays.

2. From the Configuration Properties dialog, select the field to the right side of Indexes.

3. Click the **...** button.

   The Indexes dialog displays as shown in Figure 5–2.

Figure 5–2 Indexes Dialog


Click "Add" to add new indexes and "Generate" to create bitmap indexes on all foreign key columns.

Name: INDEX2

INDEX1

4. Type a name for the index in the Name field and click **Add.**

   Repeat this step for each index you want to create.

5. If you are creating bitmap indexes, click **Generate.**

   Bitmap indexes are created on all foreign key columns. Before bitmap indexes are generated, Warehouse Builder checks the table for foreign keys. Warehouse Builder does not generate bitmap indexes if:

   No foreign keys: There are no foreign keys on the table. In this case, Warehouse Builder indicates that no index can be created because there are no foreign keys on the table.

   No columns: There are foreign keys on the table but there are no columns defined for them.

   Indexes exist: There are foreign keys on the table and some of them already have indexes. If there is already a bitmap index created on one of the columns, then the column name and the name of the existing index displays in the Impact Report dialog along with a message that no new bitmap index can be created for these columns.

The Impact Report for Bitmap Index Creation dialog displays.

6. Click **OK** if the indexes are correct. If not, click **Cancel.**

Follow the steps listed in the following section to configure your index for deployment.

Configuring Indexes

After you create indexes, you must define their parameters using the Configuration Properties dialog.

To configure indexes:

1. In the Configuration Properties dialog, expand the **Indexes** node to display the configuration parameters.
2. Configure the following parameters for indexes:

   Log to Redo Log File: Indicates whether the index creation logs in the redo log file.

   Parallel: If set to `PARALLEL`, this parameter enables parallel processing when a table is created.

   Tablespace: Specify the tablespace where the index is created.

   Set the **Index Type** parameter to `BITMAP,UNIQUE`, or leave it blank for a non-unique B*-tree index. If you are configuring a Bitmap index, the bitmap option is automatically selected.

   Local Index: A local index reflects the structure of its underlying table. When configured `TRUE`, this parameter specifies that the index is partitioned on the same columns, with the same number of partitions, and the same partition bounds as its underlying table.

   Deployable: Set to `TRUE` to indicate that you want to deploy this index.

   Index Columns: Click the **...** button to display the Index Columns dialog. From the Choices field, select the columns to be included in the index. Click the right arrows to move them to the Included list. Click **OK.**
3. Close the Configuration Properties dialog.

Renaming Indexes

To rename an index:

1. From the Configuration Properties dialog, expand **Indexes.**
2. Click the column to the right of the index you want to rename and click the **...** button.

   The Indexes dialog displays.
3. Type a new name for the index in the Rename field and click **OK.**
4. The new index name displays under **Indexes** in the Configuration Properties dialog.

About Partitions

Partitions enable you to efficiently manage very large tables and indexes by dividing them into smaller, more manageable parts. Partitions can be created to improve query and load performance and to simplify the management of physical storage. Because partitions can be managed individually and can operate independently of other partitions, they provide a structure that can be better tuned for performance. DML statements can access and manipulate individual partitions rather than entire tables or indexes.

In Warehouse Builder, you can create two types of partitions:

- **Range:** Range partitioning is based on ranges of partition key values that you establish for each partition. Only data with a particular set of values is contained in each partition. It is the most common type of partitioning and is often used with dates. For example, you might want to partition sales data into monthly partitions.
- **Hash:** When you implement hash partitioning, you choose a partitioning key and the number of partitions. Oracle9*i* divides the data evenly across all the partitions. Choose hash partitioning to control the physical placement of data across a fixed number of partitions. Hash partitioning is useful when data is not historical and there is no obvious column or column list

You can create and then configure partitions for Tables, Materialized Views, Cubes, and Dimensions. After creating the partitions, you need to configure them with physical properties to enable deployment. The following sections show you how to create and configure partitions in Warehouse Builder.

Creating Partitions

To create partitions in Warehouse Builder, you need to follow these steps:

1. Determine the type of partition you want to create: Hash or Range.
2. Create a partition key entry and indicate the type of partition you are creating. A partition key contains the set of columns that determine how to partition a table. See "Creating a Partition Key Entry" on page 5-5.
3. Create a Range or Hash partition. See "Creating Hash Partitions" on page 5-6 and "Creating Range Partitions" on page 5-6.
4. Configure the Local Index parameter for Indexes to indicate if you want to partition them or not. See "Partitioning Indexes" on page 5-7.

The following sections outline these steps in detail.

Creating a Partition Key Entry

To create a partition key entry:

1. From the Warehouse Builder navigation tree, right-click the appropriate design object and select **Configure.**

   The Configuration Properties dialog displays.
2. From the Configuration Properties dialog, select the field to the right side of Partition Keys.

   Warehouse Builder displays the Partition Keys dialog.
3. From the Choices field, select the columns you want to include.
4. Click the right arrow buttons to move the selected columns to the Included field.

5. Click **OK.**
6. From the Configuration Properties dialog, expand the Partition Keys node and expand a partition key name.
7. Select the type of partition you want to create: RANGE or HASH.

   Range partition information always overwrites Hash partition information.

   If you choose to create a Hash partition, follow the steps following this section. If you choose to create a Range partition, see "Creating Range Partitions" on page 5-6.

Creating Hash Partitions

To create a hash partition:

1. From the Configuration Properties dialog, expand the **Hash Partition Parameters** node.
2. Specify the number of Hash subpartitions in the Hash SubPartition Number field.
3. Provide a tablespace name for the partition in the Hash Partition Tablespace field.

Creating Range Partitions

To create a range partition:

1. Expand the **Range Partitions** node in the Configuration Properties dialog.
2. Create the range partitions by clicking the **...** button.

   The Range Partitions dialog displays.
3. Type a partition names in the Name field and click **Add.**

   To delete a partition name from the list, select a partition name and click **Delete.**
4. Click **OK.**

   The range partitions are displayed under the Range Partitions node in the Configuration Properties dialog.
5. Expand a range partition name to configure the following parameters:

   Date Less Than: Type the non-inclusive upper limit for the current partition. This entry is an ordered list of literal values corresponding to column_list in the partition_by_range_clause. You can substitute the keyword MAXVALUE for any literal in value_list. MAXVALUE specifies a maximum value that sorts higher than any other value, including NULL.

   Warehouse Builder uses the partition name and the Value Less Than property to generate the DDL script for partitioning the table. The Value Less Than property defines the contents of the partition.

   Tablespace: Type the name of the physical attribute of the tablespace associated with the partition. If the value is not specified, then the tablespace for this partition will take the default value of the tablespace specified by the table.

   Deployable: Select TRUE to indicate that you want to deploy this partition. Warehouse Builder only generates scripts for partitions marked deployable.

Renaming Range Partitions

To rename a range partition:

1. Open the Configuration Properties dialog for the appropriate data object.

2. Expand the Partition node.
3. Click the field next to the partition name you want to edit and click the **...** button.

   The Partitions dialog displays.
4. Highlight the value in the Rename field and type a new name.
5. Click **OK.**

Partitioning Indexes

A local index is created to reflect the structure of the underlying table. It is partitioned on the same columns as the underlying table, creating the same number of partitions or subpartitions, and partition bounds as the corresponding partitions of the underlying table.

To create a local index:

1. From the Configuration Properties dialog, expand the **Indexes** node.
2. Expand the Index Type node.
3. Set the Local Index parameter to **True** if you want to create a partitioned index.

Configuring Warehouse Builder Design Objects

In this phase, you assign physical deployment properties to the object definitions you created in Warehouse Builder by configuring properties such as tablespaces, partitions, and other identification parameters. You also configure runtime parameters such as job names, and runtime directories.

These physical properties are set using the Configuration Properties window. Properties can be set for for target modules or for each individual design object such as tables, dimensions, views, or mappings. The following sections show you how to assign physical properties to your logical design model.

Configuring Target Modules

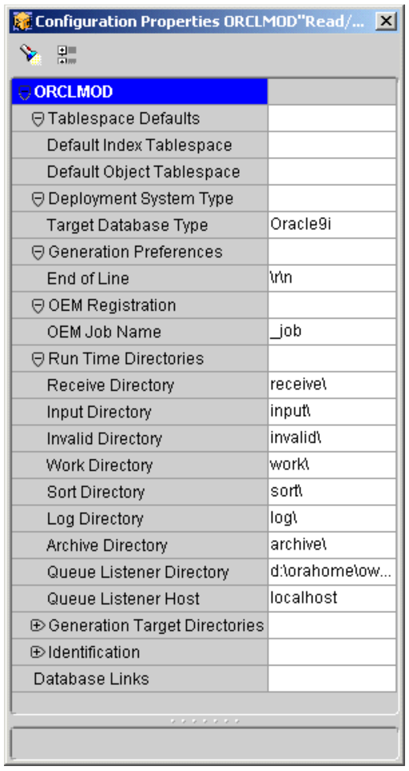
Each target module provides top level configuration options for all the objects contained in that module.

To configure a Target Module:

1. From the Warehouse Builder navigation tree, expand **Databases,** expand **Oracle,** and right-click a target module name and select **Configure.**

   Warehouse Builder displays the Configuration Properties dialog as shown in Figure 5–3.

Figure 5–3 Configuration Properties Window for Modules



ORCLMOD	
Tablespace Defaults	
Default Index Tablespace	
Default Object Tablespace	
Deployment System Type	
Target Database Type	Oracle9i
Generation Preferences	
End of Line	\r\n
OEM Registration	
OEM Job Name	_job
Run Time Directories	
Receive Directory	receive\
Input Directory	input\
Invalid Directory	invalid\
Work Directory	work\
Sort Directory	sort\
Log Directory	log\
Archive Directory	archive\
Queue Listener Directory	d:\orahome\ow...
Queue Listener Host	localhost
Generation Target Directories	
Identification	
Database Links	

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 **...** to display another properties dialog.

3. Configure the following Storage Tablespace parameters:

   Default Index Tablespace: Defines the name of each tablespace where indexes are created. The default is null. If you configure an index tablespace at the target module level and not at the object level, Warehouse Builder uses the tablespace value configured at the target module level, during code generation. If you configure a tablespace for each index at the object level, Warehouse Builder overwrites the tablespace value configured at the target module level.

   Default Object Tablespace: Defines the name of each tablespace where objects are created, for example, tables, views, or materialized views. The default is null. If you configure object tablespace at the target module level and not at the individual object level, Warehouse Builder uses the value configured at the target module level, during code generation. If you configure a tablespace for each individual object, Warehouse Builder overwrites the tablespace value configured at the target module level.

4. Configure the following Generation Preferences:

   End of Line: Defines the end of line markers for flat files. This is dependent on the platform to which you are deploying your warehouse. For UNIX, use `\n`, and for NT, use `\r\n`.

   PL/SQL Generation Mode: Defines the target database type. Code generation is based on the your choice in this field. For example, selecting Oracle Database ensures the use of Oracle Database code constructs, selecting Oracle8*i* would generate row-based code.

Oracle Warehouse Builder introduces new functionality available for Oracle9*i* databases and higher only. If you select Oracle8*i* for the PL/SQL Generation Mode, some Oracle9*i* Warehouse Builder functionalities such as Table Functions and External Tables are not available. For a list of Oracle Warehouse Builder not compatible with releases prior to Oracle9*i*, see the *Oracle Warehouse Builder Release Notes* .

5. Configure the following preferences for the 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\.

6. Configure the following 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 Extension: Type a file name extension for 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\`.

   ABAP Directory: For all ABAP configuration related to SAP tables, see Appendix 21, "Using SAP R/3 Data in Warehouse Builder".

   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`.

7. Configure the following Identification parameters:

   Main Application Short Name

   Application Short Name

   Schema Owner

   Connect String

   Remote Host Name

   Port

   Service Name

   Top Directory

   Deployable

Configuring Tables

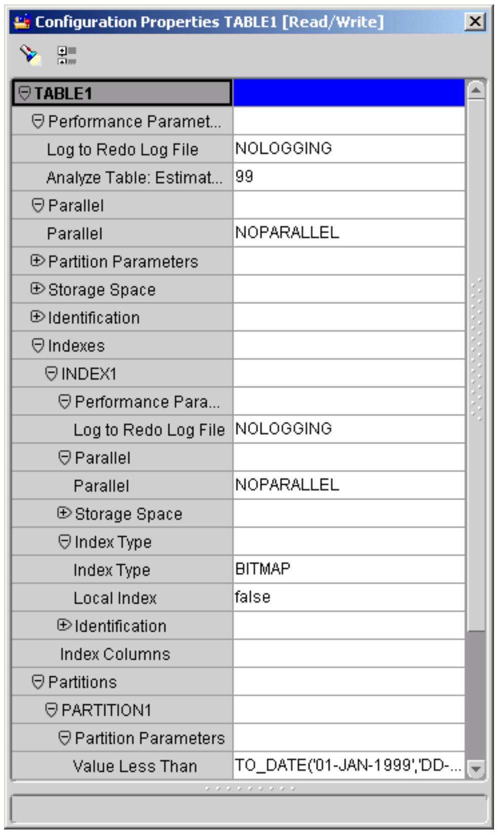
Warehouse Builder generates DDL scripts for each table defined in a target module. Follow these steps to configure a table.

To configure the physical properties for a table:

1. Right-click the name of a table and select **Configure.**

   Warehouse Builderdisplays the Configuration Properties dialog as shown in Figure 5–4.

Figure 5–4 Table Configuration Properties



Configuration Properties TABLE1 [Read/Write]

TABLE1	
Performance Paramet...	
Log to Redo Log File	NOLOGGING
Analyze Table: Estimat...	99
Parallel	
Parallel	NOPARALLEL
Partition Parameters	
Storage Space	
Identification	
Indexes	
INDEX1	
Performance Para...	
Log to Redo Log File	NOLOGGING
Parallel	
Parallel	NOPARALLEL
Storage Space	
Index Type	
Index Type	BITMAP
Local Index	false
Identification	
Index Columns	
Partitions	
PARTITION1	
Partition Parameters	
Value Less Than	TO_DATE('01-JAN-1999','DD-...

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`.
- **Analyze Table: Estimate Percent:** This value represents the sample size as a percentage of total rows. When set to a nonzero value, Warehouse Builder generates a DDL script to analyze the table.

Parallel

- **Parallel:** Enables parallel processing when the table has been created. The default is `PARALLEL`.

Storage Space

- **Tablespace:** Defines the name of each tablespace where the table is created. The default value is null. If you accept the default value of null, Warehouse Builder generates the table based on the tablespace value set in the target module configuration properties. If you configure the tablespace for individual objects, Warehouse Builder overwrites the tablespace value configured for the target module.

Identification

- **Deployable:** Select TRUE to indicate if you want to deploy this table constraint. Warehouse Builder generates scripts only for table constraints marked deployable.

Indexes

Create and configure indexes as described in "Configuring External Tables" on page 5-12 and "Configuring Indexes" on page 5-4.

Constraints

- **Using Index:** Create a constraint using an existing index.
- **Deployable:** Select TRUE to indicate if you want to deploy this table constraint. Warehouse Builder generates scripts only for table constraints marked deployable.

Configuring External Tables

Configure the following properties for an external table:

- Access Specification
- Reject
- Data Characteristics
- Parallel
- Field Editing
- Identification
- Data Files

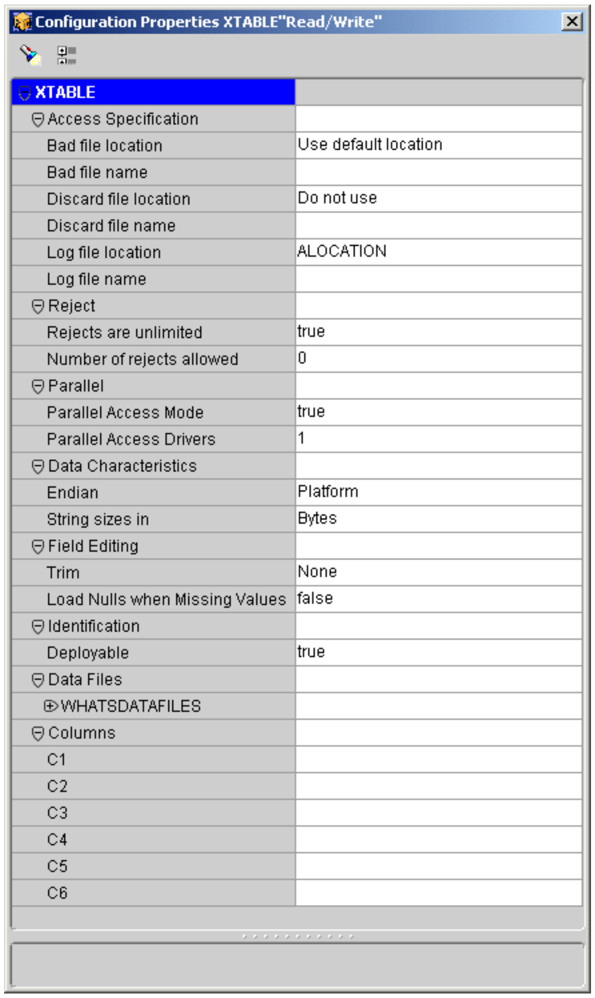
Note: When you import an external table into the repository and when you manually define access parameters for an external table, some external table configuration properties are overruled by settings on the Access Parameters tab in the External Table Properties window.

To configure the physical properties for an external table:

1. Select an external table from the navigation tree.
2. From the Edit menu, select **Configure.** You can also click the Configure icon from the tool bar.

   The Configuration Property window displays as shown in Figure 5–5.

Figure 5–5 External Table Configuration Properties



XTABLE	
Access Specification	
Bad file location	Use default location
Bad file name	
Discard file location	Do not use
Discard file name	
Log file location	ALOCATION
Log file name	
Reject	
Rejects are unlimited	true
Number of rejects allowed	0
Parallel	
Parallel Access Mode	true
Parallel Access Drivers	1
Data Characteristics	
Endian	Platform
String sizes in	Bytes
Field Editing	
Trim	None
Load Nulls when Missing Values	false
Identification	
Deployable	true
Data Files	
WHATSDATAFILES	
Columns	
C1	
C2	
C3	
C4	
C5	
C6	

3. To configure a property, click the white space and make a selection from the drop down box.

Access Specification

If you imported the external table into the repository or created the external table without specifying a source file, do not configure these properties. Access specification properties are overruled by settings on the Access Parameters tab in the External Table Properties window.

Under **Access Specification**, you can indicate the following file names and locations Warehouse Builder uses to load the external table through SQL*Loader.

- **Bad File:** If you specify a name and location for a bad file, Warehouse Builder directs the Oracle database to write to that file all records that were not loaded due to errors. For example, records written to the bad file include those not loaded due to a datatype error in converting a field into a column in the external table. If you specify a bad file that already exists, the existing file is overwritten.
- **Discard File:** If you specify a name and location for a discard file, Warehouse Builder directs the Oracle database to write to that file all records that were not loaded based on a SQL *Loader load condition placed on the file. If you specify a discard file that already exists, the existing file is overwritten.

- **Log File:** If you specify a name and location for a log file, Warehouse Builder directs the Oracle database to log messages related to the external table to that file. If you specify a log file that already exists, new messages are appended.

For each of these files, you can either specify a file name and location, select **Do not use**, or select **Use default location.**

Reject

Under **Reject,** you can indicate how many rejected rows to allow. By default, the number of rejected rows allowed is unlimited. If you set **Rejects are unlimited** to false, enter a number in **Number of rejects allowed.**

Parallel

Parallel: Enables parallel processing. If you are using a single system, set the value to `NONPARALLEL` to improve performance. If you are using multiple systems, accept the default `PARALLEL`. The access driver attempts to divide data files into chunks that can be processed separately. The following file, record, and data characteristics make it impossible for a file to be processed in parallel:

- Sequential data sources (such as a tape drive or pipe).
- Data in any multibyte character set whose character boundaries cannot be determined starting at an arbitrary byte in the middle of a string. This restriction does not apply to any datafile with a fixed number of bytes for each record.
- Records with the VAR format

Data Characteristics

If you imported the external table into the repository or created the external table without specifying a source file, do not configure these properties. Data characteristics properties are overruled by settings on the Access Parameters tab in the External Table Properties window.

Under **Data Characteristics** you can set the following properties:

- **Endian:** The default for the Endian property is Platform. This indicates that Warehouse Builder assumes the endian of the flat file matches the endian of the platform on which it resides. If the file resides on a Windows platform, the data is handled as little-endian data. If the file resides on Sun Solaris or IBM MVS, the data is handled as big-endian. If you know the endian value for the flat file, you can select big or little-endian. If the file is UTF16 and contains a mark at the beginning of the file indicating the endian, Warehouse Builder uses that endian.
- **String Sizes in:** This property indicates how Warehouse Builder handles data with multibyte character sets, such as UTF16. By default, Warehouse Builder assumes the lengths for character strings in the datafile are in bytes. You can change the selection to indicate that strings sizes are specified in characters.

Field Editing

If you imported the external table into the repository or created the external table without specifying a source file, do not configure these properties. Field editing properties are overruled by settings on the Access Parameters tab in the External Table Properties window.

Under **Field Editing,** you can indicate the type of whitespace trimming to be performed on character fields in the datafile. The default setting in Warehouse Builder is to perform no trim. All other trim options can reduce performance. You can also set the trim option to trim blanks to the left, right, or both sides of a character field.

Another option is set the trim to perform according to the SQL*Loader trim function. If you select SQL*Loader trim, fixed-length files are right trimmed and delimited files specified to have enclosures are left trimmed only when a field is missing an enclosure.

You can indicate how to handle missing fields in a record. If you set the option Trim Missing Values Null to true, fields with missing values are set to NULL. If you set the property to false, fields with missing values are rejected and sent to specified bad file.

Identification

See "Identification" on page 5-12 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. Select the field to the right of **Data Files** and click the **...** button.

   The Data Files dialog displays.

2. Enter a name for the flat file and select **OK.**

   Warehouse Builder displays the flat file under the Data Files property.

3. Expand the newly added flat file and configure the following properties:

   Data File Location: Location for the flat file.

   Data File Name: Name of the flat file.

Configuring Advanced Queues

Follow these steps to configure an AQ in Warehouse Builder.

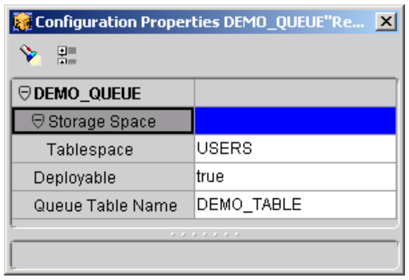
To configure an Advanced Queue:

1. From the Warehouse Builder console, highlight the name of the Advanced Queue.

2. From the Object menu, select **Configure.**

   The Configuration Properties dialog displays, as shown in Figure 5–6.

Figure 5–6 AQ Configuration Properties Window



DEMO_QUEUE	
Storage Space	
Tablespace	USERS
Deployable	true
Queue Table Name	DEMO_TABLE

3. Configure the following Storage Space parameter for the AQ:

   Tablespace: Name of the tablespace where the AQ and its corresponding table will be created.

4. Configure the following additional parameters:

   Deployable: Select TRUE if you want to generate scripts and deploy this AQ. Warehouse Builder only generates for AQs marked deployable.

   Queue Table Name: Provide the name of the AQ table that is used to persist the messages in the AQ.

5. Configure the following Generation Options:

   Generate Object Type: Indicate whether you want to deploy the object type associated with the AQ.

   Generate Queue Table: Indicate whether you want to deploy the queue table associated with the AQ.

   When you are deploying the second AQ that shares the same object type or queue table with other AQs, you must set the generation options for these properties as FALSE. If not, the deployment of the AQ will fail.

   Generate Advanced Queue: Indicate whether you want to deploy the AQ.

   Generate Temporary Table: Indicate whether you want to deploy the temporary table associated with the AQ. Even if the AQ already exists in your target system, the temporary table must be deployed by setting this option as True.

   The deployment of the queue table, the AQ, and the temporary table can be viewed under Advanced Queue in the Runtime Audit Browser.

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 navigation tree, right-click the a dimension name and select **Configure** from the pop-up menu.

   The Configuration Property window displays.

2. Follow the configuration guidelines listed for tables. For more information, see "Configuring Tables" on page 5-10.

3. Configure the following generation options for dimensions. With these parameters you can choose whether or not to generate the dimension and its underlying table.

   Generate Table: This can be set to true or false using the drop-down menu. Set to true to generate the underlying table. Set to false if you do not want to generate the underlying table.

   Generate Dimension: This can be set to true or false using the drop-down menu. Set to true to generate the dimension object. Set to false if you do not want to generate the dimension object.

For more information on dimensions, see the following section:

- "Creating a Dimension Definition" on page 3-41

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 navigation tree, right-click a cube name and select **Configure** from the pop-up menu.

   The Configuration Property window displays.

2. Follow the configuration guidelines listed for tables. For more information, see "Configuring Tables" on page 5-10.

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.

For more information on cubes, see the following sections:

- "Using Cubes" on page 3-47
- "Creating Cube Definitions" on page 3-48

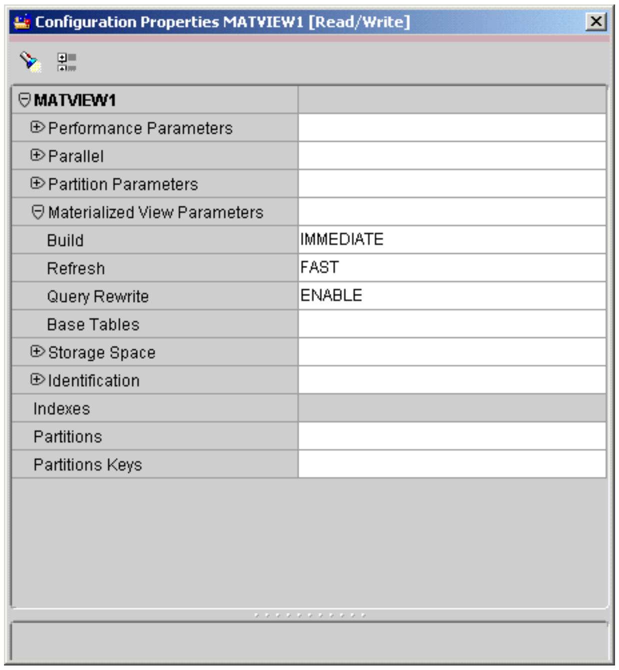
Configuring Materialized Views

To configure the physical properties for a materialized view:

1. From the navigation tree, right-click a materialized view name and select **Configure.**

   The Configuration Property window displays as shown in Figure 5–7.

Figure 5–7 Configuration Properties for Materialized Views



Configuration Properties MATVIEW1 [Read/Write]

MATVIEW1	
Performance Parameters	
Parallel	
Partition Parameters	
Materialized View Parameters	
Build	IMMEDIATE
Refresh	FAST
Query Rewrite	ENABLE
Base Tables	
Storage Space	
Identification	
Indexes	
Partitions	
Partitions Keys	

2. Follow the configuration guidelines listed for tables. For more information, see "Configuring Tables" on page 5-10.
3. Configure the Materialized View Parameters listed in the following section.

Materialized View Parameters

The following are parameters for materialized views:

Build

- **Immediate:** (Default) 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.

Refresh

- **Complete:** (Default) Oracle Database 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 5-18.
- **Force:** Oracle Database attempts to refresh using the fast mode. If unable to refresh in fast mode, Oracle Database re-executes the query upon refresh.

Query Rewrite

- **Enable:** (Default) Enables query rewrite. For other query rewrite requirements, see "Fast Refresh for Materialized Views" on page 5-18.
- **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.

Base Tables

To configure **Base Tables**, you must type the names of the 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. By default, this field is empty.

For related information, see "Using Materialized Views" on page 3-28.

Fast Refresh for Materialized Views

You can configure a materialized view in Warehouse Builder to refresh incrementally. When you update the base tables for a materialized view, the database stores updated record pointers in the materialized view log. Changes in the log tables are used to refresh the associated materialized views.

To ensure incremental refresh of materialized views in Warehouse Builder, verify the following conditions:

- The Refresh parameter must be set to ‘Fast’ and the Base Tables parameter must list all base tables.
- Each base table must have a PK constraint defined. Warehouse Builder generates a create statement based on the PK constraint and utilizes that log to refresh the dependent materialized views.

- The materialized view must not contain references to non-repeating expressions such as SYSDATE, ROWNUM, and non-repeatable PL/SQL functions.
- The materialized view must not contain references to RAW and LONG RAW data types.
- There are additional restrictions for materialized views with statements for joins, aggregations, and unions. For information on additional restrictions, refer to the 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 on views, see the following sections:

- "About Views" on page 3-25
- "Creating a View Definition" on page 3-25

Configuring Sequences

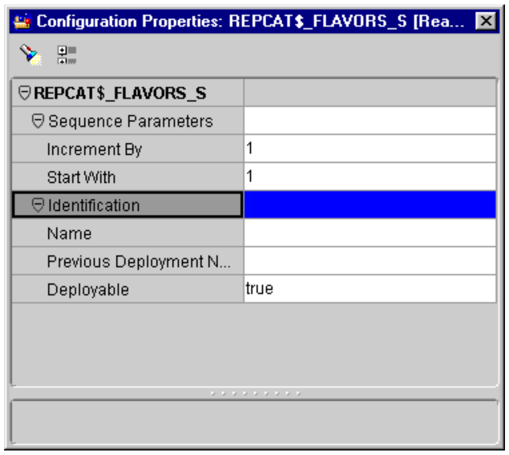
Warehouse Builder generates a script for each sequence object. A sequence object has a Start With and Increment By parameter. Both parameters are numeric.

To configure the physical properties for a sequence:

1. Right-click the name of a sequence and select **Configure.**

   The Configuration Properties dialog displays, as shown in Figure 5–8.

Figure 5–8 Sequences Configuration Properties Window



Configuration Properties: REPCAT$_FLAVORS_S [Rea...

REPCAT$_FLAVORS_S	
Sequence Parameters	
Increment By	1
Start With	1
Identification	
Name	
Previous Deployment N...	
Deployable	true

2. Configure the following properties for a sequence:

   Increment By: The number you want to increment your sequence by.

   Start With: The number you want the sequence to start with.

3. Configure the following Identification parameters:

   Deployable: Select TRUE to indicate that you want to deploy this sequence. Warehouse Builder only generates scripts for sequences marked deployable.

For related information, see the following sections:

- "About Sequences" on page 3-31
- "Creating a Sequence Definition" on page 3-31

Part II

Designing ETL Objects

This part contains the following chapters:

6

Designing Mappings

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

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

About Mappings

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. For information on how to create and define mappings in OMB Plus, see the Oracle Warehouse Builder Scripting Reference .

About Oracle Warehouse Modules

Before you can begin designing mappings, you must define an Oracle warehouse module. Warehouse Builder contains several types of modules. However, the Oracle warehouse module is the only module that stores mapping logic. For more information about Oracle warehouse modules, see "Creating Warehouse Modules" on page 3-1.

Procedure for Defining Mappings

When you define a mapping, you create a container that holds the operators defining the ETL logic. To define a mapping, complete the following steps:

1. Creating a Mapping on page 6-2

2. Adding Operators on page 6-6
3. Editing Operators on page 6-9
4. Connecting Operators on page 6-14
5. Setting Operator Properties on page 6-20
6. Configuring Mappings Reference on page 11-1
7. Reconciling Operators and Repository Objects on page 6-28

After you define the mapping you can validate the code for the mapping. For information, see Chapter 12, "Validating Objects". You can also interrelate mappings by creating process flows. For more information, see Chapter 10, "Designing Process Flows".

Creating a Mapping

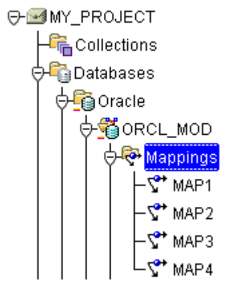
The first step in defining a mapping is to create a mapping using the Mapping Wizard.

To create a mapping:

1. Navigate to the **Mappings** node in a project. The **Mappings** node is located under an Oracle warehouse module which is contained under the Oracle folder under the Databases node.

   Figure 6–1 shows the Mappings node containing maps MAP1, MAP2, MAP3, and MAP4. The Oracle warehouse module in this example is named ORCL_MOD.

Figure 6–1 Mappings Node on the Navigation Tree



2. Right-click **Mappings** and then select **Create Mapping.**

   Warehouse Builder opens the **New Mapping Wizard.**
3. Click **Next.**

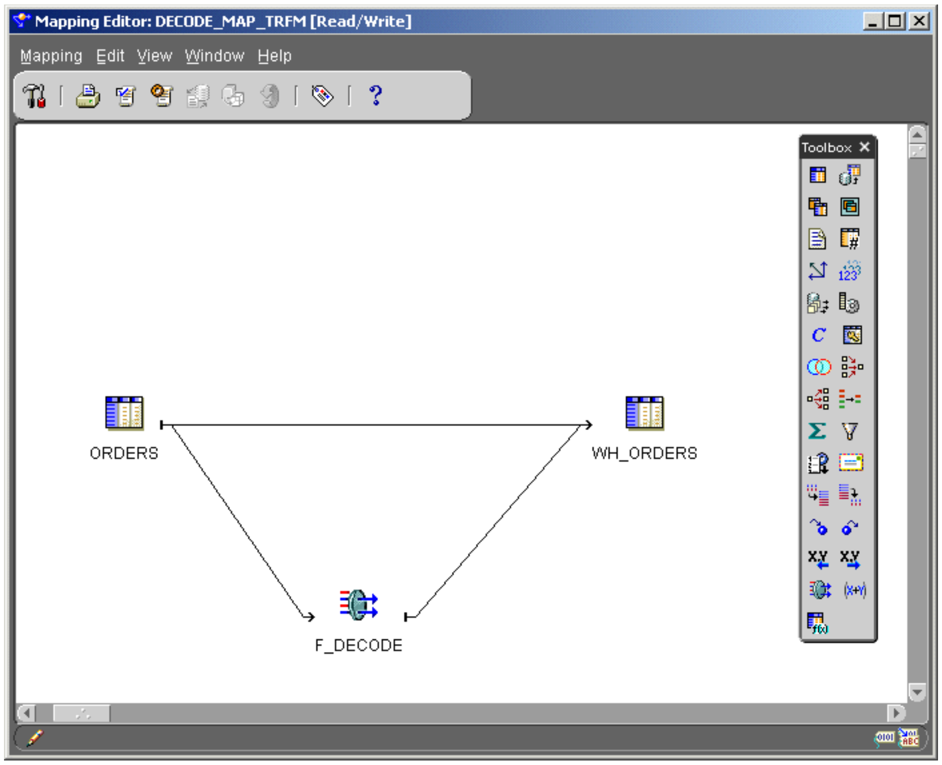
   The New Mapping Wizard displays the Name page.
4. Enter a name and description for the new mapping.
5. Click **OK.**

   Warehouse Builder stores the definition for the mapping and inserts its name in the warehouse module navigation tree. Warehouse Builder opens a **Mapping Editor** for the mapping you created. The Mapping Editor displays the name of the mapping in the title bar.

About the Mapping Editor

The Mapping Editor is the interface for designing, and editing mappings. The Mapping Editor includes a variety of operators that you add and connect to design a mapping. Figure 6–2 shows the Mapping Editor with three connected operators.

Figure 6–2 Mapping Editor Canvas


Mapping Editor: DECODE_MAP_TRFM [Read/Write]
Mapping Edit View Window Help
Toolbox
ORDERS
WH_ORDERS
F_DECODE

The Mapping Editor has the following components:

- **Menu Bar:** The menu bar provides access to the Mapping Editor commands.
- **Toolbar:** The toolbar provides access to commonly used commands.
- **Toolbox:** The toolbox contains operator icons. To include an operator, drag an operator icon on to the Mapping Editor canvas.
- **Mapping Editor Canvas:** The canvas provides the work space where you can create and modify mappings.

To open the Mapping Editor:

1. From the navigation tree, locate an Oracle warehouse module.

   These modules are located under the Oracle database folder in a project or collection, as shown in Figure 6–1 on page 6-2. If your project or collection does not have a warehouse module, create one using the instructions in "Creating Warehouse Modules" on page 3-1.

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 **Object** menu, select **Editor.**

- Select a mapping and type **Ctrl** and **O.**
- Right-click a mapping, and select **Editor...**

About Operators

The basic design element for a mapping is the operator. When you design a mapping in Warehouse Builder, you select operators from the Mapping Editor toolbox and drag them onto the canvas.

In Warehouse Builder, there are the following types of operators:

- **Source and Target Operators**: Source and target operators represent relational database objects and flat file objects in the mapping. For details on using SAP source data, see "Using SAP R/3 Data in Warehouse Builder".
- **Data Flow Operators**: Data flow operators transform data.

Source and Target Operators

Use source and target operators to represent relational database objects and flat file objects. Table 6–1 lists each source and target operator alphabetically, gives a brief description, and shows the associated icon in the Mapping Editor.

Table 6–1 Source and Target Operators

Icon	Operator	Description
	Mapping Advanced Queue Operator	Represents an Advanced Queue you previously imported.
	Mapping Cube	Represents a cube that you previously defined.
	Mapping Dimension	Represents a dimension that you previously defined.
	Mapping External Table	Represents an external table that you previously defined or imported.
	Mapping Flat File Operator	Represents a flat file that you previously defined or imported.
	Mapping Materialized View	Represents a materialized view that you previously defined.
	Mapping Table	Represents a table that you previously defined or imported.
	Mapping View	Represents a view that you previously defined or imported.

Data Flow Operators

Use data flow operators to transform data in a mapping. Table 6–2 lists each data flow operator alphabetically, gives a brief description, and shows the associated icon in the Mapping Editor. For more information on operators, see Chapter 8, "Using Mapping Operators".

Table 6–2 Data Flow Operators

Icon	Operator	Description
Σ	Aggregator Operator	Performs data aggregations, such as SUM and AVG, and provides an output row set with aggregated data.
C	Constant Operator	Produces a single output group that can contains one or more constant attributes.
	Data Generator Operator	Provides information such as record number, system date, and sequence values.
	Deduplicator Operator	Removes duplicate data in a source by placing a DISTINCT clause in the select code represented by the mapping.
(X+Y)	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.
X,Y	Mapping Input Parameter Operator	Passes parameter values into a mapping.
X,Y	Mapping Output Parameter Operator	Sends values out of a mapping.
123	Mapping Sequence Operator	Generates sequential numbers that increment for each row.
	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.
	Post-Mapping Process Operator	Calls a function or procedure after executing a mapping
	Pivot Operator	Calls a function or procedure prior to executing a mapping
	Data Generator 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.
f(x)	Post-Mapping Process 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.

Table 6–2 (Cont.) Data Flow Operators

Icon	Operator	Description
	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.

Adding Operators

For every source and target operator you add, Warehouse Builder creates and maintains a version of that object in the Warehouse Builder repository and a separate version for the Mapping Editor. For example, when you add a table to a mapping, Warehouse Builder maintains a copy of the table in the repository. Refer to it as the *repository table*. Refer to the table in the mapping as the *mapping table*.

In addition to maintaining separate repository objects and mapping objects for sources and targets, Warehouse Builder also maintains separate repository objects for the following data flow operators:

- Key LookUps
- Mapping Input Parameters
- Mapping Output Parameters
- Mapping Sequences
- Transformations

Warehouse Builder maintains separate repository objects so that you can *reconcile* changing definitions of these objects. For example, as you make changes to a mapping table, you may want to propagate those changes back to the repository table associated with it. Conversely, if you import a new metadata definition for the repository table, you may want to propagate those changes to the mapping table. You can accomplish these tasks by a process known as reconciliation. When you reconcile, you *bind* one object definition to another. For more information on binding objects, see "Reconciling Operators and Repository Objects" on page 6-28.

The concept of binding a mapping operator to a repository object is important because it affects how you add operators to a mapping.

To add an operator to a mapping:

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

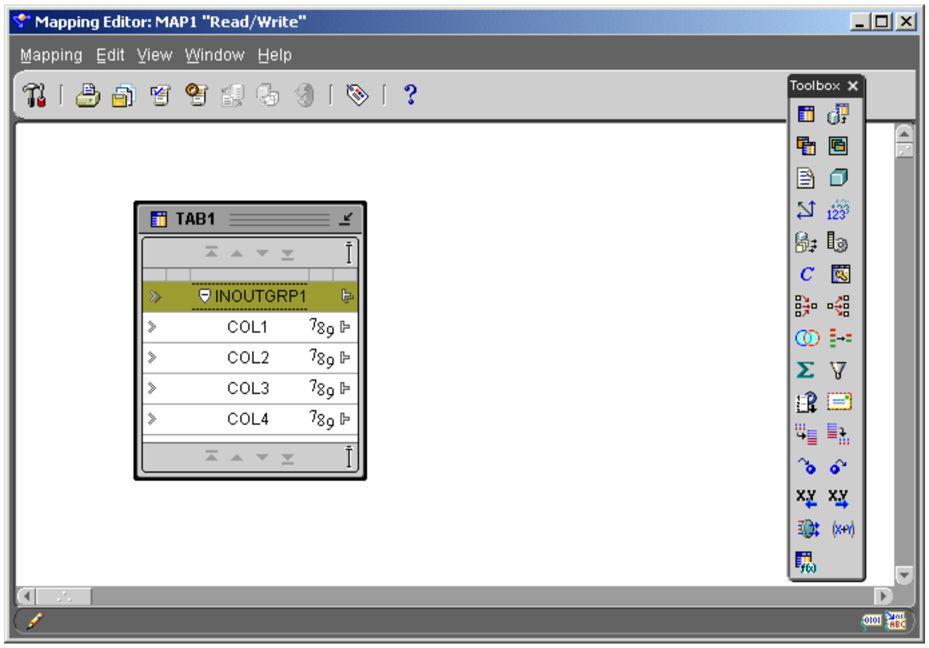
   If you select an operator that you can bind to a repository object, the Mapping Editor displays the **Add Mapping <operator name>** dialog. For details on how to use this dialog, see "Adding Bindable Operators" on page 6-7.

   If you select an operator that you cannot bind to a repository object, Warehouse Builder may display a wizard or dialog to assist you in creating the operator. For more information, refer to Chapter 8, "Using Mapping Operators", which provides an alphabetical listing of each operator and how to use them when designing mappings.

3. Follow any prompts Warehouse Builder displays and click **OK.**

   The Mapping Editor displays the operator maximized on the canvas as shown in Figure 6–3. You can view each attribute name and data type. The operator name appears in the upper left corner. If you want to minimize the icon, click the arrow in the upper right corner.

Figure 6–3 Mapping Editor Showing a Mapping Table Operator Source


Mapping Editor: MAP1 "Read/Write"
Mapping Edit View Window Help
Toolbox
TAB1
INOUTGRP1
COL1
COL2
COL3
COL4

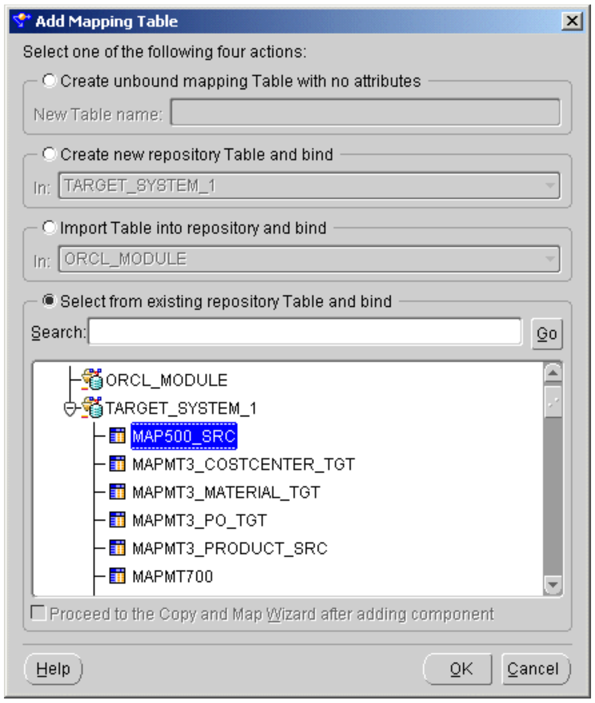
Adding Bindable Operators

Use this section for information on adding bindable operators to a mapping. Bindable operators include the following operators

- Mapping Tables
- Mapping External Tables
- Mapping Flat Files
- Mapping Dimensions
- Mapping Advanced Queues
- Mapping Cubes
- Mapping Views
- Mapping Materialized Views
- Key LookUps
- Mapping Sequences
- Transformations
- Pre Mapping Process
- Post Mapping Process

When you add an operator that you can bind to a repository object, the Mapping Editor displays the **Add Mapping <operator name>** dialog. Figure 6–4 shows the dialog for adding a mapping table operator.

Figure 6–4 Add Mapping Table Dialog


Add Mapping Table
Select one of the following four actions:
Create unbound mapping Table with no attributes
New Table name:
Create new repository Table and bind
In: TARGET_SYSTEM_1
Import Table into repository and bind
In: ORCL_MODULE
Select from existing repository Table and bind
Search:
Go
ORCL_MODULE
TARGET_SYSTEM_1
MAP500_SRC
MAPMT3_COSTCENTER_TGT
MAPMT3_MATERIAL_TGT
MAPMT3_PO_TGT
MAPMT3_PRODUCT_SRC
MAPMT700
Proceed to the Copy and Map Wizard after adding component
Help
OK
Cancel

Select one of the four options. Depending on the type of operator you select, some of the options may be greyed out.

- Create Unbound Object with No Attributes
- Create New Repository Object and Bind
- Import Object into Repository and Bind
- Select from Existing Repository Object and Bind

Create Unbound Object with No Attributes

Use this option when you want to use the Mapping Editor to define a new object. After you select **Create Unbound Object with No Attributes,** type a name for the new object. Warehouse Builder displays the operator on the canvas without any attributes. For an example on how to quickly create a staging table using an unbound table operator and the Auto-Mapping dialog, see "Example: Using the Mapping Editor to Create Staging Area Tables" on page 6-16.

Create New Repository Object and Bind

Use this option when you want to invoke a wizard to create a new object. First select a target module and then Warehouse Builder launches a wizard that assists you in creating the new operator. After you complete the wizard, Warehouse Builder saves a copy of the operator in the repository. For example. if you select this option when adding a mapping table, Warehouse Builder launches the New Table Wizard, creates the mapping table, creates the repository table, and binds the mapping table to the repository table.

Import Object into Repository and Bind

Use this option when you want to create an operator based on an object that you can import into the repository. Select a module into which you import the object.

If you select a module without a database link defined, Warehouse Builder displays the New Database Link Information dialog. See "Configuring Connections for Database Sources" on page 4-3 for information on how to define a new database link.

If you select a module with a defined database link, Warehouse Builder displays the Import Wizard. Enter the appropriate information in the dialog.

Select from Existing Repository Object and Bind

Use this option when you want to add an operator to the mapping based on an object you previously defined or imported into the repository.

Either type the prefix to search for the object or select from the displayed list of objects within the selected module.

To select multiple items, press the Control key as you click each item. To select a group of items located in a series, click the first object in your selection range, press the Shift key, and then click the last object.

You can add operators based on repository objects within the same module as the mapping or from other modules. If you select a repository object from another module, the Mapping Editor creates a connector if one does not already exist. The connector establishes a path for moving data between the mapping location and the location of the repository object. For more information about locations and connectors, see "Defining Locations" on page 3-2.

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.

To specify loading properties and conditional behaviors, use the properties windows. For information on properties windows, see "Mapping Naming Conventions" on page 6-11.

To edit an operator, group, or attribute:

1. Select an operator from the Mapping Editor canvas.

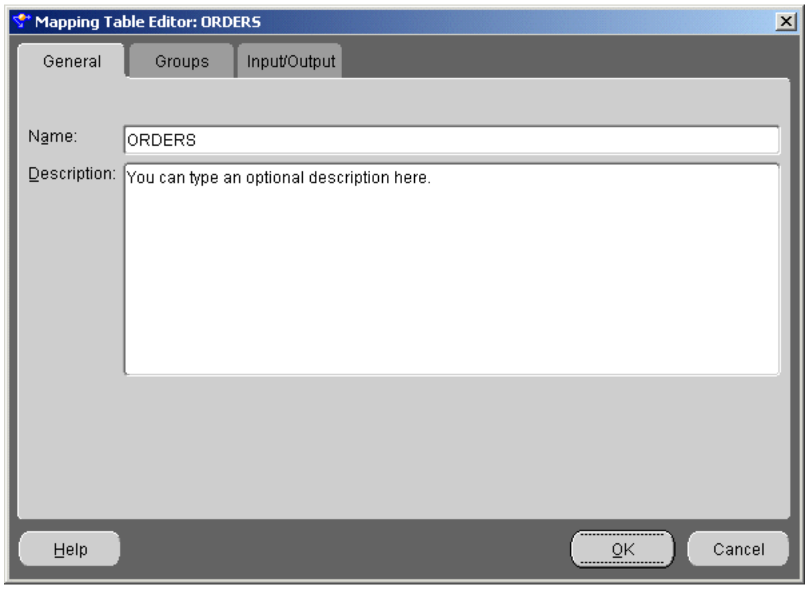
   Or select any group or attribute within an operator.

2. Right-click and select **Edit.**

   The Mapping Editor displays the operator editor with a General tab, Groups tab, and a tab for each type of group in the operator. Figure 6–5 shows the General tab on the operator editor.

   The General 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 "Naming Conventions for Operators" on page 6-11.

Figure 6–5 General Tab on the Operator Editor



Name: ORDERS

Description: You can type an optional description here.

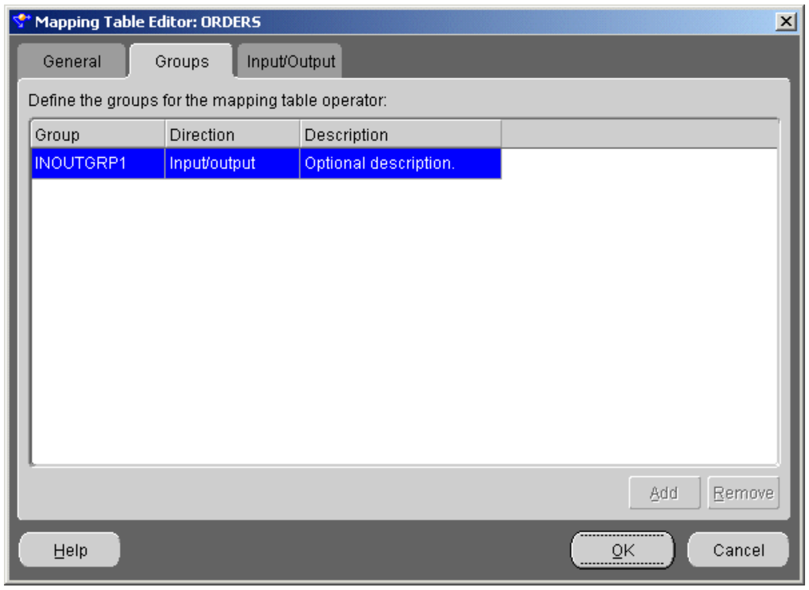
3. Edit group information on the Groups tab as shown in Figure 6–6.

   Each group has a name, direction, and optional description. You can rename the group but you cannot change the group direction. 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.

   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.

Figure 6–6 Groups Tab on the Operator Editor



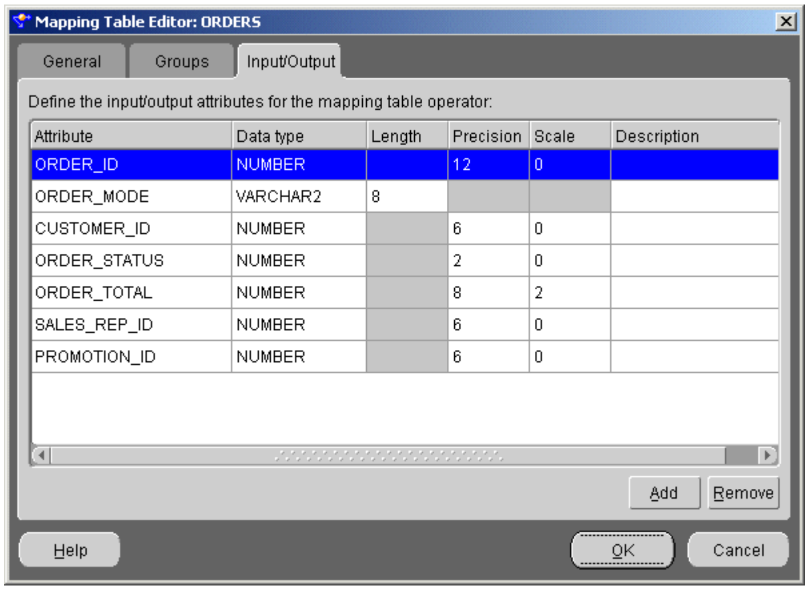
Define the groups for the mapping table operator:

Group	Direction	Description
INOUTGRP1	Input/output	Optional description.

4. Edit attribute information on the each of the remaining tabs.

Figure 6–7 shows an Input/Output tab on the Operator Editor. In this example, the operator is a table and therefore has only the Input/Output tab. Other operators can have an Input tab and an Output tab.

Figure 6–7 Input/Output Tab on the Operator Editor



Define the input/output attributes for the mapping table operator:

Attribute	Data type	Length	Precision	Scale	Description
ORDER_ID	NUMBER		12	0	
ORDER_MODE	VARCHAR2	8			
CUSTOMER_ID	NUMBER		6	0	
ORDER_STATUS	NUMBER		2	0	
ORDER_TOTAL	NUMBER		8	2	
SALES_REP_ID	NUMBER		6	0	
PROMOTION_ID	NUMBER		6	0	

You can add, remove, and edit attributes. The Mapping Editor greys out properties that you cannot edit. For example, if the data type is NUMBER, you can edit the precision and scale but not the length.

To assign correct values for datatype, length, precision, and scale in an attribute, follow PL/SQL rules. When you reconcile the operator, Warehouse Builder checks the attributes based on SQL rules.

Mapping Naming Conventions

When you name or rename objects in the Mapping Editor, use the following naming conventions.

Naming Conventions for Attributes and Groups

Attribute and group names are logical. Although the attribute names of the object are often the same as the attribute names of the operator, their properties remain independent of the attributes of the operator to which they are bound. This protects any expression or use of an attribute from corruption if it is manipulated within the operator. You can rename groups and attributes independent of their sources.

Naming Conventions for Operators

When you add operators to a mapping, the Mapping Editor displays the physical name or the business name of the operator, depending on which naming mode you selected in the project Preferences window. See "Naming Preferences" on page 2-17 for details on how to specify a naming mode.

Operator names display as business names if you select the business naming mode in the project Preferences window. When you add an operator to a mapping while working in the business naming mode, the Mapping Editor generates a default business name for the operator. You can change the business name to any name that meet the following requirements:

- The length of the operator name can be any string of 200 characters. You must use double quotes for names containing spaces.
- The operator name must be unique on its attribute group, attribute and display set level with respect to its parent.

Operator names display as physical names if you select the physical naming mode in the Project Preferences window. When you use the Warehouse Builder OMB Scripting Language, you navigate to operators based on their physical names. Because the physical name is used during scripting, physical names must meet the following requirements:

- The operator name must be unique on its group, attribute and display set level with respect to its parent.
- The operator name must contain no more than 28 characters. Two characters are used by Warehouse Builder.
- The operator name must conform to the syntax rules for basic elements as defined in the Oracle Database SQL Reference.

In addition to physical and business names, operators also have bound names. Bound names are available as a logical and physical property to operators that can be bound. They are used to reference the object during code generation and 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 perform an outbound reconciliation.
- When you rename the logical name for an operator or attribute, Warehouse Builder propagates the new logical name as the bound name when you perform an outbound reconciliation. However, logical 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 logical 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, the operators contain a display set named ALL that shows all attributes contained in the repository object represented by the operator.

Operators also contain a default display set named MAPPED that includes only those attributes in a group that are connected to another operator in a mapping. The MAPPED display set updates automatically whenever attribute connections are changed in the Mapping Editor. Table 6–3 describes the default Display Sets.

Table 6–3 Default Sets

Display Set	Description
All	Includes all attributes in an operator.

Table 6–3 (Cont.) Default Sets

Display Set	Description
Mapped	Includes only those attributes in an operator that are connected to another operator.
Hierarchies	For each hierarchy in a Dimension, a display set containing all the level attributes in that hierarchy.

See "Adding Attribute Sets" on page 3-17 for information on attribute sets. See "Editing Operators" on page 6-9 for information on operator attributes and groups.

Defining Display Sets

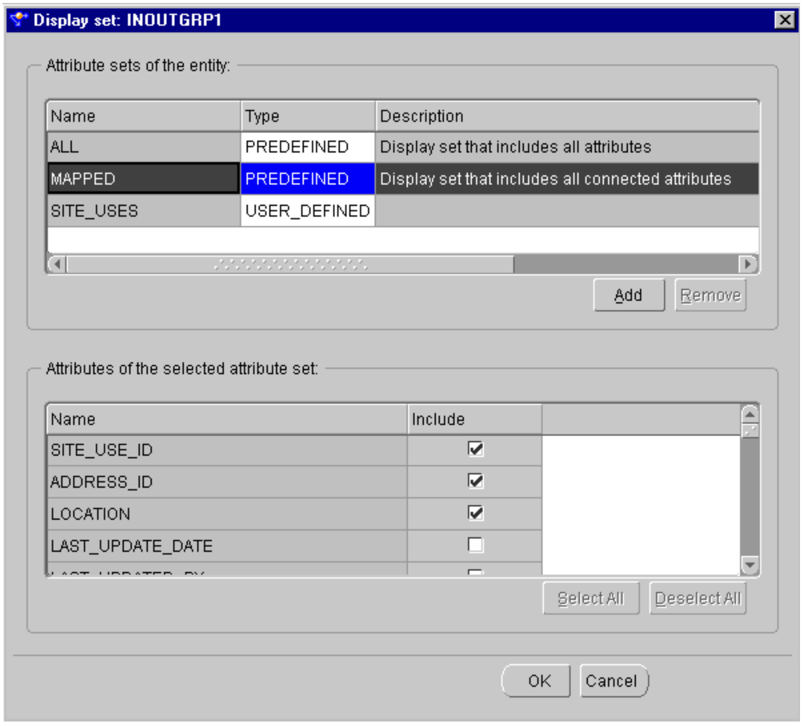
You can define display sets for any operator in a mapping.

To define a display set:

1. Right-click a group in an operator, and select Display Set.

   The Display Set dialog appears, as shown in Figure 6–8.

Figure 6–8 Display Set Dialog



Display set: INOUTGRP1

Attribute sets of the entity:

Name	Type	Description
ALL	PREDEFINED	Display set that includes all attributes
MAPPED	PREDEFINED	Display set that includes all connected attributes
SITE_USES	USER_DEFINED	

Attributes of the selected attribute set:

Name	Include
SITE_USE_ID	✓
ADDRESS_ID	✓
LOCATION	✓
LAST_UPDATE_DATE	☐

2. Click **Add.**
3. Type a name in the Name column and press **Enter.**

   All available attributes for the operator appear in **Attributes of the selected attribute set field.** 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. Select Use Display Set and select the display set.

Compressing Mappings

When you add a source or target operator to a mapping, all of the columns from the physical source or target are added as attributes to the operator. Some of these attributes may be unnecessary in the mapping. By compressing a mapping, any source or target mapping attribute that is not connected to any other attribute is deleted from the mapping. This simplifies the display of the mapping and improves performance when importing and exporting MDL files.

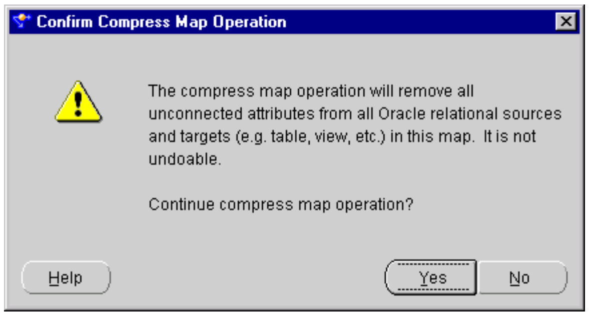
After a mapping is compressed, you can only undo the compression by performing an inbound reconciliation. Once you reconcile outbound, the compressed mapping is saved in the repository and cannot be rolled back.

To compress a mapping:

1. Open a mapping in the Mapping Editor.
2. From the **Mapping** menu, select **Compress Mapping.**

   A warning message displays stating that unconnected Oracle relational source and target attributes will be deleted, as shown in Figure 6–9.

Figure 6–9 Confirm Compress Map Operation Dialog



The compress map operation will remove all unconnected attributes from all Oracle relational sources and targets (e.g. table, view, etc.) in this map. It is not undoable.

Continue compress map operation?

3. Click **Yes** to continue. Any source or target mapping attribute that is not connected to any other attribute is deleted from the mapping except for unconnected attributes that are part of an update or delete condition and flat file sources.

Connecting Operators

After you have selected mapping source operators, data flow operators, 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.
- **Connecting Groups:** When you connect groups, you can control which attributes are connected using the Auto-Mapping dialog.

The position of the attribute where you start your mapping and the position where you release the mouse button determines the type of data flow connection you make in the mapping.

Connecting Attributes

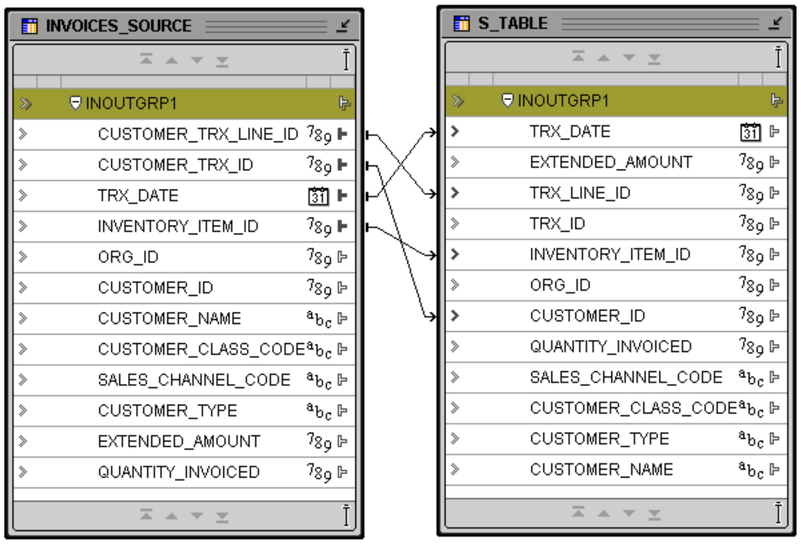
You can draw a line from a single output attribute of one operator to a single input attribute of another operator.

To connect attributes:

1. Click and hold down the mouse button while the pointer is positioned over an output attribute.
2. Drag the mouse away from the output attribute and toward the input attribute to which you want data to flow.

   As you drag the mouse, a line appears on the Mapping Editor canvas to indicate a connection, as shown in Figure 6–10.
3. Release the mouse over the input attribute.

Figure 6–10 Connected Operators in a Mapping



Repeat steps one through three until you have created all the data flow connections appropriate for your situation.

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 input only attribute.
- You cannot connect into an output only attribute.

- You cannot create mapping lines that contradict an established cardinality. Instead, use a Join operator.

Connecting Groups

When you connect groups, the Mapping Editor assists you by either automatically copying the attributes or prompts you for more information in the Auto-Mapping dialog described in "Using the Auto-Mapping Dialog" on page 6-17.

If you connect to a target group with no existing attributes, the Mapping Editor automatically copies the attributes and creates the mapping lines. This is useful for designing mappings such shown in "Example: Using the Mapping Editor to Create Staging Area Tables".

Example: Using the Mapping Editor to Create Staging Area Tables

You can use the Mapping Editor with an unbound table operator to quickly create staging area tables.

The following instructions describe how to create a staging table based on an existing source table. You can also use these instructions to create views, materialized views, flat files, and transformations.

To map a source table to a staging table:

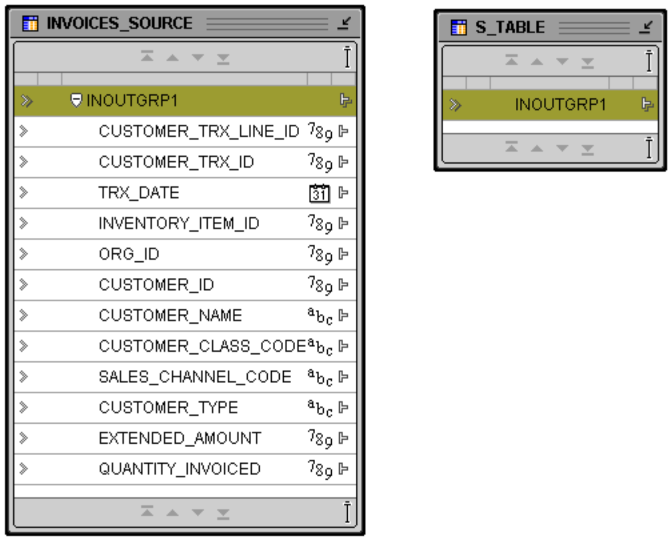
1. In the Mapping Editor, add the source table.

   From the menu bar, select **Mappings** and **Add.** Use the **Add Mapping Table** dialog to select and bind the source table operator in the mapping.

2. Add a new unbound table operator.

   From the menu bar, select **Mappings** and **Add.** From the **Add Mapping Table** dialog, select **Create unbound mapping table with no attributes.** The mapping should now resemble Figure 6–11 with one source table and one staging area table without attributes.

Figure 6–11 Unbound Staging Table without Attributes and Source Table



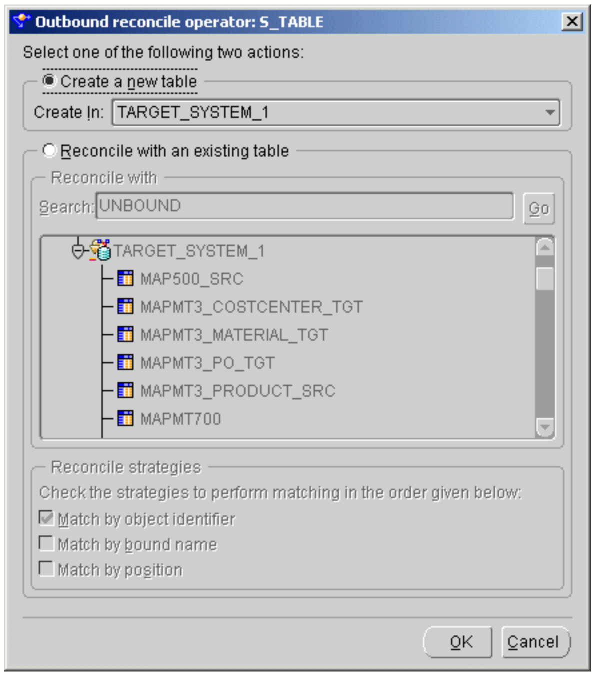
3. With the mouse button positioned over the group in the operator for the source table, click and hold down the mouse button.

4. Drag the mouse to the staging area table group.

   Warehouse Builder copies the source attributes to the staging area table and creates the corresponding mapping lines.

5. In the Mapping Editor, select the unbound table you added to the mapping. Right click and select **Reconcile Outbound.** Warehouse Builder displays the Outbound Reconcile Operator dialog as shown in Figure 6–12.

Figure 6–12 Outbound Reconcile Operator Dialog



Outbound reconcile operator: S_TABLE

Select one of the following two actions:

[✓ Create a new table]

Create In: TARGET_SYSTEM_1

Reconcile with an existing table

Reconcile with

Search: UNBOUND

TARGET_SYSTEM_1
- MAP500_SRC
- MAPMT3_COSTCENTER_TGT
- MAPMT3_MATERIAL_TGT
- MAPMT3_PO_TGT
- MAPMT3_PRODUCT_SRC
- MAPMT700

Reconcile strategies

Check the strategies to perform matching in the order given below:

[✓ Match by object identifier]

Match by bound name

Match by position

6. Select **Create a new table** and specify the target module in which to create the table.

   Warehouse Builder creates the new table in the target module you specify.

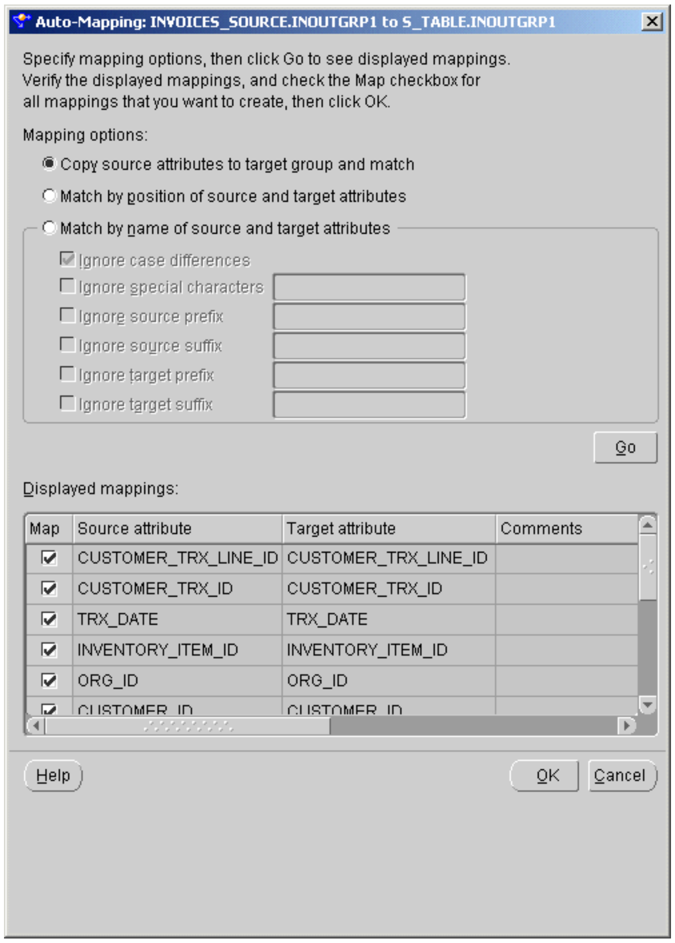
Using the Auto-Mapping Dialog

If you connect to a target with existing attributes, the Mapping Editor launches the Auto-Mapping dialog as shown in Figure 6–13.

Select one of the following rules for copying and creating mapping lines:

- Copy Source Attributes to Target Group with Existing Attributes
- Match by Position of Source and Target Attributes
- Match by Name of Source and Target Attributes

Figure 6–13 Auto-Mapping Dialog Displaying Attributes to be Mapped



Auto-Mapping: INVOICES_SOURCE.INOUTGRP1 to S_TABLE.INOUTGRP1

Specify mapping options, then click Go to see displayed mappings.
Verify the displayed mappings, and check the Map checkbox for
all mappings that you want to create, then click OK.

Mapping options:

[✓ Copy source attributes to target group and match]

Match by position of source and target attributes

Match by name of source and target attributes

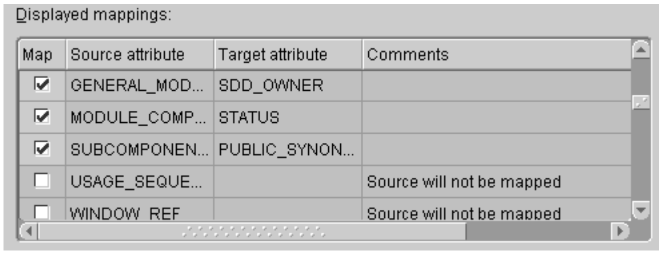
- ☑ Ignore case differences
- ☐ Ignore special characters
- ☐ Ignore source prefix
- ☐ Ignore source suffix
- ☐ Ignore target prefix
- ☐ Ignore target suffix

Displayed mappings:

Map	Source attribute	Target attribute	Comments
☑	CUSTOMER_TRX_LINE_ID	CUSTOMER_TRX_LINE_ID	
☑	CUSTOMER_TRX_ID	CUSTOMER_TRX_ID	
☑	TRX_DATE	TRX_DATE	
☑	INVENTORY_ITEM_ID	INVENTORY_ITEM_ID	
☑	ORG_ID	ORG_ID	
☑	CUSTOMER_ID	CUSTOMER_ID	

After you select one of the three options, select **Go.** The Auto-Mapping dialog displays a list of the source and target attributes to be mapped as shown in Figure 6–14.

Figure 6–14 Displayed Mappings



Displayed mappings:

Map	Source attribute	Target attribute	Comments
☑	GENERAL_MOD...	SDD_OWNER	
☑	MODULE_COMP...	STATUS	
☑	SUBCOMPONEN...	PUBLIC_SYNON...	
☐	USAGE_SEQUE...		Source will not be mapped
☐	WINDOW_REF		Source will not be mapped

You can deselect attributes by clearing the **Map** checkbox. The Auto-Mapping describes the results of your selections under **Comments.** See "About Auto-Mapping Dialog Comments" on page 6-19 for more information about these comments.

When you select **OK,** Warehouse Builder copies the source attributes to the target group and creates mapping lines between the source and target group.

Copy Source Attributes to Target Group with Existing Attributes

Use this option to copy source attributes to a target group that already contain attributes. Warehouse Builder creates mapping lines from the source attributes to the new target attributes based on the selections you make in the Auto-Mapping dialog. Warehouse Builder does not perform this operation on target groups that do not accept new input attributes such as dimension and cube target operators.

Match by Position of Source and Target Attributes

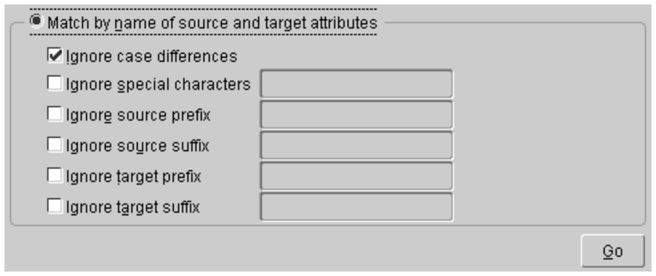
Use this option to create mapping lines between existing attributes based on the position of the attributes in their respective groups. Source and target attributes are matched in order, until all attributes for a target are matched. If a source operator contains more attributes than a target, then the remaining source attributes do not map to a target.

Match by Name of Source and Target Attributes

Use this option to create mapping lines between existing attributes with matching names, as shown in Figure 6–15. By selecting from the list of options, you can specify auto-mapping between names that do not match exactly. You can combine these options:

- **Ignore case differences:** Enables lower-case and upper-case characters to match. For example, the attributes FIRST_NAME and First_Name will match.
- **Ignore special characters:** Enables you to specify characters to ignore during the matching process. Enter the characters into the text field to the right of this option. For example, if you specify a hyphen and underscore, the attributes FIRST_ NAME, FIRST-NAME, and FIRSTNAME will all match.
- **Ignore source prefix, Ignore source suffix, Ignore target prefix, Ignore target suffix:** Enables you to 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 will match the target attribute FIRST_NAME.

Figure 6–15 Match by Name Matching Options


● Match by name of source and target attributes
[✓] Ignore case differences
[ ] Ignore special characters
[ ] Ignore source prefix
[ ] Ignore source suffix
[ ] Ignore target prefix
[ ] Ignore target suffix

After you set the matching criteria, click **Go.**

If you attempt to map a source group with no attributes, or if you are matching by name and there are no matches, an error dialog displays.

If one or more of the attributes can be matched, the **Displayed Mappings** field displays the matches. You can verify and deselect the mapping lines before they are implemented.

About Auto-Mapping Dialog Comments

In the **Displayed Mapping** area of the Auto-Mapping dialog, the **Comments** column contains information about the results from your matching criteria.

When a mapping results in duplicated attributes, the following messages appear:

- **Target was already mapped:** A target attribute can take part in only one mapping. This target attribute is already used in a mapping.
- **Target may not be double-mapped:** If name matching finds multiple matches to a target attribute, only one mapping can be created. The first mapping is selected, and all other mappings to that target attribute are not selected. Selecting one attribute deselects any other attribute.
- **Source may be double-mapped:** A source attribute can be mapped to different target attributes. If name matching finds multiple matches from a source attribute, all of these mappings can be created. You can then deselect the target attributes that you do not want to map to by clicking their check boxes.

When a source group has more attributes than a target group or a target group has more attributes than a source group, the following messages appear:

- **Source will not be mapped:** No target attributes are available for the source.
- **Target will not be mapped:** No source attributes are available for the target.

If you open the Auto-Mapping dialog while the Mapping Editor is in read-only mode, an error displays. You must have read/write permission before you can use auto-mapping. See "Supporting Multiple Users" on page 2-11 for more information on read/write permissions.

Setting Operator Properties

This section discusses how to set properties for the following operators in a mapping:

- Source and Target Operator Properties on page 6-21
- Attribute Properties on page 6-25
- Flat File Operators Properties on page 6-27

For information on setting other operator types, see the following sections:

- Chapter 21, "Using SAP R/3 Data in Warehouse Builder"
- Chapter 8, "Using Mapping Operators"

Each operator, group, and attribute has a properties window associated with it. Use properties windows to specify loading settings and other conditional settings. 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.
- **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. For information on how to set properties for groups, look up the operator by name in Chapter 8, "Using Mapping Operators".
- Attribute Properties**:** Properties that pertain to attributes in source and target operators. Examples of attribute properties are data type, precision, and scale.

  To add, delete, or rename attributes, use the operator editor. For information on the operator editor, see "Editing Operators" on page 6-9.

To open the properties windows:

1. Click one of the properties.

The properties window displays the name of the property you selected in the title bar.

The Mapping Editor displays a description for the property in a window at the bottom of the properties window. You can expand the window to view the description.

2. For objects that contain many properties, click the flashlight icon in the upper left corner and enter the property name to search for in **Find.**

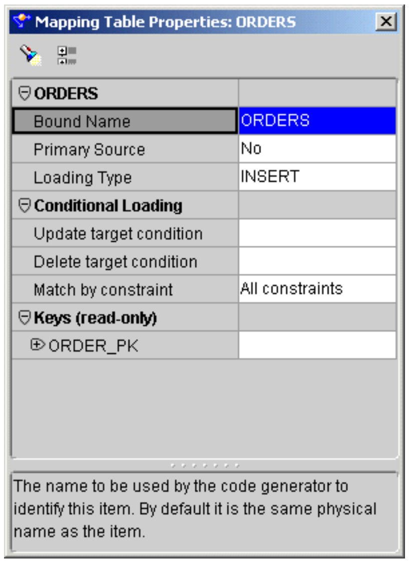
3. To sort the list alphabetically, click the plus icon in the upper left corner.

   By default, the properties window lists the properties by category.

4. Select an operator in the Mapping Editor and double-click it or right-click and select **Operator Properties.**

   The Mapping Editor displays the properties window for the object you select. Figure 6–16 shows the properties window for a mapping table operator.

Figure 6–16 Properties Window for a Mapping Table Operator



ORDERS	
Bound Name	ORDERS
Primary Source	No
Loading Type	INSERT
Conditional Loading	
Update target condition	
Delete target condition	
Match by constraint	All constraints
Keys (read-only)	
ORDER_PK	

The name to be used by the code generator to identify this item. By default it is the same physical name as the item.

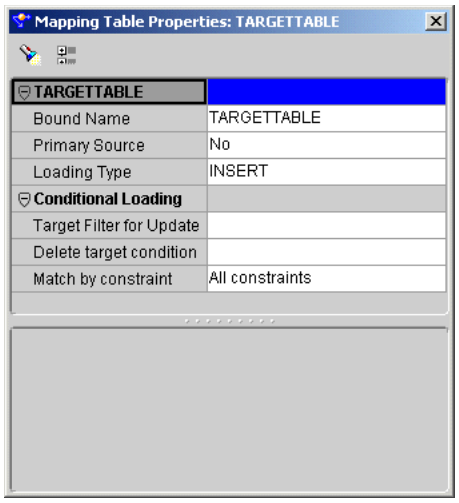
Source and Target Operator Properties

The properties window contains the following categories of parameters for source and target operators:

- **Operator Name:** Under the operator name, you can view the Bound Name, and set Primary Source indicator, and the Loading Type.
- **Conditional Loading:** You can set Target Filter for Update, Delete Target Condition, 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.

The operator properties window displays as shown in Figure 6–17.

Figure 6–17 Properties Window for Mapping Table Operator


Mapping Table Properties: TARGETTABLE

TARGETTABLE	
Bound Name	TARGETTABLE
Primary Source	No
Loading Type	INSERT
Conditional Loading	
Target Filter for Update	
Delete target condition	
Match by constraint	All constraints

Bound Name

The name used by the code generator. If an operator is currently bound and reconciled, 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 reconcile it to a repository object.

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.

- **INSERT:** The incoming row sets are inserted into the data target. Insert fails if a row already exists with the same primary or unique key.
- **UPDATE:** The incoming row sets are used to update existing rows in the data target. If no rows exist for the specified match conditions, no changes are made.
- **INSERT/UPDATE:** For each incoming row, an insert operation is performed first. If the insert fails, an update operation occurs. If there is no matching records for update, the insert is performed. If you select Insert/Update and the default mode is row based, you must set unique constraints on the target. If you select Insert/Update and the default operating mode is set based, Warehouse Builder generates a merge statement to Oracle Database.
- **UPDATE/INSERT:** For each incoming row, an update operation is performed first. If you select Update/Insert and the default operating mode for the target is set-based, Warehouse Builder generates a merge statement to Oracle Database.
- **DELETE:** The incoming row sets are used to determine which of the rows at the target get deleted.
- **TRUNCATE/INSERT:** The data target is truncated before the incoming row set is inserted into the data target. If you choose this option, the procedure cannot be rolled back even if the execution of the mapping fails.
- **DELETE/INSERT:** The rows in the data target are deleted before the incoming row set is inserted into the data target.

- **CHECK/INSERT:** The data target is checked to see if it contains any rows. If not, the incoming row sets are inserted into the data target.
- **None:** No operation is performed on the data target. This setting is useful for testing. Extraction and transformations run but have no effect on the target.

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 Database *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 6–4 lists the INTO TABLE clauses associated with each load type and their affect on data in the existing targets.

Table 6–4 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 Filter for Update

If the condition evaluates to true, the row is included in the update loading operation.

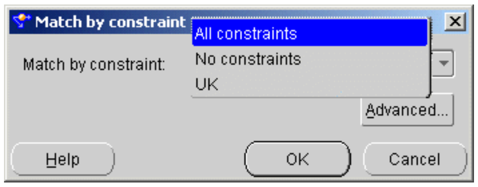
Delete Target Condition

If evaluated to true, the row is included in the delete loading operation.

Match By Constraint

When loading target operators with the UPDATE or the DELETE conditions, you can specify matching criteria. You can set matching and loading criteria manually or choose from several built-in options. Use Match By Constraint to indicate whether unique or primary key information on a target overrides the manual matching and loading criteria set on its attributes. When you click the property Match By Constraint, Warehouse Builder displays a drop down box listing the constraints defined on that operator and the built-in loading options, as shown in Figure 6–18.

Figure 6–18 Match By Constraint Options for Operators



If you select **All Constraints**, all manual attribute load settings are overruled and the data is loaded as if the load and match properties of the target attributes were set as displayed in Table 6–5.

Table 6–5 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.

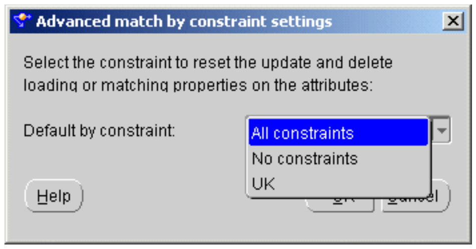
If you select a constraint previously defined for the operator, all manual attribute load settings are overruled and the data is loaded as if the load and match properties of the target were set as displayed in Table 6–7.

Table 6–6 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

If you made changes at the attribute level and you want to default all settings, click **Advanced.** Warehouse Builder displays a drop down box listing the loading options as shown in Figure 6–19. Warehouse Builder defaults the settings based on the constraint type you select.

Figure 6–19 Advanced Settings for Match By Constraint


Advanced match by constraint settings

Select the constraint to reset the update and delete loading or matching properties on the attributes:

Default by constraint: All constraints

No constraints

UK

For example, if you want to reset the match properties for all key attributes, click **Advanced,** select **No Constraints,** and click **OK.** Warehouse Builder overwrites the manual load settings and loads the data based on the settings displayed in Table 6–7.

Table 6–7 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,** Warehouse Builder overwrites the manual load settings and loads the data based on the settings displayed in Table 6–8

Table 6–8 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

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.

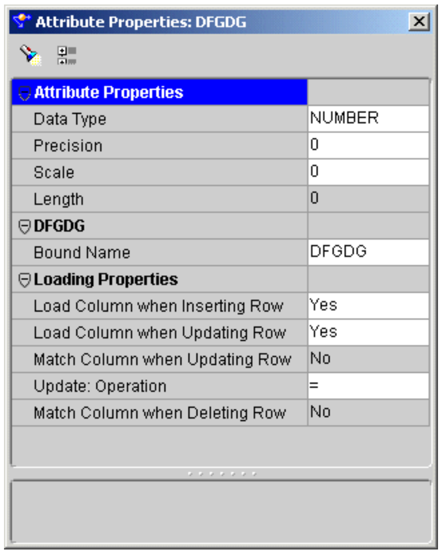
Referenced Keys
If the operator contains a foreign key, **Referenced Keys** displays the primary key or unique key for the referenced object.

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 and set the Data Type, Precision, Scale, and Length.
- **Loading Properties:** The mapping table, mapping dimension, mapping cube, mapping view, and mapping materialized view operators 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.

Figure 6–20 shows the attribute parameters for mapping tables, mapping dimensions, mapping views, and mapping materialized views.

Figure 6–20 Operator Attribute Properties Window



Attribute Properties	
Data Type	NUMBER
Precision	0
Scale	0
Length	0
DFGDG	
Bound Name	DFGDG
Loading Properties	
Load Column when Inserting Row	Yes
Load Column when Updating Row	Yes
Match Column when Updating Row	No
Update: Operation	=
Match Column when Deleting Row	No

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 unbound.

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.

Load Column When Inserting Row
This setting prevents data from moving to a target even though it is mapped to do so. If you select **Yes** (default), the data will reach the mapped target.

Load Column When Updating Row

This setting prevents the selected attribute data from moving to a target even though it is mapped to do so. If you select **Yes** (default), the data reaches the mapped target attribute. If all columns of a unique key are not mapped, then the unique key is not used to construct the match condition. If no columns of a unique key are mapped, Warehouse Builder displays an error. If a column (not a key column) is not mapped, then it is not used in loading.

Match Column When Updating Row

This setting updates a data target row only if there is a match between the source attribute and mapped target attribute. If a match is found, then an update occurs on the row. If you set this property to **Yes** (default), the attribute is used as a matching attribute. If you use this setting, then all the key columns must be mapped. If there is only one unique key defined on the target entity, use constraints to override this setting.

Update: Operation

You can specify an update operation to be performed when Warehouse Builder locates a matching row and updates the target. An update operation is performed on the target attribute using the data of the source attribute. Table 6–9 lists the update operations you can specify and describes the update operation logic.

Table 6–9 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)

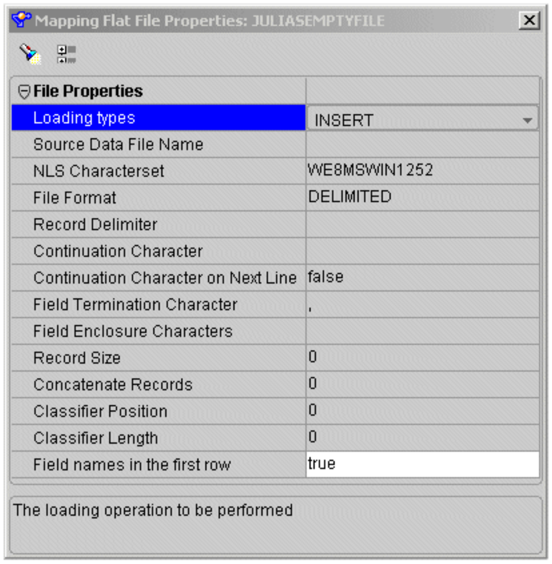
Match Column When Deleting Row

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.

Flat File Operators Properties

You can set properties for a flat file operator as either a source or target. You can set Loading Types 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 6–21.

Figure 6–21 Properties Window for Mapping Flat File Operator



File Properties	
Loading types	INSERT
Source Data File Name	
NLS Characterset	WE8MSWIN1252
File Format	DELIMITED
Record Delimiter	
Continuation Character	
Continuation Character on Next Line	false
Field Termination Character	,
Field Enclosure Characters	
Record Size	0
Concatenate Records	0
Classifier Position	0
Classifier Length	0
Field names in the first row	true

The loading operation to be performed

Loading Types

Select a loading type from the drop-down list:

- **Insert:** Creates a new target file. If a target file already exists, then it is replaced with a new target file.
- **Update:** Creates a new target file. If a target file already exists, then it is appended.
- **None:** No operation is performed on the data target. This setting is useful for test runs where all transformations and extractions are run but have no effect on the target.

Field Names in the First Row

Set this property to **True** if you want to write the field names in the first row of the operator or **False** if you do not.

Reconciling Operators and Repository Objects

For every operator you define in a mapping, there can be a corresponding definition in the Warehouse Builder repository. When you make changes to an operator in a mapping, you may want to reconcile the operator and its corresponding repository definition.

When you reconcile operators, you ensure that the operator matches the repository object it represents. You can reconcile in one of two ways:

- **Inbound Reconciliation:** This updates or reconciles the operator with the definition of a specified repository object.
- **Outbound Reconciliation:** This updates a selected repository object to reflect changes you made to the operator in a mapping.

Reconciliation is different from synchronization. Synchronization ensures that you are up-to-date with changes made by other users in a multiuser environment. Reconciliation updates operators with their corresponding repository objects.

Inbound Reconciliation

Inbound reconciliation updates the operator with the definition of a specified repository object. You can also use Inbound Reconcile to bind an unbound operator to a repository object. You can reconcile an operator by name, position, object identifier match, or any combination of these methods.

You can use inbound reconciliation for any of the following reasons:

- **Propagate changes:** Use inbound reconciliation to propagate changes you made in a repository object to its associated operator. Changes to the repository object can include structural changes, attribute name changes, attribute data type changes.
- **Update an operator based on a new repository object:** You can associate an operator with a new repository object. For example, if you are migrating mappings from one version of a data warehouse to a newer version and you maintain different object definitions for each version. Or, if you want to provide access to the underlying source data through views rather than directly against the base tables.
- **Prototype mappings using tables:** When you are satisfied with the transformation logic, you can switch to other object types for production mappings such as views, materialized views, or cubes.

Unless you remove attributes, inbound reconciliation has no impact on the dependent relationship between other operators in the mapping to repository objects. Warehouse Builder preserves those dependencies.

When you perform inbound reconciliation on an external table operator, Warehouse Builder updates the operator based on the repository external table only and not its associated flat file. To update an external table based on its associated flat file, see "Reconciling an External Table Definition with a Record in a File" on page 3-22.

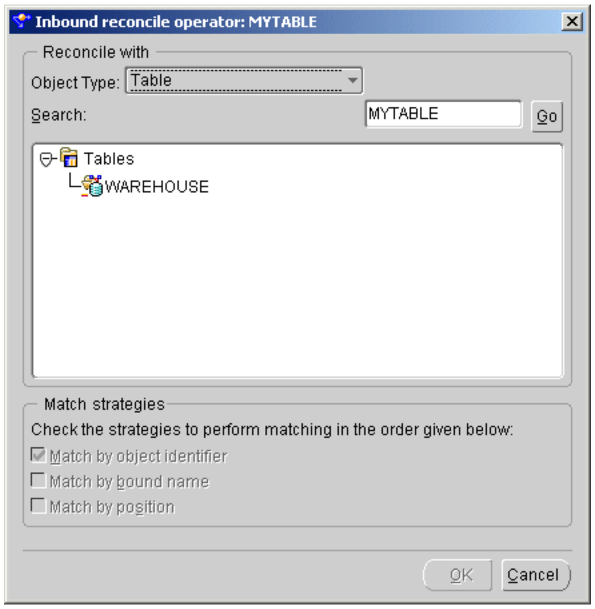
Reconciliation also updates additional logical properties of an operator. For example, for a Mapping Flat File operator, information about the character set and filename are refreshed.

To perform inbound reconciliation on an operator:

1. Select an operator on the Mapping Editor canvas.
2. From the **Edit** menu, select **Reconcile Inbound** or right-click the header of the operator and select **Reconcile Inbound.**

   The **Inbound Reconcile Operator** dialog displays as shown in Figure 6–22.

Figure 6–22 Inbound Reconcile Operator Dialog


Inbound reconcile operator: MYTABLE

Reconcile with

Object Type: Table

Search: MYTABLE Go

Tables
WAREHOUSE

Match strategies

Check the strategies to perform matching in the order given below:

- [✓] Match by object identifier
- [ ] Match by bound name
- [ ] Match by position

OK Cancel

3. From **Reconcile with,** select the type of repository object to use to update the operator. The types of objects available depends on the operator type. See Table 6–10 on page 6-30 for a list of available objects.
4. Set the Match Strategies.

   Matching by Object Identifier: Use this strategy if you want to keep your operators consistent with changes to the previously bound repository object. Match by object identifier is not available if you want to reconcile to a different repository object.

   Matching by Bound Name: Use this strategy if you want to maintain equivalence of operator bound names and object physical names. You can use this strategy with a different repository object if there are changes that alter the structure of the operator.

   Matching by Position: Use this strategy for reconciliation with a different repository object if you want to preserve the business names of your operator attributes. This strategy is most effective when the only changes to the repository object are the addition of columns, fields, or parameters at the end of the object.

   For more information, see "Match Strategies" on page 6-33.
5. Click **OK.**

Reconciling Operators based on Repository Objects

Table 6–10 describes the repository objects that you can use to update an operator by performing an inbound reconciliation.

Table 6–10 Operators Reconciled with Repository Objects

Operator Object	Repository Objects Available for Reconciliation
Mapping Table	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
Mapping External Table	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes

Table 6–10 (Cont.) Operators Reconciled with Repository Objects

Operator Object	Repository Objects Available for Reconciliation
Mapping View	Tables, External Tables, Views, Materialized Views, Files, Dimensions and Cubes
Mapping Materialized View	Tables, External Tables, Views, Materialized Views, Files, Dimensions and Cubes
Mapping Sequence	Sequences only
Mapping Flat File	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
Mapping Dimension	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
Mapping Cube	Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
Key Lookup Operator	Tables only
Pre Mapping Process Operator	Transformations only
Post Mapping Process Operator	Transformations only
Mapping Transformation Operator	Transformations only

Outbound Reconciliation

Outbound reconciliation updates a selected repository object to reflect changes in the operator. You can perform outbound reconciliation on tables, views, materialized views, transformations, and flat file operators. You can use outbound reconciliation for any of the following reasons:

- **Create new repository objects:** Use outbound reconciliation to create a new repository object in a target module. The new repository object must be of the same type. For example, you can use outbound reconciliation to create a new table based on an existing table, but you cannot create a new view based on an existing table.
- **Propagate changes:** Use outbound reconciliation to propagate changes you made in an operator to its associated repository object. When you rename the logical name for an operator or attribute, Warehouse Builder propagates the first 30 characters of the logical name as the bound name.
- **Replace repository objects:** Use outbound reconciliation to replace an existing repository object.

Outbound reconciliation has no impact on the dependent relationship between other operators in the mapping to repository objects. Table 6–11 lists the operators eligible for Outbound Reconcile.

Table 6–11 Outbound Reconcile Operators

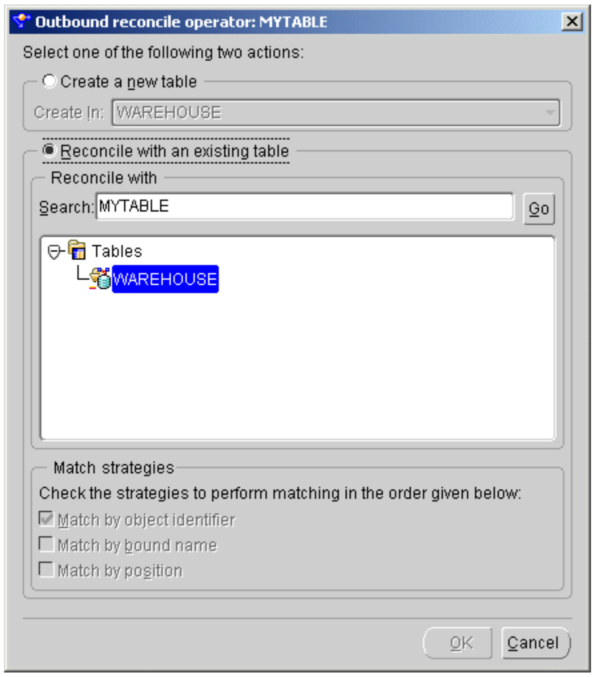
Mapping Objects	Create Repository Objects	Propagate Changes	Replace Repository Objects	Notes
External Tables	Yes	Yes	Yes	Updates the repository external table only and not the flat file associated with the external table. See "Reconciling an External Table Definition with a Record in a File" on page 3-22.
Flat Files	Yes	Yes	No	Creates a new, comma-delimited flat file for single record type flat files only. Cannot replace an existing file.
Mapping Input Parameters	Yes	Yes	Yes	Copies input attributes and data types as input parameters.
Mapping Output Parameters	Yes	Yes	Yes	Copies output attribute and data types as return specification for the function.
Materialized Views	Yes	Yes	Yes	Copies attributes and data types as columns.
Tables	Yes	Yes	Yes	Copies attributes and data types as columns. Constraint properties are not copied.
Transformations	Yes	Yes	Yes	
Views	Yes	Yes	Yes	Copies attributes and data types as columns.

To perform an outbound reconciliation on an existing operator:

1. Select an operator on the canvas.
2. From the **Edit** menu, select **Reconcile Outbound** or right-click the header of the operator and select **Reconcile Outbound.**

   The **Outbound Reconcile** dialog displays, as shown in Figure 6–23.

Figure 6–23 Outbound Reconcile Dialog (for Tables)


Outbound reconcile operator: MYTABLE
Select one of the following two actions:
Create a new table
Create In: WAREHOUSE
Reconcile with an existing table
Reconcile with
Search: MYTABLE
Go
Tables
WAREHOUSE
Match strategies
Check the strategies to perform matching in the order given below:
Match by object identifier
Match by bound name
Match by position
OK
Cancel

3. Select either **Create a new <object type>** or **Reconcile with an existing <object type>**.

   If you select **Create a new <object type>,** Warehouse Builder creates the object in the target module you specify. This option is useful for quickly creating staging tables for a data warehouse. For instructions how to use outbound reconciliation and the Auto-Mapping dialog to create staging tables, see "Example: Using the Mapping Editor to Create Staging Area Tables" on page 6-16.

   If you select **Reconcile with an existing <object type>,** you can make selections for either updating or replacing the object.

   To update the repository object, select the associated operator from the **Reconcile with** list. For example, to update the S_TABLE repository object, select the S_TABLE operator. The types of changes propagated from the operator to the repository object include changes to an operator business name and physical name and changes to an attribute data type.

   To replace the repository object, select the another operator from the **Reconcile with** list. For example, to replace the S_TABLE1 repository object, select S_TABLE2 operator. You can only replace a repository object with the same type of repository object. For instance, you can use outbound reconciliation to replace TABLE_A with TABLE_B but not VIEW_B.

4. Set the Match Strategies. For more information, see "Match Strategies" on page 6-33.

5. Click **OK.**

Match Strategies

You can specify the following strategies for reconciling an object in a mapping:

- Match by Object Identifier
- Match by Bound Name

- Match by Position

You can specify more than one reconciliation strategy. If you select more than one strategy, then Warehouse Builder performs the matches in the following order:

1. Match by object identifier
2. Match by position
3. Match by name

Match by Object Identifier
Use this strategy if you want operators to be consistent with changes to the repository object and if you want to maintain separate business names for your operator attributes despite changes to physical names in the repository object. Match by object identifier is not available if you want to reconcile with a different type of repository object.

This strategy uses the unique object identifiers to determine the correlation between the operator attributes and those of the selected repository object. Warehouse Builder removes attributes from the operator that do not correspond to an attribute in the repository object. This can happen if an attribute was added to the operator and not reconciled, or if a column was removed from the repository object before reconciliation. When an attribute is removed, any incoming or outgoing mapping lines connected to that attribute are also removed from the canvas. Attributes of the selected repository object that cannot be matched with those of the operator are added as new attributes at the end of the operator. Mapping lines for matched attributes are preserved.

Match by Bound Name
Use this strategy if you want bound names in the operator to be consistent with physical names for the object. If a repository object column was renamed, it is interpreted as if the column were deleted and a new column inserted. The mapping lines for renamed attributes are removed. You can also use this strategy with a different repository object if there are changes in the repository object that would change the structure of the operator.

This strategy uses matching between the bound names of the operator attributes and the physical names of the repository object attributes. Matching is case-sensitive. On inbound reconciliation, attributes of the operator that cannot be matched with those of the repository object are removed. When an attribute is removed, any incoming or outgoing mapping line connected to that attribute is also removed from the canvas. Attributes of the selected repository object that cannot be matched with those of the operator are added as new attributes to the operator. Mapping lines for matched attributes are preserved. Because bound names are read-only after you have bound an operator to a repository object, it is not possible to manipulate them to achieve a different match result during inbound reconciliation.

Match by Position
This strategy matches operator attributes with columns, fields, or parameters of the selected repository object by position. The first attribute of the operator is reconciled with the first attribute of the repository object, the second with the second, and so on. If the operator has more attributes than the repository object, then the excess attributes are removed from the operator. If you remove an attribute, any incoming or outgoing mapping line connected to that attribute is also removed from the canvas. If the selected repository object has more attributes than the operator, then they are added as new attributes to the end of the operator. Mapping lines for existing attributes in the

operator are preserved. Use this strategy to reconcile with a different repository object if you want to preserve the business names of your operator attributes. This strategy is most effective when the only changes to the repository object are the addition of extra columns, fields, or parameters at the end of the object.

7 Debugging Mappings

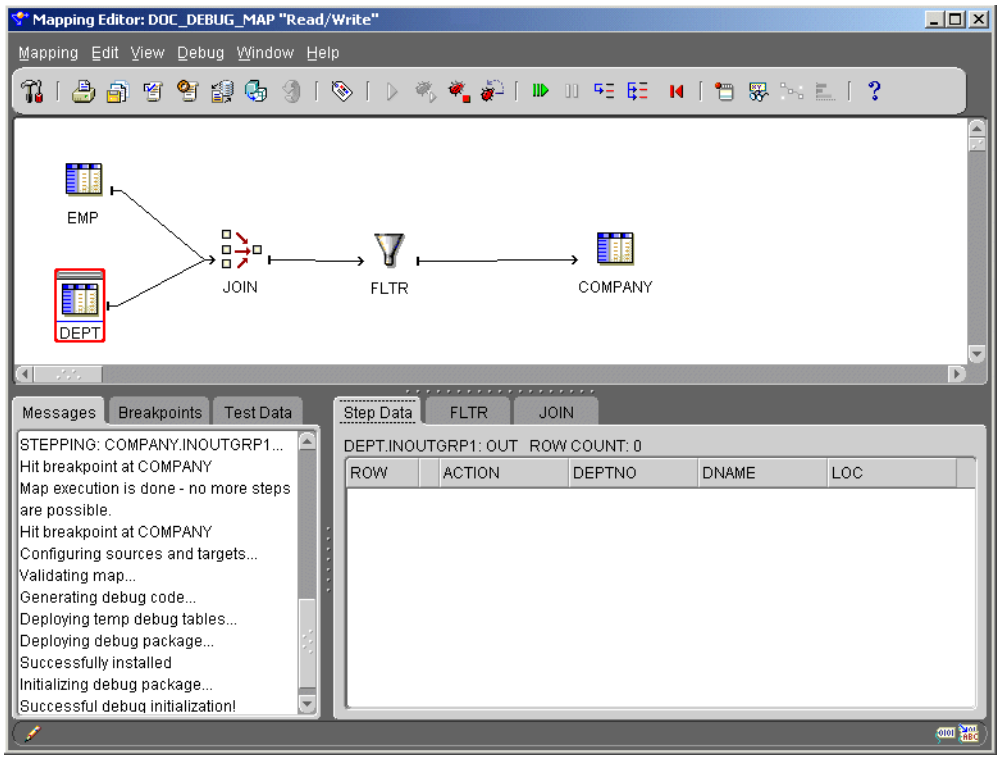
Warehouse Builder provides debugging capabilities for data flows within the Mapping Editor. These debugging features, available from the toolbar and debug menu in the Mapping Editor, are only enabled after you begin a debug session and connect to a valid target schema. You can then run a mapping in debug mode using a defined set of test data to follow the flow of data as it is extracted, transformed, and loaded and ensure that the designed data flow is behaving 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.

The following topics explain the debug process and how it can be used in the Mapping Editor:

About Debug Mode in the Mapping Editor

After you start a debug session, the Mapping Editor displays additional panels intended to provide detailed information about the debug session. Additional debug buttons in the toolbar and the Debug menu are also enabled after the debug session begins.The status bar along the bottom of the Mapping Editor also contains information about the status of the debug session.

Figure 7–1 Mapping Editor in Debug Mode


Mapping Editor: DOC_DEBUG_MAP "Read/Write"
Mapping Edit View Debug Window Help
EMP
DEPT
JOIN
FLTR
COMPANY
Messages
Breakpoints
Test Data
Step Data
FLTR
JOIN

```
STEPPING: COMPANY.INOUTGRP1...
Hit breakpoint at COMPANY
Map execution is done - no more steps
are possible.
Hit breakpoint at COMPANY
Configuring sources and targets...
Validating map...
Generating debug code...
Deploying temp debug tables...
Deploying debug package...
Successfully installed
Initializing debug package...
Successful debug initialization!
```

DEPT.INOUTGRP1: OUT ROW COUNT: 0

ROW	ACTION	DEPTNO	DNAME	LOC

The left bottom panel 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.
- **Test Data:** Displays a list of all data objects used in the mapping. The list also indicates which data objects have test data defined.

The right bottom panel includes Step Data and watch point tabs that contain input and output information for the operators being debugged. The Step Data tab contains information about the current step in the debug session. Additional tabs can be added for each watch you set. These watch tabs allow you to keep track and view data that has passed or will pass through an operator regardless of the currently active operator in the debug session. Operators that have more than one input group or more than one output group display an additional drop down list that enables you to select a specific group.

The debug buttons in the toolbar, as shown in Figure 7–2, and Debug menu options become enabled only after the debug session begins. Table 7–1 describes each of these buttons.

Figure 7–2 The Mapping Editor Debug Tools

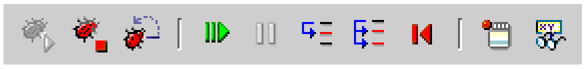


Table 7–1 Mapping Editor Debug Tools

Icon	Action	Description
	Debug Start	Click this button to start the debug session.
	Debug End	Click this button to end the debug session.
	Debug Re-initialize	Click this button to re-initialize the debug session. Re-initialize re-deploys the debug code and resets the debug session.
	Resume	Click this button to resume the debug session. The debug session will run until the next breakpoint. If there are no breakpoints it will finish debugging the mapping.
	Pause	Click this button to pause the mapping you are debugging. This functionality is not enabled in the current release.
	Step	Click this button to step through an operator row by row.
	Skip	Click this button to skip through an operator. All rows for that operator will be processed.
	Reset	Click this button to reset the debug session and start debugging from the beginning.
	Set Breakpoint	Click this button to set a breakpoint on the selected operator. If a breakpoint already exists on the operator, clicking this button removes the breakpoint.
	Set Watch Point	Click this button to set a watch on the selected operator. If a watch point already exists on the operator, clicking this button removes the watch.

Starting a Debug Session

To start a debug session:

1. From the Mapping Editor, select **Debug** and then **Start.** You can also click the Start Debug button from the toolbar.

   The Mapping Editor switches to debug mode with the debug panels appearing in the bottom of the editor and a Connection Information dialog displays.

2. Specify connection information for the target schema you are using for debug runs.

   The generated code is deployed to this schema. You can connect to another target schema while you are in a debug session by selecting **Re-connect** from the **Debug** menu.

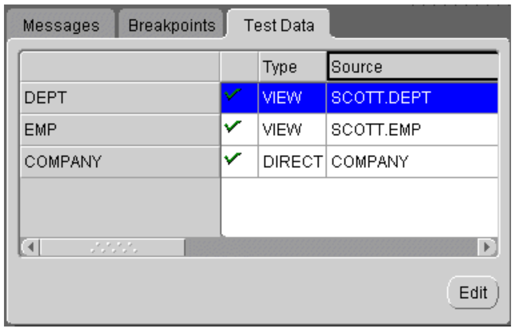
After the connection has been established, a message displays to indicate that you may want to define test data. If you have previously defined test data, then you will be asked if you want to continue with initialization. In order to debug a mapping, each source or target operator must be bound to a database object and test data must be defined for the database object. By default, the operators in the mapping are connected to local sources and targets based on matching object names.

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 columns in the data source you select matches the columns in the mapping operators. There are two possible types: Direct Access and Deployed as View. If there is an exact match then the type is listed as Direct Access. If 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.

Use the Edit button 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.

Figure 7–3 Test Data Tab



		Type	Source
DEPT	✓	VIEW	SCOTT.DEPT
EMP	✓	VIEW	SCOTT.EMP
COMPANY	✓	DIRECT	COMPANY

To define or edit test data:

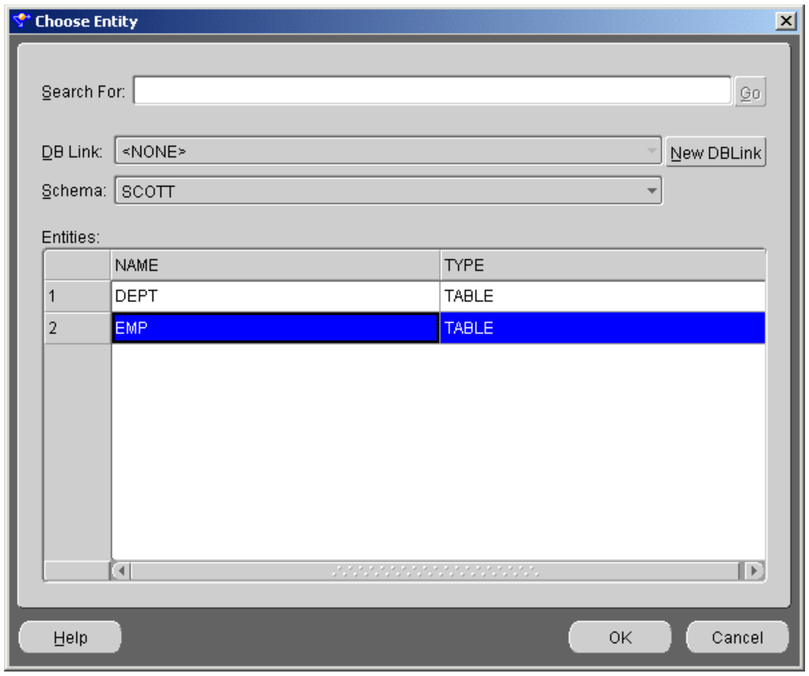
1. From the Test Data tab in the Mapping Editor, select an operator from the list and click the **Edit** button.

   The Define Test Data dialog displays, as shown in Figure 7–5.

2. Specify a Access Name for the object.

   To bind an existing database object, type the name into the Access Name field or click the **Browse** button to search for an existing database that you want to use. The Choose Entity dialog displays as shown in Figure 7–4. Use this dialog to create or select a database link and schema. Then select a database object from the list of entities and click **OK.**

Figure 7–4 Choose Entity Dialog



	NAME	TYPE
1	DEPT	TABLE
2	EMP	TABLE

To create and bind a new table and copy the existing set of test data shown in the test data spreadsheet, click the **Create New Table** button on the Define Test Data dialog. For more information, see "Creating New Tables" on page 7-6.

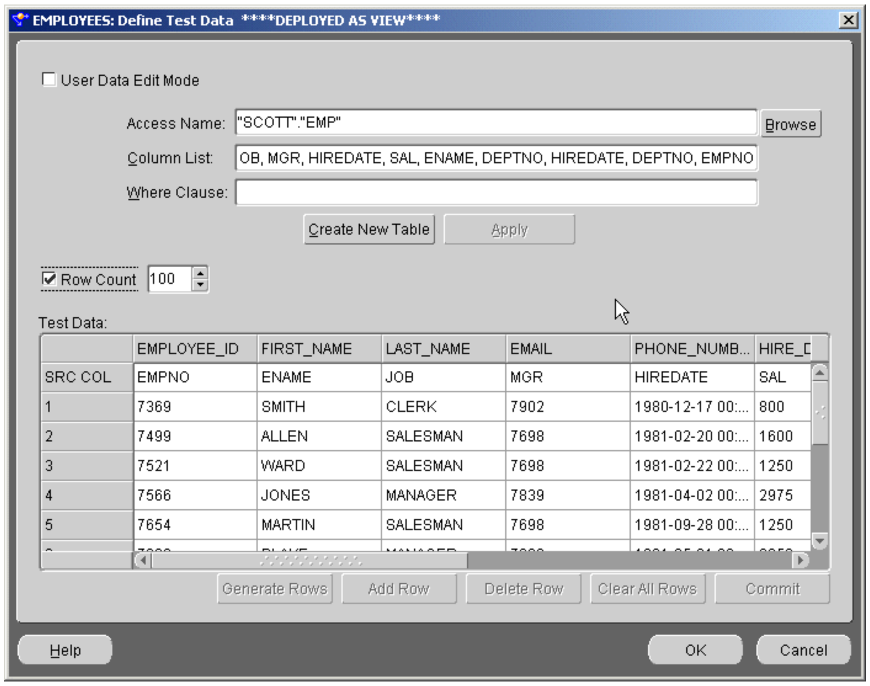
3. To edit data, check the User Data Edit Mode check box.

   This enables you to create specific test data necessary to debug particular scenarios for which no suitable data can be found in the bound database object. You can click each of the cells in the test data panel and edit the values.

4. To restrict the number of rows selected to be used as test data, check the Row Count check box and set number of rows. By default, the row count is set to 100 rows.

   This enables you to limit the amount of data in the debug session.

Figure 7–5 Test Data Definition Dialog



User Data Edit Mode [ ]

Access Name: "SCOTT"."EMP"

Column List: OB, MGR, HIREDATE, SAL, ENAME, DEPTNO, HIREDATE, DEPTNO, EMPNO

Where Clause:

Row Count [✓] 100

Test Data:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMB...	HIRE_D
SRC COL	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
1	7369	SMITH	CLERK	7902	1980-12-17 00:...	800
2	7499	ALLEN	SALESMAN	7698	1981-02-20 00:...	1600
3	7521	WARD	SALESMAN	7698	1981-02-22 00:...	1250
4	7566	JONES	MANAGER	7839	1981-04-02 00:...	2975
5	7654	MARTIN	SALESMAN	7698	1981-09-28 00:...	1250

Creating New Tables

If you create a new table using the Create New Table button on the Define Test Data dialog, the name of the table will be the name of the data operator prefixed by DBG_. If this name exists in the target, it will be suffixed by a sequence number to guarantee a unique object name. The table is created in the target schema you specified when you started the debug run. The debugger will not automatically drop that table so that you can always reuse it for other sessions. Constraints are not carried over for the new table.

You can edit test data at anytime using the Define Test Data dialog. If you change the binding of the operator to another database object, you must re-initialize the debug session to implement the change before running the mapping again in debug mode.

Note: The data loaded in the target definitions will be implicitly committed. If you do not want the target objects updated, you should create copies of target objects using the Create New Table button.

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.

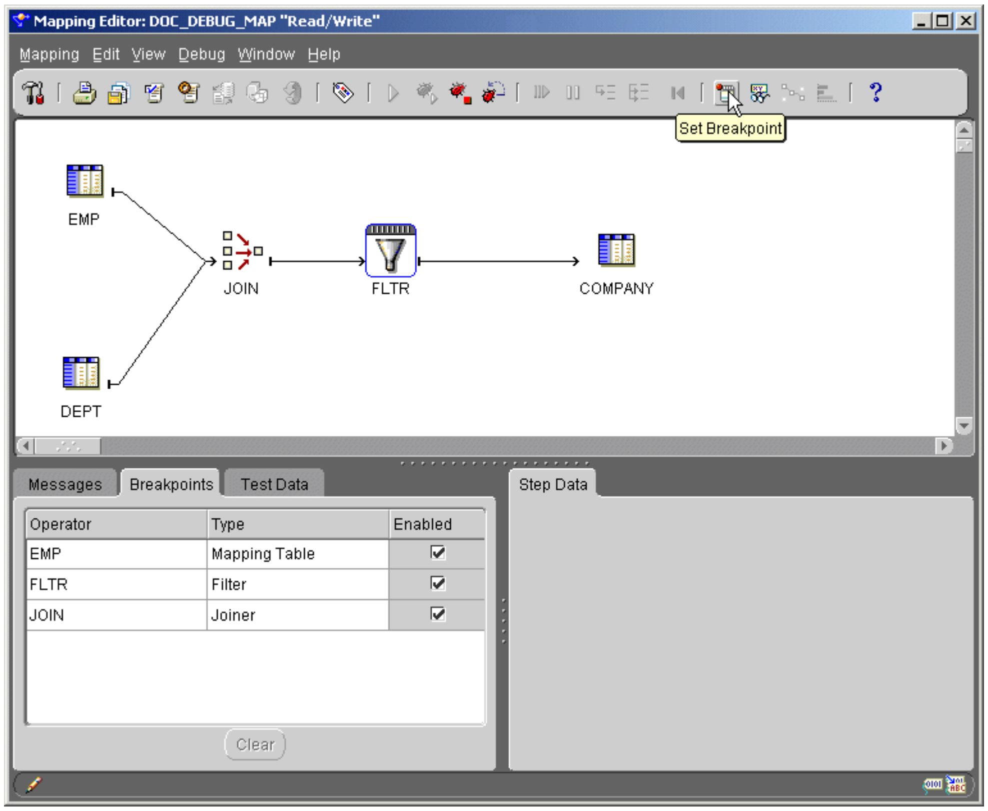
To set or remove a breakpoint:

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 in order to disable or enable them.

Figure 7–6 Breakpoints in Debug Mode



Operator	Type	Enabled
EMP	Mapping Table	✓
FLTR	Filter	✓
JOIN	Joiner	✓

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.

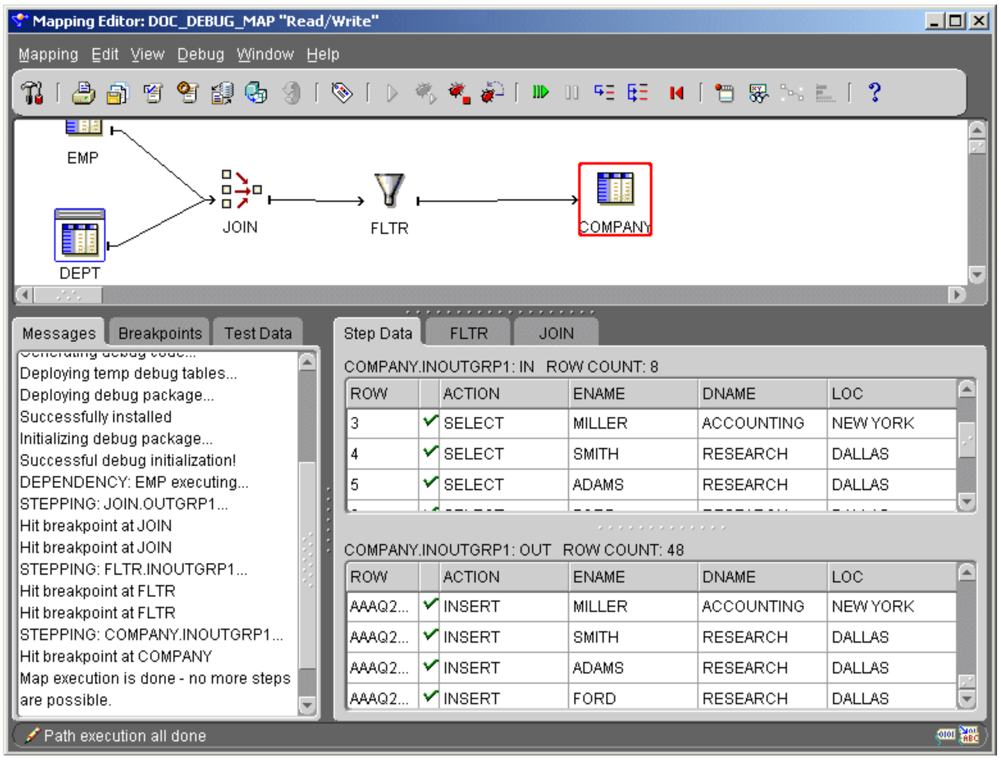
To set a watch:

1. 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 as shown in Figure 7–7.

2. 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.

Figure 7–7 Watches as Separate Tabs on Right-bottom Panel



```
Deploying temp debug tables...
Deploying debug package...
Successfully installed
Initializing debug package...
Successful debug initialization!
DEPENDENCY: EMP executing...
STEPPING: JOIN.OUTGRP1...
Hit breakpoint at JOIN
Hit breakpoint at JOIN
STEPPING: FLTR.INOUTGRP1...
Hit breakpoint at FLTR
Hit breakpoint at FLTR
STEPPING: COMPANY.INOUTGRP1...
Hit breakpoint at COMPANY
Map execution is done - no more steps
are possible.
```

COMPANY.INOUTGRP1: IN ROW COUNT: 8

ROW	ACTION	ENAME	DNAME	LOC
3	✓ SELECT	MILLER	ACCOUNTING	NEW YORK
4	✓ SELECT	SMITH	RESEARCH	DALLAS
5	✓ SELECT	ADAMS	RESEARCH	DALLAS

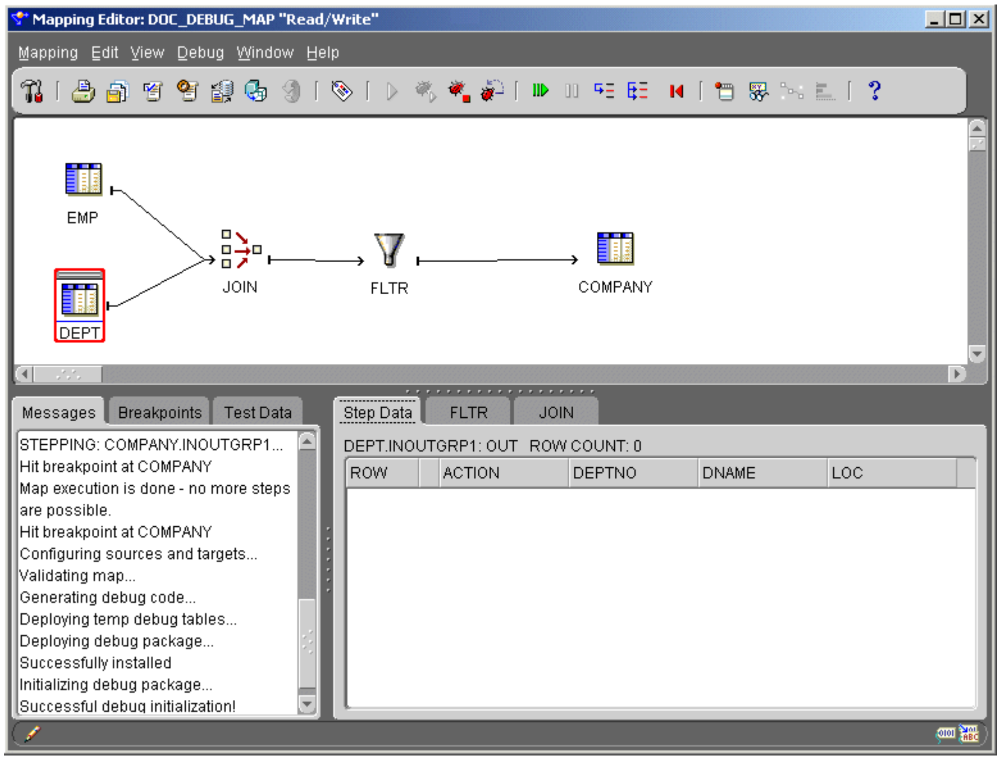
COMPANY.INOUTGRP1: OUT ROW COUNT: 48

ROW	ACTION	ENAME	DNAME	LOC
AAAQ2...	✓ INSERT	MILLER	ACCOUNTING	NEW YORK
AAAQ2...	✓ INSERT	SMITH	RESEARCH	DALLAS
AAAQ2...	✓ INSERT	ADAMS	RESEARCH	DALLAS
AAAQ2...	✓ INSERT	FORD	RESEARCH	DALLAS

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 the Re-initialize button on the toolbar. Warehouse Builder generates the debug code and deploys the package to the target schema you specified. Figure 7–8 shows an initialized mapping that is ready for debugging.

Figure 7–8 Initialized Mapping Ready for Debugging


Mapping Editor: DOC_DEBUG_MAP "Read/Write"
Mapping Edit View Debug Window Help
EMP
DEPT
JOIN
FLTR
COMPANY
Messages
Breakpoints
Test Data
Step Data
FLTR
JOIN

```
STEPPING: COMPANY.INOUTGRP1...
Hit breakpoint at COMPANY
Map execution is done - no more steps
are possible.
Hit breakpoint at COMPANY
Configuring sources and targets...
Validating map...
Generating debug code...
Deploying temp debug tables...
Deploying debug package...
Successfully installed
Initializing debug package...
Successful debug initialization!
```

DEPT.INOUTGRP1: OUT ROW COUNT: 0

ROW	ACTION	DEPTNO	DNAME	LOC

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 a Leading Source and Path

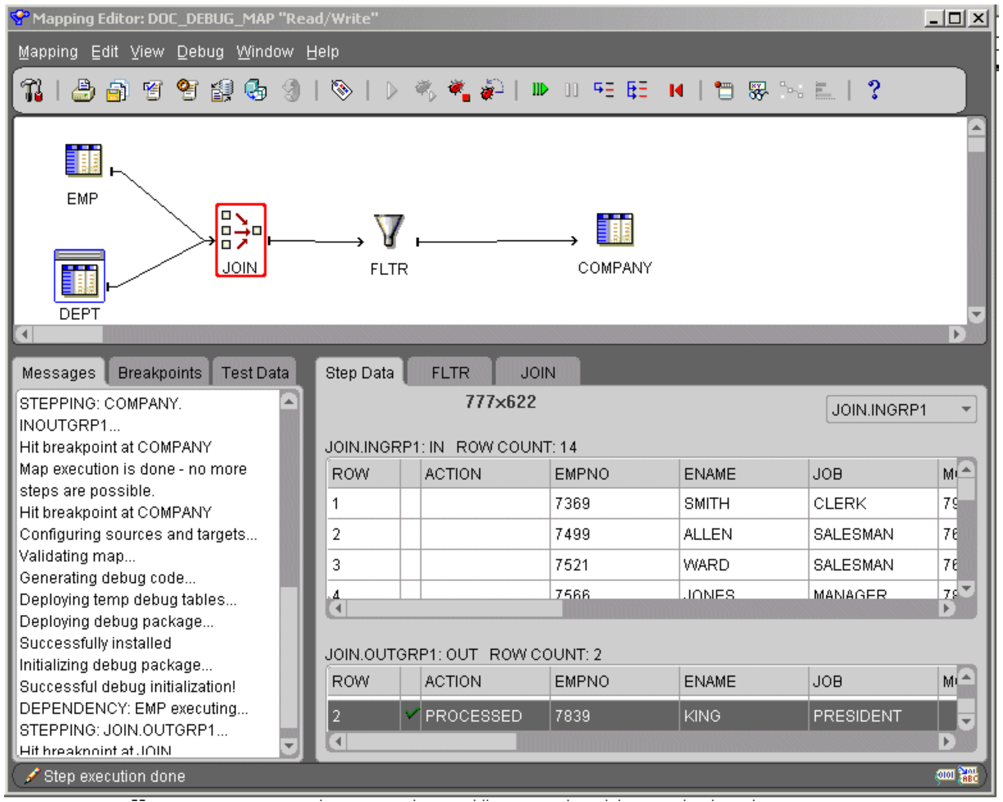
A mapping may have more than one source and more than one path to debug. If a mapping has more than one source then Warehouse Builder will ask you what leading source to begin with. For example, when two tables are mapped to a joiner, you will have to select the leading source table.

There may be multiple paths that the debugger can walk through after it has finished one path. This is for example the case when you use a splitter. Having finished one path the debugger will ask you whether you would like to complete the other paths as well.

The mapping finishes if all target operators have been processed or if the maximum number of errors as configured for the mapping has been reached. The debug connection and test data definitions are stored when you commit changes to the Warehouse Builder repository. Breakpoint and watch settings are not stored and must be re-set each time you open a mapping.

As the debugger runs it generates debug messages whenever applicable. You can follow the data flow through the operators. The active operator is indicated by a red box surrounding the operator as shown in Figure 7–9.

Figure 7–9 The Mapping Debugger


Mapping Editor: DOC_DEBUG_MAP "Read/Write"
Mapping Edit View Debug Window Help
EMP
DEPT
JOIN
FLTR
COMPANY
Messages Breakpoints Test Data
Step Data FLTR JOIN
777x622
JOIN.INGRP1
Step execution done

```
STEPPING: COMPANY.
INOUTGRP1...
Hit breakpoint at COMPANY
Map execution is done - no more
steps are possible.
Hit breakpoint at COMPANY
Configuring sources and targets...
Validating map...
Generating debug code...
Deploying temp debug tables...
Deploying debug package...
Successfully installed
Initializing debug package...
Successful debug initialization!
DEPENDENCY: EMP executing...
STEPPING: JOIN.OUTGRP1...
Hit breakpoint at JOIN
```

JOIN.INGRP1: IN ROW COUNT: 14

ROW	ACTION	EMPNO	ENAME	JOB	M
1		7369	SMITH	CLERK	79
2		7499	ALLEN	SALESMAN	76
3		7521	WARD	SALESMAN	76
4		7566	JONES	MANAGER	78

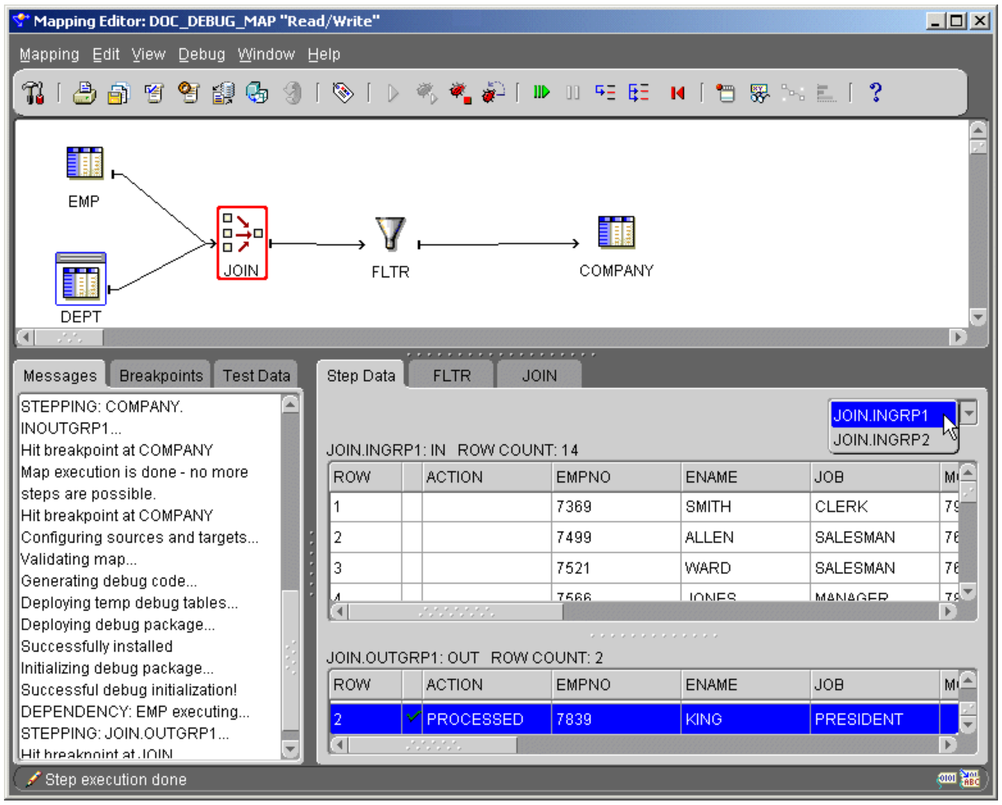
JOIN.OUTGRP1: OUT ROW COUNT: 2

ROW	ACTION	EMPNO	ENAME	JOB	M
2	✓ PROCESSED	7839	KING	PRESIDENT	

Step Data Panel

If an operator has more than one input or output group then the debugger will have a drop down list in the upper right corner, above the input or output groups. Use this drop down list to select the group you are interested in. This applies both to the step data and to a watch. Figure 7–10 shows an example for the join operator.

Figure 7–10 Multiple Input Groups for a Join Operator



```
STEPPING: COMPANY.
INOUTGRP1...
Hit breakpoint at COMPANY
Map execution is done - no more
steps are possible.
Hit breakpoint at COMPANY
Configuring sources and targets...
Validating map...
Generating debug code...
Deploying temp debug tables...
Deploying debug package...
Successfully installed
Initializing debug package...
Successful debug initialization!
DEPENDENCY: EMP executing...
STEPPING: JOIN.OUTGRP1...
Hit breakpoint at JOIN
```

JOIN.INGRP1: IN ROW COUNT: 14

ROW		ACTION	EMPNO	ENAME	JOB	M
1			7369	SMITH	CLERK	79
2			7499	ALLEN	SALESMAN	76
3			7521	WARD	SALESMAN	76
4			7566	JONES	MANAGER	78

JOIN.OUTGRP1: OUT ROW COUNT: 2

ROW		ACTION	EMPNO	ENAME	JOB	M
2	✓	PROCESSED	7839	KING	PRESIDENT	

Debugging Mappings with Correlated Commit

If you begin a debug session for a mapping that has the Correlated Commit parameter set to ON, the mapping is not debugged using paths. If Correlated Commit is 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. If Correlated Commit is set to ON, 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.

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, 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, 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.

Re-Initializing a Debug Session

If 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 in order to continue debugging the mapping with the new changes. The re-initialize button on the toolbar or the re-initialize menu item in the debug menu can be used to regenerate and re-deploy the debug code. The mapping debug session will start from the beginning after re-initialization.

Scalability

Scalability when debugging a mapping applies both to the amount of data that is passed as well as to the number of columns displayed in the step data panel. The define test data dialog provides a row limit that you can use to limit the amount of data that flows through the mapping. Also, you can define your own data set by creating your own table and manipulating the records manually.

To restrict the number of columns displayed on the step data window or on a watch tab you can use display sets. By default every operator has a display set ALL and a display set MAPPED to display only the mapped attributes. You can manually add display sets on sources by using the mapping editor directly. Select the use display set option under the right mouse button on an input or output group to select the display set.

Restrictions

These issues include functions currently included in the mapping debugger that have not yet been enabled as well as functions that intend to be added in future releases.

1. Mappings run using the debug mode in the Mapping Editor are intended to be used for debug purposes only. Mappings run from the Mapping Editor are not as performant as mappings that are run using the Deployment Manager. This is attributed to the set up of temporary objects necessary to support the debugging capabilities. Use the Deployment Manager to run mappings.
2. You cannot pause an active debug run using the pause button on the toolbar or the associated item in the debug menu.
3. Mapping statistics will be provided in a future release. The statistics will appear as an additional tab on the left bottom panel.
4. You cannot use the Runtime Audit Browser to view the results of a mapping run in debug mode.
5. Breakpoint and watch settings are not preserved between debug sessions.
6. Only mappings that would be implemented as a PL/SQL package can currently be run in debug mode.
7. The following mapping operators are not supported when running mappings in debug mode:
   - Advanced Queue
   - Flat File
   - Source Table using from an ABAP application

8 Using Mapping Operators

This chapter provides details on how to use operators in a mapping to transform data and how to use the Expression Builder to create expressions. Operators are listed alphabetically.

This chapter includes the following topics:

Using the Expression Builder

Some of the data flow operators described in this chapter 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

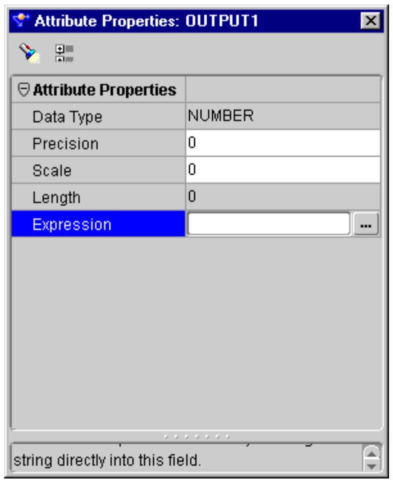
You can open the Expression Builder from the operator property windows in the operators such as filters, joiners, splitters, and aggregators.

You can open the Expression Builder from the attribute property windows in the operators such as expressions, data generators, and constants.

To open the Expression Builder:

1. From the operator properties window or the attribute property window, click the **...** button in the expression field. Figure 8–1 shows the attribute properties window.

Figure 8–1 Attribute Property Window

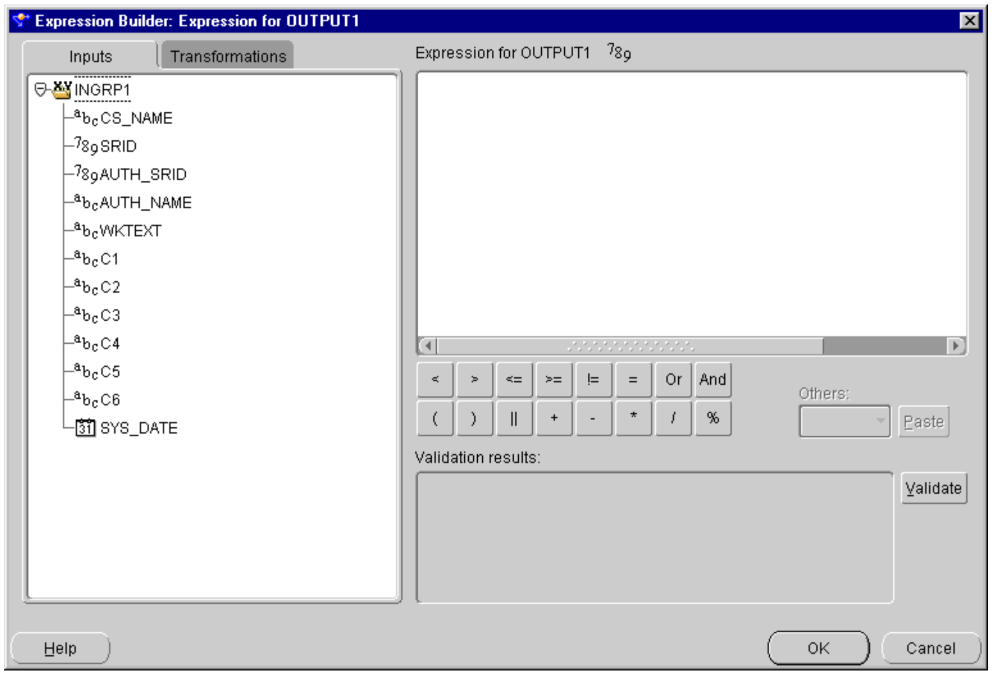


Attribute Properties	
Data Type	NUMBER
Precision	0
Scale	0
Length	0
Expression	

string directly into this field.

The Expression Builder displays as shown in Figure 8–2.

Figure 8–2 Expression Builder Interface



2. Create an expression by:

   - Typing text into the expression field.

   - Dragging items from the Inputs and Transformations tabs on the left panel and dropping them into the **Expression** field on the right.
   - Double clicking on items from the Inputs and Transformations tabs on the left panel.
   - Clicking arithmetic operator buttons available under the **Expression** field.

3. Click **Validate.**

   This verifies the accuracy of the expression syntax.

4. Click **OK** to save the expression and close the Expression Builder.

The Expression Builder User Interface

The Expression Builder contains the following parts:

- In the left panel, the navigation tree displays two tabs:
  - **Inputs Tab:** A list of input parameters.
  - **Transformations Tab:** A list of predefined functions and procedures located in the Oracle Transformation Library, the Global Shared Library, and a custom Transformation Library. See the Oracle Warehouse Builder Transformation Guide for more information.
- **Expression Field:** At the top of the right panel is the **Expression** field. Use this field to type and edit expressions.
- **Arithmetic Operator Buttons:** Below the **Expression** field are buttons for arithmetic operators. Use these buttons to build an expression without typing. The arithmetic operators available vary by the type of data flow operator that is active.
- **Others:** A drop-down list of available SQL clauses that are appropriate for the active expression type.

  Beginning in Oracle9*i*, the `CASE` function is recommended over the `DECODE` function because the `CASE` function generates both SQL and PL/SQL while `DECODE` is limited to SQL. If you use the `DECODE` function in an expression, Warehouse Builder promotes it to `CASE` where appropriate during code generation. This enables you to deploy the `DECODE` functionality in all operating modes (such as setbased or rowbased) and transparently across Oracle database releases (8.1, 9.0 and higher).

  For example, Warehouse Builder converts the function

  ```
  DECODE (T1.A, 1, ‘ABC’, 2, ‘DEF’, 3, ‘GHI’, ‘JKL’)
  ```

  to the following:

  ```
  CASE T1.A WHEN 1 THEN ‘ABC’
  WHEN 2 THEN ‘DEF’
  WHEN 3 THEN ‘GHI’
  ELSE ‘JKL’
  ```

- **Validation Results Field:** At the bottom of the right panel is the **Validation Results** field. After you select the **Validate** button to the right if this field, this field displays the validation results.
- **Validate Button:** Use this button to validate the current expression in the Expression Builder. Validation ensures that all mapping objects referred to by the expression have associated repository objects. The expressions you create with the Expression Builder are limited to the operator inputs and to any transformations

available in a project. This limitation protects the expression from becoming invalid because of changes external to the operator. If the deployment database is different from the design repository, it may not accept the expression. If this happens, the expression may be valid but incorrect against the database. In this case, expression errors can only be found at deployment time.

Aggregator Operator

The Aggregator operator calculates data aggregations, such as summations and averages and provides an output row set with the aggregated data.

Because each Aggregator operator shares a GROUP BY and HAVING clause, 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.

The Aggregator operator has one input group and one output group. Connecting the source to the input group produces the corresponding aggregated row set in the output group. The resulting output includes the column used by the GROUP BY or the HAVING clause.

The Aggregator operator contains the following properties:

- **Group By Clause:** Defines how the incoming row set is grouped to return a single summary row for each group. An ordered list of attributes in the input group specifies how this grouping is performed. Adding an attribute in the GROUP BY clause defaults the aggregation expression to NONE.
- **Having Clause:** A boolean condition restricting 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.
- **Expression:** 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, ensuring a correct result in the preceding example.

  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 use an Aggregator operator in a mapping:

1. 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 and select **Edit.**

   Warehouse Builder displays the Operator Editor.
4. On the Output Attributes tab, select **Add** to add output attributes.
5. Click **OK** to close the Operator Editor.

6. From the attribute properties window, define expressions for each output attribute. For detailed instructions, see "Defining the Output Attributes from Aggregator Operators" on page 8-5.
7. Define the Group By Clause for the operator. For detailed instructions, see "Defining the Output Attributes from Aggregator Operators" on page 8-5.

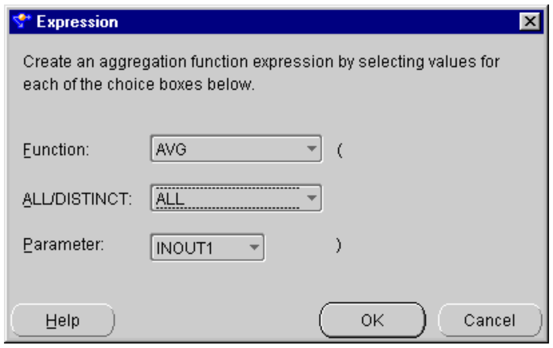
Defining the Output Attributes from Aggregator Operators

To define expressions for output attributes:

1. In the Aggregator operator on the mapping canvas, right click an output attribute and select **Attribute Properties.**
2. Click the **...** button to the right of the Expression property.

   The Expression dialog displays as shown in Figure 8–3.

Figure 8–3 Expression Dialog



Expression

Create an aggregation function expression by selecting values for each of the choice boxes below.

Function: AVG (

ALL/DISTINCT: ALL

Parameter: INOUT1)

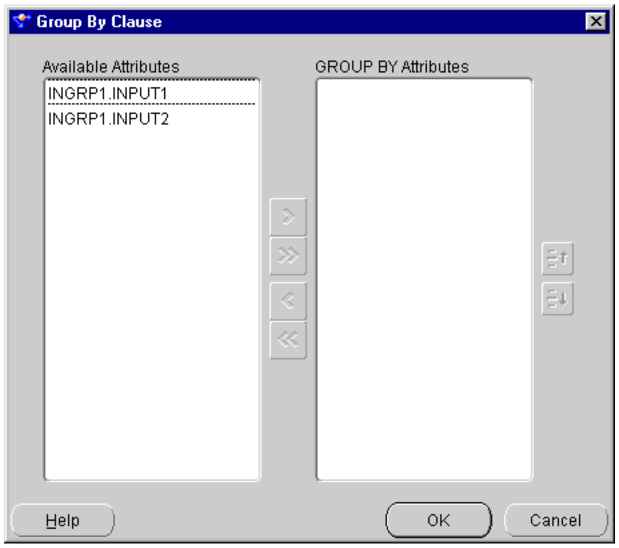
3. Select a Function, ALL or DISTINCT, and a parameter from the drop-down lists.
4. Click **OK.**

To define the Group By Clause:

1. In the Aggregator operator on the mapping canvas, right click an output attribute and select **Attribute Properties.**
2. Click the **...** button to the right of the Group By Clause property.

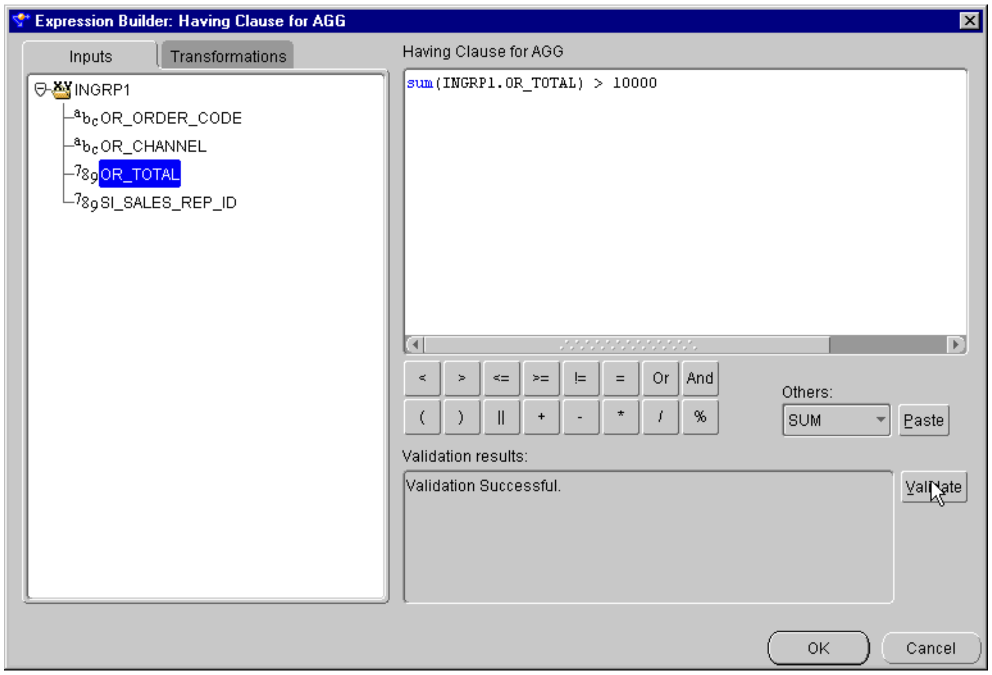
   The Group By Clause dialog displays as shown in Figure 8–4.

Figure 8–4 Group By Clause Dialog



3. Move the attributes from the Available Attributes list to the GROUP BY Attributes list.
4. Click **OK.**
5. Click the **...** button to define a HAVING clause using the Expression Builder.
6. Create an expression such as `sum(INGRP1.OR_TOTAL) > 10000` as shown in Figure 8–5.

Figure 8–5 Expression Builder Showing a Sum Statement



```
sum(INGRP1.OR_TOTAL) > 10000
```

Validation Successful.

7. Map the attributes you edited from the output group of the aggregator operator to the attributes in the target.

Constant Operator

The Constant operator enables you to define constants. Constants are initialized at the beginning of the execution of the mapping. Constant values can be used anywhere in a mapping.

The Constant operator produces a single output group that can contain one or more constant attributes. For any defined constant data type, the output expression must be a valid SQL expression returning a value of the same data type. For VARCHAR, CHAR, or VARCHAR2 data types, you must enclose constant string literals within single quotes. For example, `‘my_string’`.

The Constant operator contains the following property:

- **Expression:** Defines the constant value represented by the attribute.

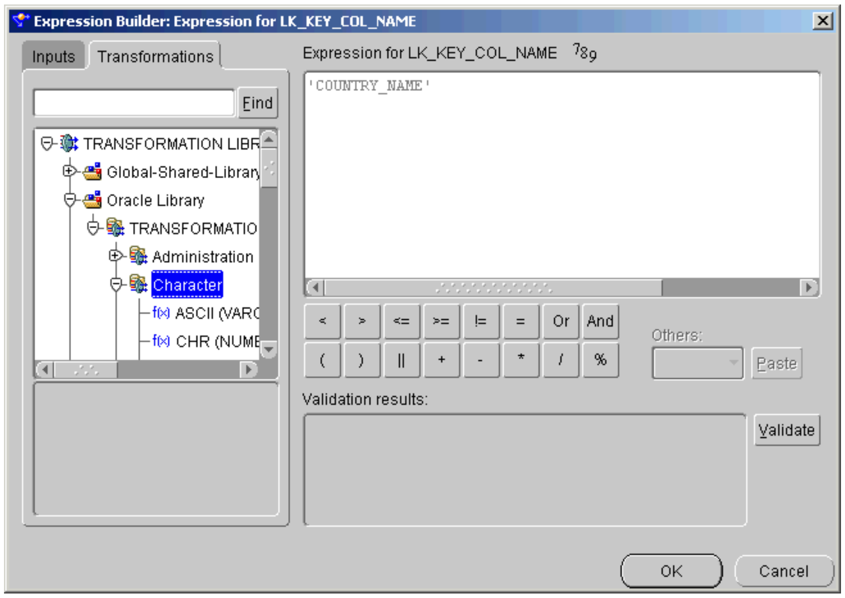
To use a constant operator in a mapping:

1. Drop a **Constant** operator onto the Mapping Editor canvas.
2. Right-click the operator and select **Edit.**

   Warehouse Builder displays the Operator Editor.
3. On the Output Attributes tab, select add to create an output attribute.
4. Click **OK** to close the Operator Editor.
5. On the operator in the mapping canvas, right-click an attribute and select **Attribute Properties.**

   The Attributes properties window displays.
6. Enter an expression in the Expression field or click **...** to define an expression using the Expression Builder shown in Figure 8–6. The length, precision, and scale properties assigned to the attributes of the Constant operator must match the actual values returned by the expressions defined in the mapping.

Figure 8–6 Expression Builder Showing A Constant


Expression Builder: Expression for LK_KEY_COL_NAME

Expression for LK_KEY_COL_NAME

'COUNTRY_NAME'

7. Select **OK.**

8. Connect the output attribute to the appropriate target attribute.

Data Generator Operator

Use a Data Generator operator to provide information such as record number, system date, and sequence values.

Recommendation: For PL/SQL mappings use a Constant Operator or Mapping Sequence Operator instead of a Data Generator.

For mappings with flat file sources, the Data Generator operator also provides a place to enter constant information. 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

It is possible for Warehouse Builder to 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.

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, then the date is loaded in the format dd-mon-yy. After the load, you only access it in that format. If the system date is loaded into a DATE column, then you can only access it in a variety of formats including the time and the date. 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 8–1 lists the options available for sequence values.

Table 8–1 Sequence Value Options

Value	Description
column_name	The name of the column in the database that the sequence is assigned to.
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 a record is rejected because of a format error or an Oracle error, the generated sequence numbers are not reshuffled to mask this. 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 three rows inserted are numbered 10, 14, and 16, not 10, 12, 14. The sequence of inserts is preserved despite data errors. When you correct the rejected data and reinsert it, you can manually set the columns to match the sequence.

Although the Data Generator operator has only one output group, it has predefined attributes corresponding to Record Number, System Date, and a typical Sequence. While modification of these attributes is not recommended, you can create new attributes. The Data Generator operator is only valid for a SQL*Loader mapping.

Note: There can only be one Data Generator operator in a mapping.

The Data Generator attribute contains the following property:

- **Expression:** Expression to use when you map this attribute. Make sure the value entered for the expression is valid SQL*Loader syntax.

To use a data generator in a mapping:

1. Drop a **Data Generator** operator onto the Mapping Editor canvas.
2. Right-click the operator and select **Edit.**

   Warehouse Builder displays the Operator Editor.
3. Select the Output Attributes tab.

   Warehouse Builder displays the pre-defined output attributes RECNUM, SYS_DATE, 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 Operator Editor.
6. On the operator in the mapping canvas, right-click the RECUM attribute and select **Attribute Properties.**

   Warehouse Builder displays the Attribute Properties window.
7. In the expression field, click the **...** to open the Expression Builder and define an expression.

8. Repeat steps 6 and 7 for the SEQUENCE attribute.

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.

To remove duplicates:

1. Drop the Deduplicator operator onto the Mapping Editor canvas.
2. Connect the attributes from the source operator to the group input/output of the Deduplicator operator.
3. Connect the attributes from the Deduplicator operator group to the attributes of the target operator.

Expression Operator

The 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. 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.

The Expression operator contains only one input group and one output group. These are created automatically when you drop the operator onto the Mapping Editor canvas.

The output expressions for this operator cannot contain any aggregation functions. To use aggregation functions, use the Aggregator operator. See "Aggregator Operator" on page 8-4.

Every output attribute in an Expression operator contains the following properties:

- **Data Type:** The data type of the attribute.
- **Precision:** The precision of the attribute, used for numeric type attributes only.
- **Scale:** The scale of the attribute, used for numeric type attributes only.
- **Length:** The length of the attributes, used for character type attributes only.
- **Expression:** The expression template for the output attribute. For code generation, the input attributes are replaced by the input attribute names in the expression template.

To use an expression operator in a mapping:

1. Drop an **Expression** operator onto the Mapping Editor canvas.
2. Connect the appropriate source attributes to the Expression input group.

   Warehouse Builder copies the input attributes into the operator.
3. Right-click the operator in the mapping canvas and select **Edit.**

   Warehouse Builder displays the Operator Editor.
4. On the Output Attributes tab, select **Add** and specify the attribute name, data type, and other properties.

5. Click **OK** to close the Operator Editor.
6. From the Expression operator, right-click the output attribute and select the **Attribute Properties.**
7. Click the field to the right of the **Expression** property and enter a filter condition expression or click **...** to open the Expression Builder and define an expression.
8. Close the Attribute Properties window.
9. Connect the Expression output attribute to the appropriate target attribute.

Filter Operator

You can conditionally filter out rows using the Filter 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 connect a source operator to the Filter operator, apply a filter condition, and send a subset of rows to the next operator.

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.

The Filter operator contains the following property:

- **Filter Condition:** The boolean condition that determines which rows are sent to the output row set.

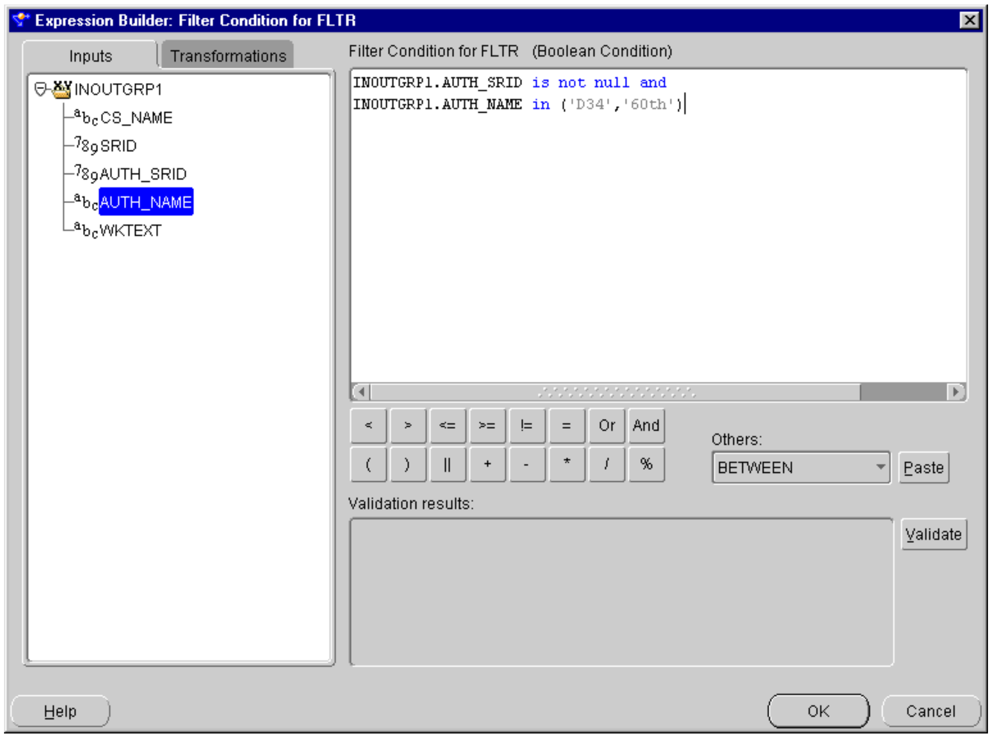
When you generate a mapping that includes a Filter operator, the Code Viewer displays filter condition expressions as a WHERE clause for set-based view mode. The filter input names in the original filter condition are replaced by actual column names from the source table, qualified by the source table alias.

To use 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 attribute for the Filter operator.
3. Right-click the Filter operator header and select **Operator Properties.**

   The Filter Properties window displays.
4. Click the field to the right of the Filter Condition property and enter a filter condition expression. Or click **...** to define a filter condition using the Expression Builder as shown in Figure 8–7.

Figure 8–7 Expression Builder Showing a Filter Condition



```
INOUTGRP1.AUTH_SRID is not null and
INOUTGRP1.AUTH_NAME in ('D34','60th')
```

5. Click **OK** in the Expression Builder and close the Filter Properties window.
6. Connect the Filter operator outputs to the input/output group in the target.

Joiner Operator

You can use the Joiner operator to join multiple row sets from different sources with different cardinalities, and produce a single output row set.

The Joiner operator uses a boolean condition that relates column values in each source row set to at least one other row set.

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 the default foreign keys result in duplicate WHERE clauses, the Joiner operator will remove the duplicate clauses. This can happen if the join condition references several foreign keys. For example, if table T1 has a foreign key FK1 point to unique key UK1 in table T2 and table T2 has a foreign key FK2 pointing to unique key UK2 in T1, the resulting join condition

```
T1.A = T2.A AND T1.B = T2.B /*All instances of FK1 -> UK1 are reduced to one where
clause*/ AND
T2.B = T1.B AND T2.C = T1.C /*All instances of FK2 -> UK2 are reduced to one where
clause*/
```

is generated by the Joiner operator as

```
T2.A = T2.A AND T1.B = T2.B AND T2.C = T1.C
```

The Joiner operator contains the following properties:

- **Join Condition:** The text expression template for the Join Condition. For code generation, the input attributes are replaced by the source columns. The expression is a valid SQL expression that can be used in a WHERE clause.

> **Note:** The join condition is defined in a PL/SQL context. For SAP sources, Warehouse Builder can generate ABAP code by interpreting the PL/SQL join condition in the ABAP context. ABAP can only join over defined foreign key relationships.

Attributes in the Joiner operator contain the following properties:

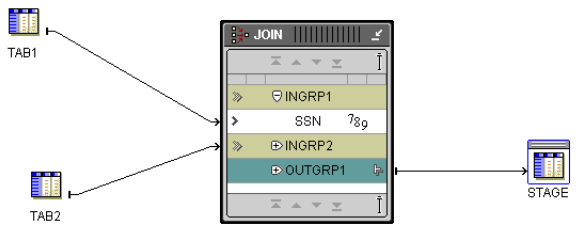
- **Data Type:** The data type of the attribute.
- **Precision:** The precision of the attribute, used for numeric type attributes only.
- **Scale:** The scale of the attribute, used for numeric type attributes only.
- **Length:** The length of the attributes, used for string type attributes only.

To use 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.

   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.

Figure 8–8 Joiner Operator in a Mapping



4. Right-click the Joiner operator header and select **Operator Properties.**

   The Joiner properties window displays.
5. Enter a join condition in the Join Condition field or click **...** to define an expression using the Expression Builder.
6. Close the Joiner property window.

Joiner Restrictions

The join condition expression cannot contain aggregation functions, such as SUM. Compile errors result when deploying the generated code for the mapping. A Joiner can have an unlimited number of input groups but only one output group.

The order of input groups in a joiner is used as the join order. The major difference between ANSI join and an Oracle join is that ANSI join must clearly specify join order. An Oracle join does not require it.

The filter condition is applied after join. For example, consider the following join:

```
Input1.c --- +
Input2.c --- +---> Joiner
Input3.c --- +
```

with the following conditions:

- **Condition 1:** Input1.c (+) = Input2.c (+)
- **Condition 2:** Input2.c = Input3.c
- **Condition 3:** Input1.c is null

The first two conditions are true joins while the third is a filter condition. If ANSI code is to be generated, Warehouse Builder interprets the statement as

```
select ...
from Input1 full outer join Input2 on (Input1.c = Input2.c)
join Input3 on (Input2.c = Input3.c)
where Input1.c is not null;
```

Specifying a Full Outer Join

If your target warehouse is based on Oracle9*i* or a later version, the Warehouse Builder joiner 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. For example,

```
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.

Note: The relational operator is not restricted to equality. You can also use other operators such as, >, <, !=, >=, <= .

When using the Oracle SQL syntax for partial outer join such as T1.A = T2.B (+), if you place a (+) sign on both sides of the relational operator, it is invalid Oracle SQL syntax. However, Warehouse Builder translates any condition with the double (+) sign into ANSI SQL syntax. For example,

```
SELECT ...
FROM T1 FULL OUTER JOIN T2 ON (T1.A = T2.B);
```

- When using full outer join, keep in mind the following:

- If you specify a full outer join condition for a non-Oracle9*i* target system type, you will receive a validation error and the code will not be generated.
- 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;
  ```

- The input group order is used as the ANSI join order. If you are joining more than two input groups, the order of the input groups determines the join order. Unlike Oracle's proprietary join syntax, the ANSI join syntax requires you to explicitly specify the join order.
- When you create a joiner, you must order the input groups exactly in the order you want to join them. For example, if you create three input groups in the order T1, T2, T3 and the join condition is

  ```
  T1.A (+) = T2.A (+) and T2.A = T3.A
  ```

  Warehouse Builder generates the following:

```
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.

- 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 (+) amd T2.B = T3.B (+)` will cause validation errors and code will not be generated.

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 Oracle*8i*, 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 Oracle*8i*. 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 Database database. Specifying a full outer join to an Oracle*8i* 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 8-13 for adding a Joiner operator.
2. Enter a full outer join statement in the Join Condition field or click **...** to define an expression using the Expression Builder.
3. Close the Joiner operator property window.

Key Lookup Operator

Use the Key Lookup operator to lookup data from a table, view, cube, or dimension.

For example, you can use the Key Lookup operator when you define a mapping that loads a cube and when you define surrogate keys on the dimension. In this example, you create a Key Lookup operator that looks up the surrogate key in the dimension table and returns the corresponding original record to form the foreign key relationship.

The table, view, cube, or dimension is bound to the Key Lookup operator. You can have multiple Key Lookup operators in the same mapping.

The key that you look up can be any unique value. It does not need to 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.

The output of the Key Lookup operator corresponds to the columns in the lookup object. If multiple rows in the lookup table match the key inputs, the cardinality of the output differs from the input. This produces results inconsistent with the data flowing into the target operator and generates an error at runtime. To ensure that only a single lookup row is found for each key input row, use keys in your match condition.

You can use inbound reconciliation on Key Lookup outputs. Outbound reconciliation is disabled. See "Reconciling Operators and Repository Objects" beginning on page 6-28 for more information.

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.

When you validate this operator:

- Warehouse Builder displays a warning if the condition does not use a complete unique key in the lookup table. If a unique key is not used, multiple rows may be retrieved for a single input row.
- Warehouse Builder displays a warning if the condition contains an equal comparison between attributes of mismatched data types. Although the Oracle database performs implicit conversions, the results may cause a runtime error.

To use a Key Lookup operator in a mapping:

1. Drop a **Key Lookup** operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Mapping Key Lookup dialog.

2. Use the Add Mapping Key Lookup dialog to select one or more tables.

   Warehouse Builder adds to the mapping canvas one Key Lookup operator for each table you select. For more information on using the Add Mapping Key Lookup dialog, see "Adding Bindable Operators" on page 6-7.

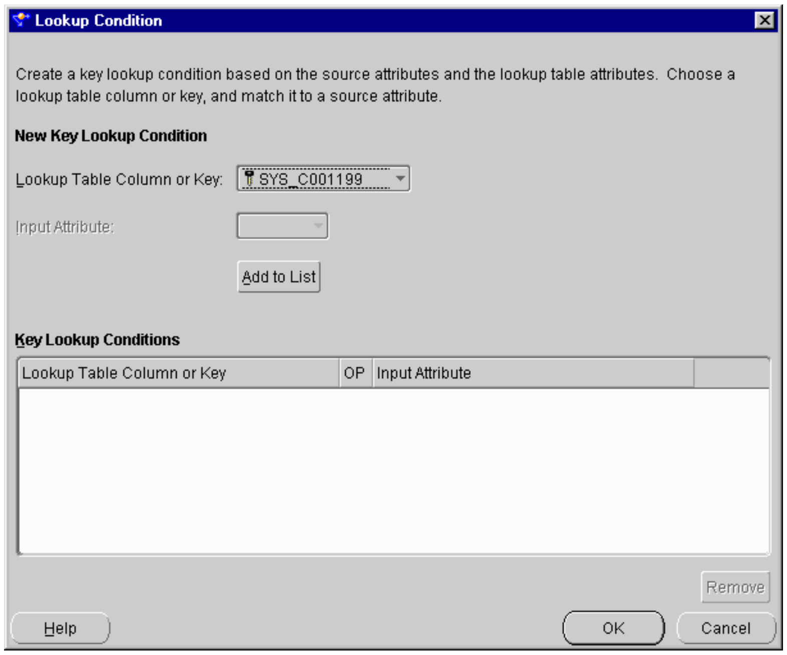
3. Connect the source attributes to the input group of the Key Lookup operator.
4. Right-click the Key Lookup operator header and select **Operator Properties.**

   Warehouse Builder displays the Key Lookup Properties window.

5. Click the field to the right of the Lookup Condition and click the **...** button.

   The Lookup Condition dialog displays as shown in Figure 8–9.

Figure 8–9 Lookup Condition Dialog


Lookup Condition

Create a key lookup condition based on the source attributes and the lookup table attributes. Choose a lookup table column or key, and match it to a source attribute.

New Key Lookup Condition

Lookup Table Column or Key: SYS_C001199

Input Attribute:

Key Lookup Conditions

Lookup Table Column or Key	OP	Input Attribute

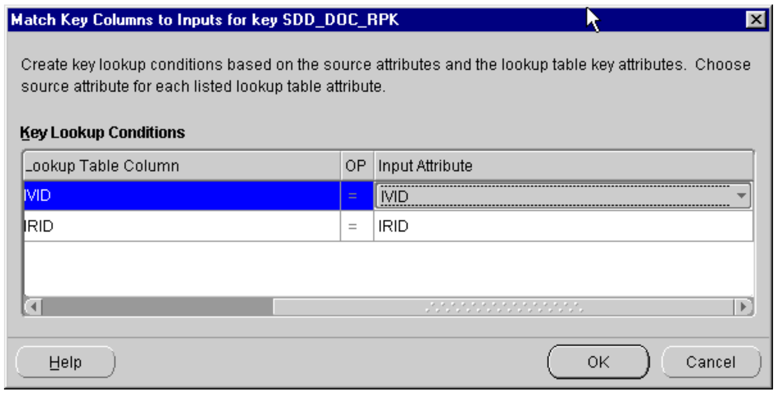
6. Select the lookup entity attribute from the **Lookup Table Column or Key** drop-down list shown in Figure 8–10.

   Choose attributes to compare to the selected lookup table column.

   For a non-composite key, select an input attribute or a key from the drop-down list. Click **Add to List.** The column or input pairs are added to the table at the bottom of the main dialog.

   For a composite key, click **Add to List.** The Match Key Columns to Input for Key dialog displays. Select the key input attributes from the **Input Attribute** drop-down list. Click **OK.**

Figure 8–10 Match Key Columns to Input for Key Dialog


Match Key Columns to Inputs for key SDD_DOC_RPK

Create key lookup conditions based on the source attributes and the lookup table key attributes. Choose source attribute for each listed lookup table attribute.

Key Lookup Conditions

Lookup Table Column	OP	Input Attribute
IVID	=	IVID
IRID	=	IRID

7. Click **OK** to close the Lookup Condition dialog.

Mapping Advanced Queue Operator

Mapping Advanced Queue (AQ) operators are used to propagate messages in queues from source to target systems.

In Warehouse Builder, you can use AQs as sources or targets. You can map an AQ source object to a target table or a staging table and then deploy the mapping to your target database. When you run the AQ mapping, the changes are propagated from the AQ to the target table in your warehouse. After a mapping runs successfully, subsequent invocations of the same mapping will not see the messages that have already been processed. If the mapping fails, no messages will be dequeued from the source AQs.

Warehouse Builder also enables you to use AQs as targets in your warehouse. Source AQs in your repository may represent a central AQ that integrates data from different messaging systems or applications. Warehouse Builder enables you to map this source AQ to a target AQ and propagate the messages from one type of messaging system to another.

For more information, see:

- Using Advanced Queues on page 3-33
- About Advanced Queues on page 3-34
- Creating Definitions for Advanced Queues on page 3-34

Creating AQ Mappings

To use a Mapping Advanced Queue operator in a mapping:

1. Drag and drop the Mapping Advanced Queue operator icon onto the mapping canvas.

   The Add Mapping Advanced Queue dialog displays.

2. You can choose to import new AQ definitions into the repository or select a previously imported AQ from the repository.

   Each AQ bound to an AQ operator in a mapping must belong to the same warehouse module as the mapping. If not, you will receive an error when you validate the mapping.

   For more information on how to use the Add Mapping Advanced Queue dialog, see "Adding Bindable Operators" on page 6-7.

3. Click **OK.**
4. Connect the output attribute of a Mapping AQ operator to the input group of a target operator.

Example of an AQ in a Mapping

A local company assigns a customer service representative to every customer by region. As the customer base increases, the company hires new service representatives to manage new and existing customers. The new service representative assignments to customers are tracked by using queues. You create an AQ with messages capturing the changes made to the Customer table.

You can import this AQ into Warehouse Builder and use it as a data source in a mapping. Connect the AQ source to the Customer target table on the mapping editor.Warehouse Builder generates PL/SQL scripts for this mapping, which you can deploy to your target warehouse. When you run the mapping, the messages are dequeued and the new assignments of the service representatives in the source Customer table are propagated to the Customer table in the target database.

If you register each mapping in Warehouse Builder as a separate subscriber of each AQ, then different mappings referencing the same AQ do not interfere with each other. Each mapping sees all the messages in the AQ published to all its subscribers.

Reconciling Advanced Queues

You can perform an inbound reconciliation on an AQ operator to update it with any changes in the repository definition of the AQ object to which it is bound. For more information, see "Inbound Reconciliation" on page 6-29. Warehouse Builder does not currently enable you to outbound reconcile an AQ operator.

Advanced Queue Operator Properties

You can configure the following properties for a Mapping AQ operator used in a mapping:

- From the Mapping Editor, right-click the AQ operator and select **Operator Properties.**

  Warehouse Builder displays the Mapping Advanced Queue Properties dialog containing the Bound Name property. This name is used to identify the AQ during code generation.

- From the Mapping Editor, right-click the AQ operator and select **Edit.**

  Warehouse Builder displays the Mapping Advanced Queue Editor containing three tabs: General, Groups, Input/Output.

General

Rename the Mapping AQ operator by highlighting the previous name and typing over it. Type an optional description for the Mapping AQ operator.

Groups

Rename the Mapping AQ operator group by highlighting the previous name and typing over it. Each AQ operator has exactly one INOUT group. This is a read-only field. Type an optional description for the Mapping AQ operator group.

Input/Output

You cannot add more attributes to the INOUT group of an AQ operator. You can view and edit the attribute name, datatype, length, precision, scale, and optional description.

AQ Prerequisites for Mapping Execution

AQs deployed from Warehouse Builder are proxy AQs that represent the AQ source on a remote system. Because Oracle does support remote dequeuing of messages, you need to first deploy an agent of the source AQ in your local schema and register the locally deployed proxy AQ as a subscriber to the AQ on the source system. For the Warehouse Builder AQ mappings to run correctly, you must follow these steps in your source and warehouse systems:

- Grant the following privileges to your AQ source system user:

  ```
  execute on dbms_aq
  execute on dbms_aqadm
  ```

```
execute on aq$_agent
```

- Grant the following privileges to your warehouse system user:

  ```
  execute on dbms_aq
  execute on dbms_aqadm
  execute on aq$_agent
  ```

- From your AQ source system, create a database link to the AQ target system.

  The target user can now deploy the target proxy AQ to the target instance.
- When you configure AQs for deployment, make sure that both the AQ object and the mapping that has AQ operators bound to this AQ object are deployed in the same location.
- Before deploying a mapping, you must successfully deploy the temporary tables associated with the AQ object to which the Mapping AQ operator is bound.
- After deploying the proxy AQ to the target system, you must register the locally deployed proxy AQ as a subscriber to the AQ on the source system. See "Registering AQ as a Subscriber" on page 8-21.
- Concurrent executions of the same mapping using the AQ source operators is not supported.
- If an AQ is a target in a mapping, make sure that this target AQ has at least one subscriber before the mapping is executed. Otherwise, you will receive a runtime error during mapping execution.
- The maximum number of subscribers to an AQ cannot exceed 1024. For an AQ q1, the number of mappings that have a source AQ staging component bound to q1 should be less than 1024.
- The upgrade of a mapping which contains an AQ may result in the loss of data contained in the AQ before the upgrade. This is because the subscriber of the proxy AQ will be dropped and re-created during the upgrade.

Registering AQ as a Subscriber

If an AQ is defined in your source system, you need to follow these manual steps to register the deployed proxy AQ as a subscriber to the source AQ and schedule propagation of the AQ to your target warehouses:

1. Create a DB link from your source system to your data warehouse.
2. Register the AQ as a subscriber to all messages.
3. Schedule propagation of AQ from source to the target system.

The following is an example of how you can register the AQ as a subscriber and schedule its propagation to the target system.

```
declare
subscriber sys.aq$_agent;
begin
subscriber := sys.aq$_agent('subscribername','schema.queue2',0); --
schema.queue2 refers to the queue that is subscribing
dbms_aqadm.add_subscribe(queue_name=>'QUEUE1',subscriber=>subscriber,rule=>'
',transformation=>''); -- QUEUE1 refers to the queue that 'queue2 is
subscribing to
dbms_aqadm.schedule_propagation('SCHEMA.QUEUE1',NULL,SYSDATE,'','','60'); --
SCHEMA.QUEUE1 are the schema and queuename of queue1. This command starts
the message propagation from queue1 to queue2 with a latency of 60 seconds
```

```
end;
```

Mapping Flat File Operator

You can use a Mapping Flat File operator as either a source or target.

However, the two are mutually exclusive within the same mapping. There are differences in code generation languages for flat file sources and targets. Subsequently, mappings can contain a mix of flat files, relational objects, and transformations, but with the restrictions discussed further in this section.

Flat File Source Operators

As a source, the Mapping Flat File operator acts as the row set generator that reads from a flat file using the SQL*Loader utility. You cannot use a flat file source operator if you are mapping to a flat file target or to an external table. When you design a mapping with a flat file source, you can use the following operators:

- Filter Operator on page 8-11
- Constant Operator on page 8-7
- Data Generator Operator on page 8-8
- Mapping Sequence Operator on page 8-37
- Expression Operator on page 8-10
- Transformation Operator on page 8-83
- 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. You can either use an existing flat file, or identify a new flat file as a target during design and define it using outbound reconciliation. See "Outbound Reconciliation" on page 6-31.

Note: A mapping can contain a maximum of 50 flat file target operators at one time.

When you use a flat file as a target, understanding both flat files and external tables will help you determine which feature to use. 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. Refer to "External Tables versus Flat File Operators" on page 3-19 for more information.

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 either drop the same flat file into the mapping as a target again and map to a different record type, or create a separate mapping for each record type you want to load.

You have the following options for Mapping 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

Using Previously Imported Flat Files

This scenario describes using a flat file object that has been previously imported and sampled. You can find instructions for importing and sampling flat file objects in Chapter 4, "Importing Data Definitions".

To use a previously imported flat file as a source or a target:

1. Drag and drop a Mapping Flat File operator onto your mapping editor canvas.

   The Add File dialog displays.
2. Click **Select from existing repository file and bind.**
3. Highlight the desired flat file object.
4. Click **OK.**
5. Prior to deployment, you must follow the steps for "Configuring Flat File Operators" on page 11-9 and for "Configuring Mappings Reference" on page 11-1.

Importing and Binding New Flat Files into Your Mapping

In this option, you use a flat file as either a source or target, but you have not yet imported or sampled the flat file. You import the flat file as you design your mapping.

To import and bind a new flat file into your mapping:

1. Drop a Mapping Flat File onto your mapping canvas.

   The Add File dialog displays.
2. Click **Import file into repository and bind.**
3. Highlight the module into which you want to import the flat file.
4. Click **OK.**

   The Import File Wizard Welcome page displays. For more information on importing Flat Files, see "Using the Import Metadata Wizard Flat Files" on page 4-21.
5. At the Welcome page, click **Next.**

   The Import File Wizard Object Selection page displays.
6. Navigate to the location and select the flat file.
7. Click **Next.**

   The Import File Wizard Summary and Import page displays. A red ball next to the file indicates that there is no metadata information available about the file structure.

- If the flat file has the same structure as another file you have already imported and sampled, click the **Same As** field and select the identical file from the drop list.
- If there are no previously imported or sampled files with this file structure, click **Sample** and follow the instructions for "Using the Flat File Sample Wizard" on page 4-23.

8. Click **Finish.**

   The Mapping Flat File operator appears on the mapping canvas.

9. Prior to deployment, you must follow the steps for "Configuring Flat File Operators" on page 11-9 and for "Configuring Mappings Reference" on page 11-1.

Defining New Flat File Sources or Targets in Mappings

As you build your mapping, you can create a new flat file object by selecting the **Create unbound mapping file with no attributes** option. Using this method, the only option is to create a comma-delimited, single-record-type flat file operator. You can leave the flat file unbound, or you can outbound reconcile to the repository.

Note: You can only perform outbound reconciliation if the flat file is new to that repository. For more information on reconciliation, refer to "Reconciling Operators and Repository Objects" on page 6-28.

To use a new unbound Flat File operator in a mapping:

1. Drag and drop a Mapping Flat File operator onto the mapping canvas.

   The Add File dialog displays.

2. Click **Create unbound mapping File with no attributes.**
3. Enter the new file name.
4. Click **OK.**

   The Mapping Flat File operator with no attributes appears in your mapping.

5. Define attributes for the new flat file operator by right-clicking on the operator and choosing **Edit.** Enter the attributes manually on the Input/Output tab, or by auto-mapping them from another operator.

   For instructions on defining attributes, refer to "Editing Operators" on page 6-9.

6. Right-click the flat file operator and select **Reconcile Outbound** to create a new repository flat file.
7. Select the flat file module in which you want to create the new flat file.
8. Click **OK.**

   A new comma-delimited flat file is created in your repository.

9. Prior to deployment, you must follow the steps for "Configuring Flat File Operators" on page 11-9 and for "Configuring Mappings Reference" on page 11-1.

Extracting a Master-Detail Structure from Flat Files

If you are extracting data from a multiple-record-type file with a master-detail structure and mapping to tables, add a Mapping Sequence operator to the mapping to retain the relationship between the master and detail records through a surrogate primary key or foreign key relationship. A master-detail file structure is one where a master record is followed by its detail records. In Example 8–1, records beginning with "E" are master records with Employee information and records beginning with "P" are detail records with Payroll information for the corresponding employee.

Example 8–1 A Multiple-Record-Type Flat File with a Master-Detail Structure

```
E 003715 4 153 09061987 014000000 "IRENE HIRSH" 1 08500
P 01152000 01162000 00101 000500000 000700000
P 02152000 02162000 00102 000300000 000800000
E 003941 2 165 03111959 016700000 "ANNE FAHEY" 1 09900
P 03152000 03162000 00107 000300000 001000000
E 001939 2 265 09281988 021300000 "EMILY WELLMET" 1 07700
P 01152000 01162000 00108 000300000 001000000
P 02152000 02162000 00109 000300000 001000000
```

In Example 8–1, the relationship between the master and detail records is inherent only in the physical record order: payroll records correspond to the employee record they follow. However, if this is the only means of relating detail records to their masters, this relationship is lost when Warehouse Builder loads each record into its target table.

Maintaining Relationships Between Master and Detail Records

You can maintain the relationship between master and detail records if both types of records share a common field. If Example 8–1 contains a field Employee ID in both Employee and Payroll records, you can use it as the primary key for the Employee table and as the foreign key in the Payroll table, thus associating Payroll records to the correct Employee record.

However, if your file does not have a common field that can be used to join master and detail records, you must add a sequence column to both the master and detail targets (see Table 8–2 and Table 8–3) to maintain the relationship between the master and detail records. Use the Mapping Sequence operator to generate this additional value.

Table 8–2 represents the target table containing the master records from the file in Example 8–1 on page 8-25. The target table for the master records in this case contains employee information. Columns E1-E10 contain data extracted from the flat file. Column E11 is the additional column added to store the master sequence number. Notice that the number increments by one for each employee.

Table 8–2 Target Table Containing Master Records

E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11
E	003715	4	153	09061987	014000000	"IRENE	HIRSH"	1	08500	1
E	003941	2	165	03111959	016700000	"ANNE	FAHEY"	1	09900	2
E	001939	2	265	09281988	021300000	"EMILY	WELSH"	1	07700	3

Table 8–3 represents the target table containing the detail records from the file in Example 8–1 on page 8-25. The target table for the detail records in this case contains payroll information, with one or more payroll records for each employee. Columns P1-P6 contain data extracted from the flat file. Column P7 is the additional column added to store the detail sequence number. Notice that the number for each payroll record matches the corresponding employee record in Table 8–2.

Table 8–3 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 "Direct Path Loading for Performance" **on page 8-30**.

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 Flat Files" on page 4-21.
- For additional information on using the Mapping Flat File as a source, see "Flat File Source Operators" on page 8-22.
- For additional information on using Mapping Tables, see "Adding Bindable Operators" on page 6-7.
- For additional information on using Mapping Sequences, see "Mapping Sequence Operator" on page 8-37.
- For additional information on configuring mappings, see "Configuring Mappings Reference" on page 11-1.

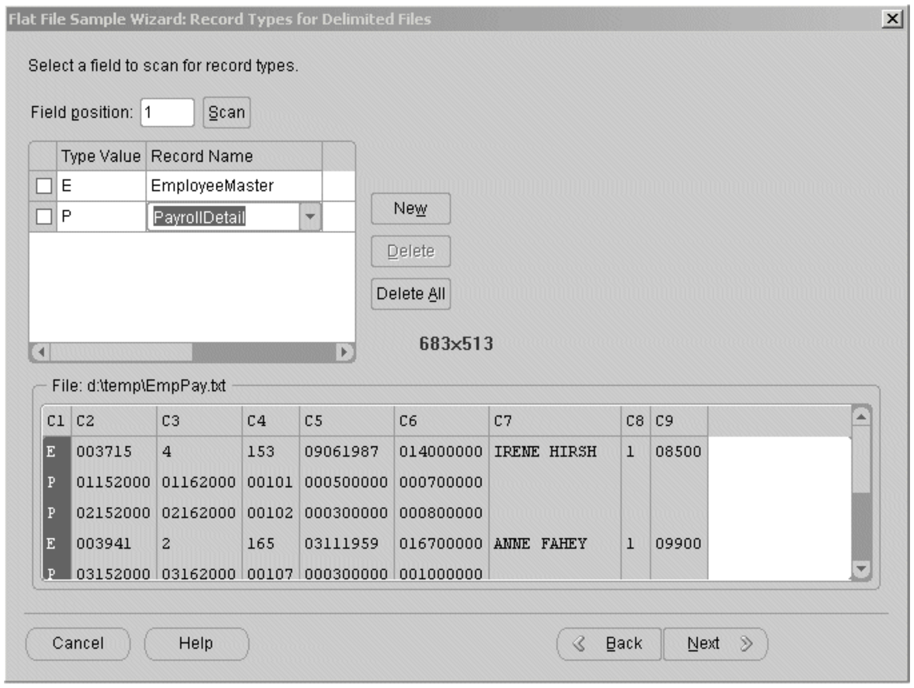
To extract from a master-detail flat file and maintain master-detail relationships:

1. Import and sample a flat file source that consists of master and detail records.

   When naming the record types as you sample the file, assign descriptive names to the master and detail records, as shown in Figure 8–11. This makes it easier to identify those records in the future.

   Figure 8–11 shows the Flat File Sample Wizard for a multiple-record-type flat file containing department and employee information. The master record type (for employee records) is called EmployeeMaster, while the detail record type (for payroll information) is called PayrollDetail.

Figure 8–11 Naming Flat File Master and Detail Record Types


Flat File Sample Wizard: Record Types for Delimited Files
Select a field to scan for record types.
Field position: 1
Scan
New
Delete
Delete All
683x513
Cancel
Help
Back
Next

Type Value	Record Name
E	EmployeeMaster
P	PayrollDetail

File: d:\temp\EmpPay.txt

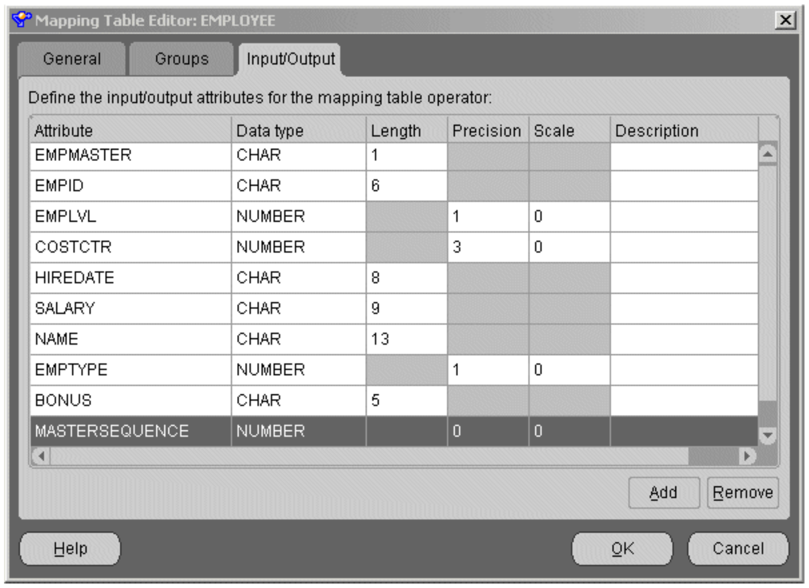
C1	C2	C3	C4	C5	C6	C7	C8	C9
E	003715	4	153	09061987	014000000	IRENE HIRSH	1	08500
P	01152000	01162000	00101	000500000	000700000			
P	02152000	02162000	00102	000300000	000800000			
E	003941	2	165	03111959	016700000	ANNE FAHEY	1	09900
P	03152000	03162000	00107	000300000	001000000			

2. Drop a Mapping Flat File operator onto the mapping editor canvas and specify the master-detail file from which you want to extract data.
3. Drop a Mapping Sequence operator onto the mapping canvas.
4. Drop a Mapping Table operator for the master records onto the mapping canvas.

   You can either select an existing repository table that you created earlier or create a new unbound mapping table with no attributes. You can then map or copy all required fields from the master record of the file operator to the master table operator (creating columns) and perform an outbound reconciliation to define the table later.

   The table must contain all the columns required for the master fields you want to load plus an additional numeric column for loading sequence values, as shown in Figure 8–12.

Figure 8–12 Adding a Sequence Column to the Master and Detail Target Tables



Define the input/output attributes for the mapping table operator:

Attribute	Data type	Length	Precision	Scale	Description
EMPMASTER	CHAR	1			
EMPID	CHAR	6			
EMPLVL	NUMBER		1	0	
COSTCTR	NUMBER		3	0	
HIREDATE	CHAR	8			
SALARY	CHAR	9			
NAME	CHAR	13			
EMPTYPE	NUMBER		1	0	
BONUS	CHAR	5			
MASTERSEQUENCE	NUMBER		0	0	

5. Drop a Mapping Table operator for the detail records onto the mapping canvas.

   You can either select an existing repository table that you created earlier or create a new unbound mapping table 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 reconcile 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.

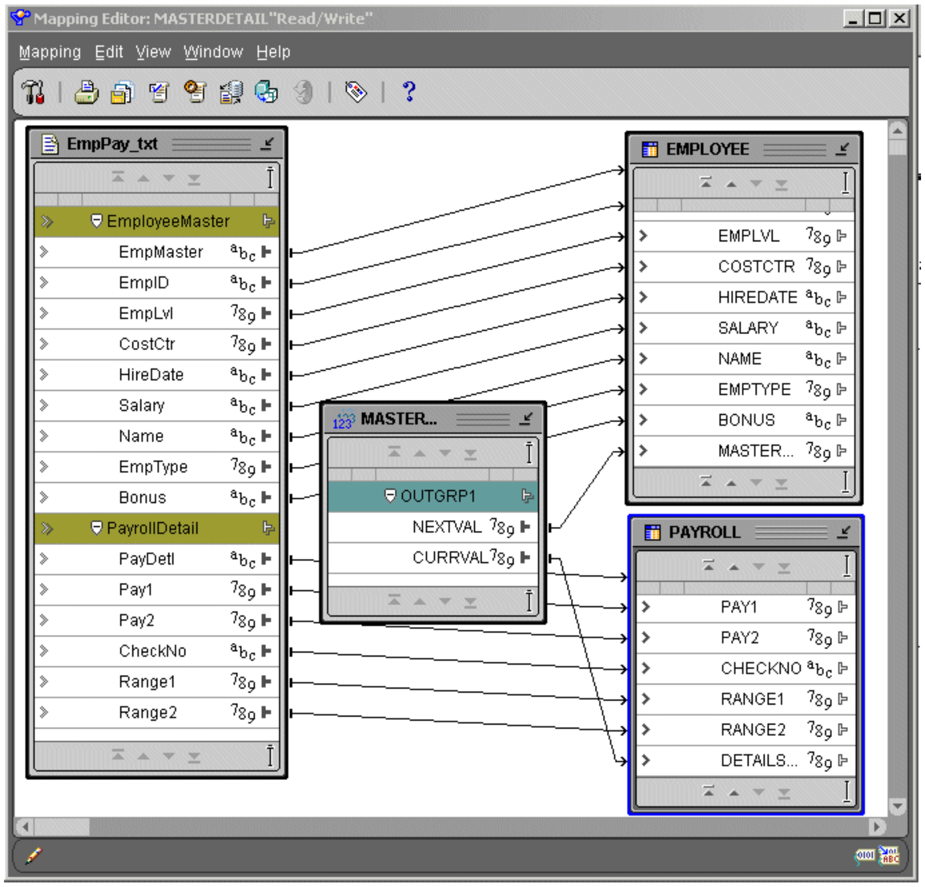
6. Map all of the necessary flat file master fields to the master table and detail fields to the detail table, as shown in Figure 8–13.

7. Map the Mapping Sequence `NEXTVAL` attribute to the additional sequence column in the master table, as shown in Figure 8–13.

8. Map the Mapping Sequence `CURRVAL` attribute to the additional sequence column in the detail table, as shown in Figure 8–13.

   Figure 8–13 shows a completed mapping with the flat file master fields mapped to the master target table, the detail fields mapped to the detail target table, and the `NEXTVAL` and `CURRVAL` attributes from the Mapping Sequence mapped to the master and detail target tables, respectively.

Figure 8–13 Completed Mapping from Master-Detail Flat File to Two Target Tables



9. Configure the mapping with the following parameters:

   Direct Mode: False

   Errors Allowed: 0

   Row: 1

   Trailing Nullcols: True (for all tables)

Error Handling Suggestions

This section contains error handling recommendations for files with varying numbers of errors.

If your data file almost never contains errors:

1. Create a mapping with a Sequence operator (see "Mapping Sequence Operator" on page 8-37).
2. Configure a mapping with the following parameters:

   Direct Mode= false

   ROW=1

   ERROR ALLOWED = 0
3. Generate the code and run an SQL Loader script.

   If the data file has errors, the loading stops when the first error happens.

4. Fix the data file and run the control file again with the following configuration values:

   CONTINUE_LOAD=TRUE

   SKIP=*number of records already loaded*

If your data file is likely to contain a moderate number of errors:

1. Create a primary key (PK) for the master record based on the `seq_nextval` column.
2. Create a foreign key (FK) for the detail record based on the `seq_currval` column which references the master table PK.

   In this case, master records with errors will be rejected with all their detail records. You can recover these records by following these steps.
3. Delete all failed detail records that have no master records.
4. Fix the errors in the bad file and reload only those records.
5. If there are very few errors, you may choose to load the remaining records and manually update the table with correct sequence numbers.
6. In the log file, you can identify records that failed with errors because those errors violate the integrity constraint. The following is an example of a log file record with errors:

```
Record 9: Rejected - Error on table "MASTER_T", column "C3".
ORA-01722: invalid number
Record 10: Rejected - Error on table "DETAIL1_T".
ORA-02291: integrity constraint (SCOTT.FK_SEQ) violated - parent key not found
Record 11: Rejected - Error on table "DETAIL1_T".
ORA-02291: integrity constraint (SCOTT.FK_SEQ) violated - parent key not found
Record 21: Rejected - Error on table "DETAIL2_T".
ORA-02291: invalid number
```

If your data file always contains many errors:

1. Load all records without using the Mapping Sequence operator.

   Load the records into independent tables. You can load the data in Direct Mode, with the following parameters that increase loading speed:

   ROW>1

   ERRORS ALLOWED=MAX
2. Correct all rejected records.
3. Reload the file again with a Sequence operator (see "Mapping Sequence Operator" on page 8-37).

Direct Path Loading for Performance

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 Flat Files" on page 4-21.
- For additional information on using the Mapping Flat File as a source, see "Flat File Source Operators" on page 8-22.
- For additional information on using Mapping Tables, see "Adding Bindable Operators" on page 6-7.
- For additional information on using the Data Generator operator, see "Data Generator Operator" on page 8-8.
- For additional information on using the Constant operator, see "Constant Operator" on page 8-8.
- For additional information on configuring mappings, see "Configuring Mappings Reference" on page 11-1.

To extract from a master-detail flat file using direct path load to maintain master-detail relationships:

1. Import and sample a flat file source that consists of master and detail records.

   When naming the record types as you sample the file, assign descriptive names to the master and detail records, as shown in Figure 8–11 on page 8-27. This will make it easier to identify those records in the future.

2. Drop a Mapping Flat File operator onto the mapping canvas and specify the master-detail file from which you want to extract data.
3. Drop a Data Generator and a Constant operator onto the mapping canvas.
4. Drop a Mapping Table operator for the master records onto the mapping canvas.

   You can either select an existing repository table that you created earlier, or create a new unbound mapping table with no attributes and perform an outbound reconcile to define the table later.

   The table must contain all the columns required for the master fields you plan to load plus an additional numeric column for loading the `RECNUM` value.

5. Drop a Mapping Table for the detail records onto the mapping canvas.

   You can either select an existing repository table that you created earlier, or create a new unbound mapping table with no attributes and perform an outbound reconcile to define the table later.

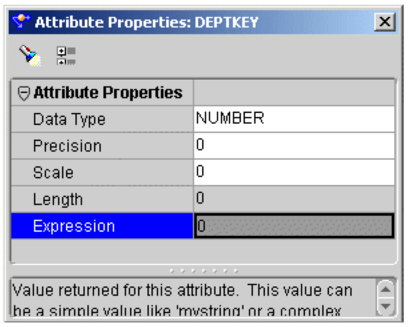
   The table must contain all the columns required for the detail fields you plan to load plus an additional numeric column for loading a `RECNUM` value, and a column that will be updated with the unique identifier of the corresponding master table row.

6. Map all of the necessary flat file master fields to the master table and detail fields to the detail table, as shown in Figure 8–15 on page 8-32.
7. Map the Data Generator operator's `RECNUM` attribute to the `RECNUM` columns in the master and detail tables, as shown in Figure 8–15 on page 8-32.
8. Add a constant attribute in the Constant operator.

   If the master row unique identifier column is of a `CHAR` datatype, make the constant attribute a `CHAR` type with the expression `'*'`.

   If the master row unique identifier column is a number, make the constant attribute a `NUMBER` with the expression `'0'`. Figure 8–14 shows the expression property of the constant attribute set to `'0'`. This constant marks all data rows as "just loaded."

Figure 8–14 Constant Operator Properties



Attribute Properties: DEPTKEY

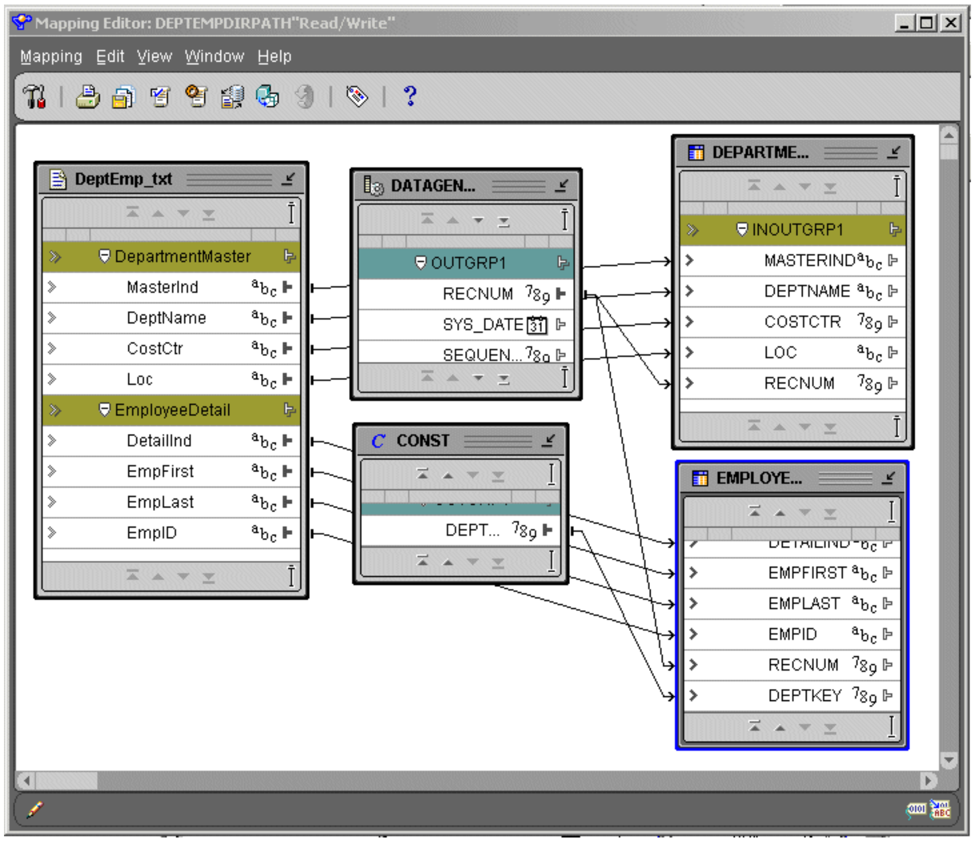
Attribute Properties	
Data Type	NUMBER
Precision	0
Scale	0
Length	0
Expression	0

Value returned for this attribute. This value can be a simple value like 'mystring' or a complex

9. Map the constant attribute from the Constant operator to the detail table column that will later store the unique identifier for the corresponding master table record.

   Figure 8–15 shows a completed mapping with the flat file's master fields mapped to the master target table, the detail fields mapped to the detail target table, the `RECNUM` attributes from the Data Generator operator mapped to the master and detail target tables, respectively, and the constant attribute mapped to the detail target table.

Figure 8–15 Completed Mapping from Master-Detail Flat File with a Direct Path Load



10. Configure the mapping with the following parameters:

Direct Mode: True

Errors Allowed: 0

Trailing Nullcols: True (for each table)

11. 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.
12. Run the SQL*Loader script.
13. 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;
  /
```

Subsequent Operations

After the initial loading of the master and detail tables, you can use the loaded sequence values to further transform, update, or merge master table data with detail table data. For example, if your master records have a column that acts as a unique identifier (such as an Employee ID), and you want to use it as the key to join master and detail rows (instead of the sequence field you added for that purpose), you can update the detail table(s) to use this unique column. You can then drop the sequence column you created for the initial load. Operators such as the Aggregator, Filter, or Match and Merge operator can help you with these subsequent transformations.

Mapping Input Parameter Operator

You can introduce information external to Warehouse Builder as input into a mapping using a Mapping Input Parameter. For example, you can use an Input Parameter operator to pass SYSDATE to a mapping that loads data to a staging area. Use the same Input Parameter to pass the timestamp to another mapping that loads the data to a target.

When you a generate mapping, Warehouse Builder creates PL/SQL package. Mapping input parameters become part of the signature of the main procedure in the package.

The Mapping Input Parameter has a cardinality of one. It creates a single row set that can be combined with another row set as input to the next operator.

The names of the input attributes become the names of the mapping output parameters. The parameters can be used by connecting the attributes of the Mapping Input Parameters operator within the mapping editor. You can have only one Mapping Input Parameter in a mapping.

The Mapping Input Parameter contains the following properties:

- **Default Value:** The character string value which, if specified, is placed in the generated code as the default value for the specified attribute. For example, if the value entered is ‘1-JUN-2001’ then the generated code contains DEFAULT ‘1-JUN-2001’.

The default value for the mapping input parameter appears in the DEFAULT clause following the function parameter declarations in the generated PL/SQL package. For example, if a mapping parameter named param1 with data type VARCHAR2 is defined with a default value of 'HELLO', the generated main function in the PL/SQL package appears as:

```
param1 IN VARCHAR2 DEFAULT 'HELLO'
```

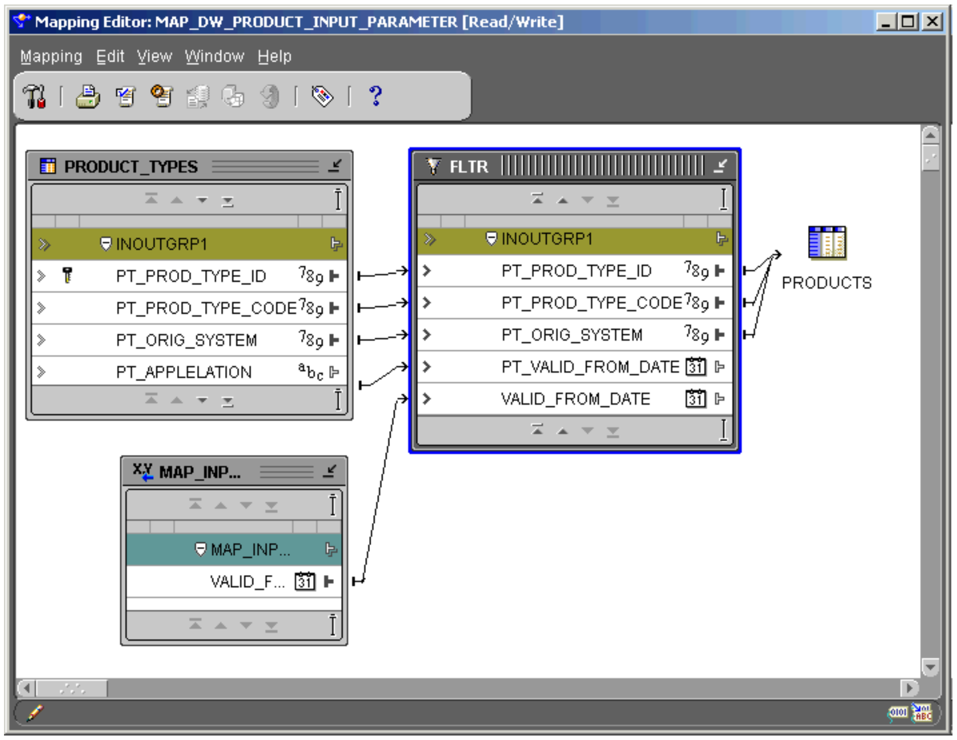
- **Data Type:** Specifies the data type for this input parameter.

To use a Mapping Input Parameter operator in a mapping:

1. Drag and drop a Mapping Input operator onto the Mapping Editor canvas.
2. Right-click the Mapping Input operator and select **Edit.**
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. Connect the Input Parameter operator MAP_INPUT attribute to a group in the target operator as shown in Figure 8–16.

Figure 8–16 Mapping Editor Showing A Mapping Input Parameter



Mapping Output Parameter Operator

Use the Mapping Output Parameter operator to send values out of a mapping to applications external to Warehouse Builder. When you a generate mapping, Warehouse Builder creates a PL/SQL package. Mapping Output Parameters become part of the signature of the main procedure in the package.

The Mapping Output Parameter has only one input group. You can have only one Mapping Output Parameter in a map. Only attributes that are not associated with a row set can be mapped into a Mapping Output Parameter. 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. A Mapping Output Parameter is not valid for a SQL*Loader mapping.

The default value for the Mapping Output Parameter appears in the DEFAULT clause following the function parameter declarations in the generated PL/SQL package. For example, if a mapping parameter named `param1` with data type VARCHAR2 is defined with a default value of `'HELLO'`, the generated main function in the PL/SQL package appears as:

```
param1 OUT VARCHAR2 DEFAULT 'HELLO'
```

If a Mapping Output Parameter named `param1` has data type VARCHAR2, the generated main function in the PL/SQL package appears as:

```
param1 OUT VARCHAR2
```

The Mapping Output Parameter contains the following properties:

- **Data Type:** Specifies the data type for this output parameter.
- **Bound Name:** Specifies the physical name for this output parameter.

Note: Mapping Output Parameters cannot be mapped to any operator. They can be mapped from Constants, Mapping Input Parameters, or the Output of a Pre- or Post-Mapping Process including the return value.

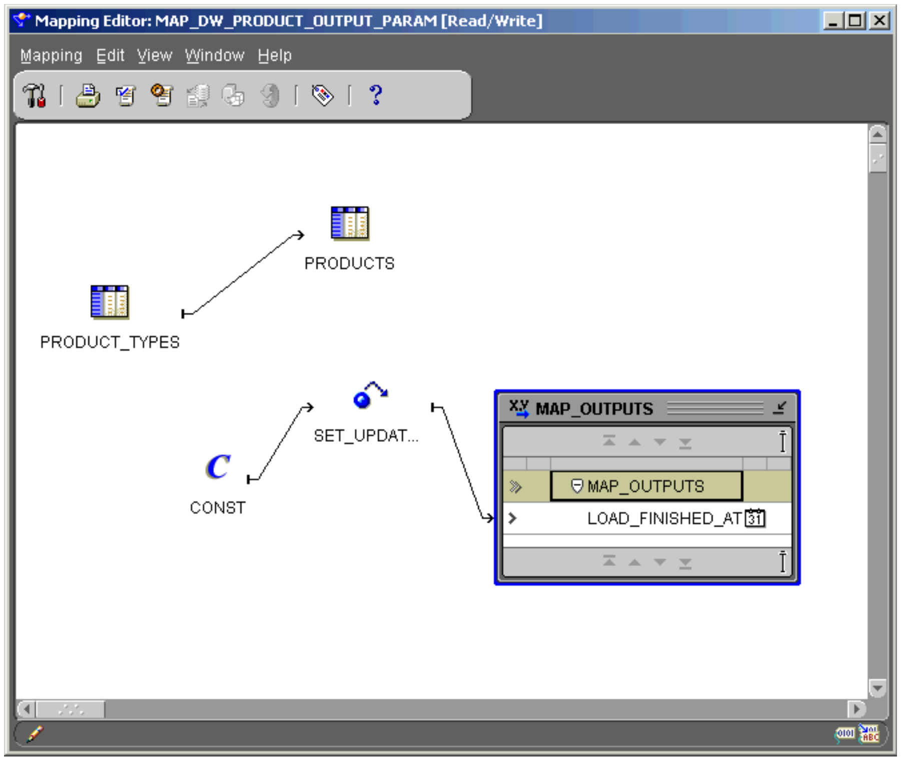
To use a Mapping Output Parameter operator in a mapping:

1. Drop a Mapping Output operator onto the Mapping Editor canvas.
2. Right-click the Mapping Output 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.

   See Figure 8–17 for an example of an Mapping Output Parameter in a mapping.

Figure 8–17 Mapping Editor Showing An Output Parameter Operator


Mapping Editor: MAP_DW_PRODUCT_OUTPUT_PARAM [Read/Write]
Mapping Edit View Window Help
PRODUCTS
PRODUCT_TYPES
SET_UPDAT...
CONST
MAP_OUTPUTS
MAP_OUTPUTS
LOAD_FINISHED_AT

Mapping Sequence Operator

A Mapping Sequence operator generates sequential numbers that increment for each row. For example, you can use the Sequence operator to create surrogate keys while loading data into a dimension table.

You can connect a Mapping 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.

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 reconcile Mapping Sequences to a repository sequence in one of the modules. The repository sequence must be generated and deployed before the mapping containing the Mapping Sequence is deployed to avoid errors in the generated code package. See "Adding Bindable Operators" on page 6-7 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 Reference" on page 11-3 for more information on configuring mode settings.

The Mapping Sequence contains the following property:

- **Bound Name:** The name of the sequence database object that is used in the generated code. If the sequence has been reconciled with a repository component, the Bound Name remains the same as the physical name of the repository sequence.

To use a Mapping Sequence operator in a mapping:

1. Drag and drop the Mapping Sequence operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Mapping Sequence dialog.

2. Use the Add Mapping Sequence dialog to create or select a sequence. For more information on these options, see "Adding Bindable Operators" on page 6-7.

3. Connect the sequence to a target attribute.

Match-Merge Operator

The Match-Merge operator is a data quality operator that you can use to first match and then merge data.

When you match records, you determine through business rules which records in a table refer to the same data. When you merge records, you consolidate into a single record the data from the matched records.

This section includes information and examples on how to use the Match-Merge operator in a mapping. The Match-Merge operator together with the Name-Address operator support householding, the process of identifying unique households in name and address data.

Example: Matching and Merging Customer Data

Consider how you could utilize the Match-Merge operator to manage a customer mailing list. Use matching to find records that refer to the same person in a table of customer data containing 10,000 rows. For example, you can define a match rule that screens records that have similar first and last names. Through matching you may discover that 5 rows refer to the same person. You can merge those records into one new record. For example, you can create a merge rule to retain the values from the one of the five matched records with the longest address. The newly merged table now contains one record for each customer.

Table 8–4 shows records that refer to the same person prior to using the Match-Merge operator.

Table 8–4 Sample Records

Row	FirstName	LastName	SSN	Address	Unit	Zip
1	Jane	Doe	NULL	123 Main Street	NULL	22222
2	Jane	Doe	111111111	NULL	NULL	22222
3	J.	Doe	NULL	123 Main Street	Apt 4	22222
4	NULL	Smith	111111111	123 Main Street	Apt 4	22222
5	Jane	Smith-Doe	111111111	NULL	NULL	22222

Table 8–5 shows the single record for Jane Doe after using the Match-Merge operator. Notice that the new record retrieves data from different rows in the sample.

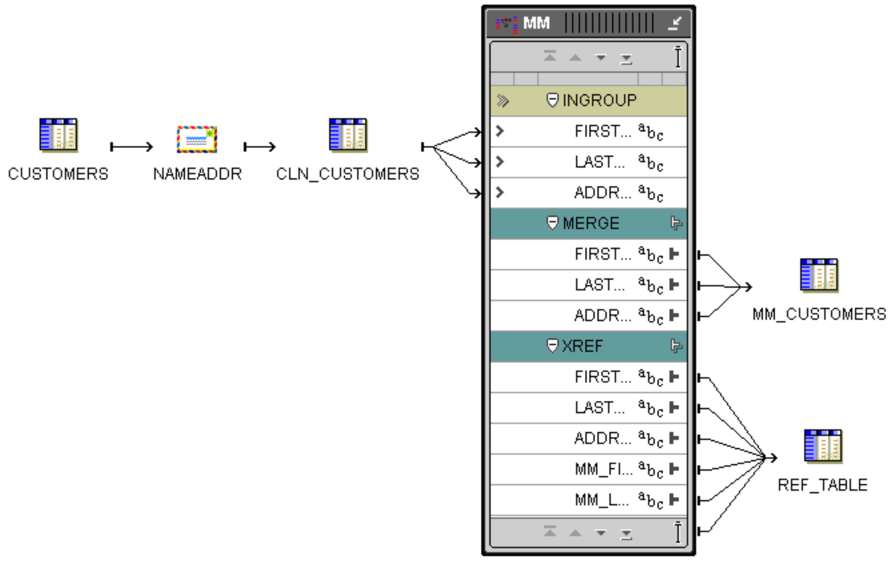
Table 8–5 Match-Merge Results

FirstName	LastName	SSN	Address	Unit	Zip
Jane	Doe	111111111	123 Main Street	Apt 4	22222

Designing Mappings with a Match-Merge Operator

Figure 8–18 shows a mapping you can design using a Match-Merge operator. Notice that the Match-Merge operator is preceded by a Name-Address operator, NAMEADDR, and a staging table, CLN_CUSTOMERS. You can design your mapping with or without a Name-Address operator. Preceding the Match-Merge operator with a Name-Address operator is desirable when you want to ensure your data is clean and standardized before launching time consuming match and merge operations.

Figure 8–18 Match-Merge Operator in a Mapping



Whether you include a Name-Address operator or not, be aware of the following considerations as you design your mapping:

- **PL/SQL output:** The Match-Merge operator can generate two outputs, both PL/SQL outputs only. The MERGE group includes the merged data. The XREF group is an optional group you can design to document the merge process.
- **Row based operating mode:** When the Match-Merge operator matches records, it compares each row with the subsequent row in the source and generates row based code only. These mappings, therefore, can only run in row based mode.
- **SQL based operators before Match-Merge:** The Match-Merge operator generates only PL/SQL outputs. If you want to include operators that generate SQL code only, you must design the mapping such that they precede the Match-Merge operator. For example, operators such as the Join, Key Lookup, and Set operators must precede the Match-Merge operator. A mapping designed with operators that generate set based code after a Match-Merge operator is invalid and Warehouse Builder does not generate code for such mappings.
- **SQL input:** With one specific exception, the Match-Merge operator requires SQL input. If you want to precede a Match-Merge with an operator that generates only

PL/SQL output such as the Name-Address operator, you must first load the data to 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 "Refining Data from Match-Merge Operators" on page 8-52.

Using the Match-Merge Operator

You have the following options for using a match-merge operator:

- **Define a new match-merge operator:** Drag match-merge operator from the Toolbox onto the mapping. The Mapping Editor launches a wizard.
- **Edit an existing match-merge operator:** Right-click the operator and select **Edit.** The Mapping Editor opens the Match-Merge Editor.

Whether you are using the operator wizard or the Operator Editor, complete the following pages:

- General
- Groups
- Input Connections
- Input Attributes
- Merge Output
- Cross Reference Output
- Match Bins
- Match Rules
- Merge Rules

General

Use the General page to specify a name and optional description for the operator. By default, the wizard names the match-merge operator "MM."

Groups

By definition, the Match-Merge operator has one input group and two output groups. You can rename the groups and add optional descriptions, but you cannot add or delete groups in the Match-Merge operator. The default names for the input group is INGRP1. The default names for the output groups are MERGE and XREF.

You assign attributes to the INGRP1 on the Input Connections page and then further edit those attributes on the Input Attributes page. You define attributes for the MERGE group on the Merge Output page. You can optionally assign attributes to the XREF group on the Cross Reference Output page.

Input Connections

Use the Input Connections page to select attributes to copy and map into the operator.

To complete the Input connections page for an operator:

1. Select complete groups or individual attributes from the left panel.

   To search for a specific attribute or group by name, type the text in **Search for** and click **Go.** To find the next match, click **Go** again.

   Hold the Shift key down to select multiple groups or attributes. If you want to select attributes from different groups, you must first combine the groups with a Joiner or Set operator.

2. Use the left to right arrow button in the middle of the page to move your selections to the right side of the wizard page.

   You can use the right to left arrow to move groups or attributes from the input connections list. Warehouse Builder removes the selection from the input group and removes the mapping line between the source operator and the current operator.

Input Attributes

Use the Input page to modify the attributes you selected in the Input Connections tab or wizard page.

You can perform the following tasks from the Match-Merge Input Attributes page:

- **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.
- **Add attributes:** Use the Add button to add input attributes that you did not previously select in the Input Connections tab or wizard page.

Merge Output

Use the Merge Output tab to specify the attributes for the output MERGE group. The MERGE group produces a consolidated record with the attributes you select.

Cross Reference Output

Use the Cross Reference Output page to optionally select attributes for the XREF group. Although the Match-Merge operator creates the XREF group by default, you have the option of adding attributes to the group or leaving it empty.

The XREF group is an optional group you can define to document the merge process. It enables you to create a foreign key relationship between the original data set to the new merged data set. You can send the attributes from the XREF group to a table that records the corresponding source row for each merged row.

Every row from INGRP1 corresponds to a row in the XREF group. To design the XREF group, select original attribute values and merged attributes from **Source Attributes** on the left. Warehouse Builder displays the merged attributes in **Source Attributes** with a default prefix of “MM_”. Use **Set Prefix** at the bottom left corner of the page to change the prefix.

Match Bins

Use the Match Bins page to limit the set of possible matches to a manageable number. When Warehouse Builder matches the rows, it compares each row with the subsequent row for all rows within the same grouping. This can greatly enhance performance since Warehouse Builder searches for matches only within groupings and not throughout the entire data set.

While you want to define Match Bins that separate records into manageable groupings, you also want to avoid separating records that should be matched. The attribute(s) you select for grouping like data depends on your data. For example, if you have a table of customer address with a million rows, you may want to group the data by partial street name, city name, and zip code.

Ideally, you should attempt to keep the number of records in each grouping under 2000. The number of comparisons Warehouse Builder must perform is based on the following formula:

```
n=(b*(b-1))/2
```

where n is number of comparisons and b is number of records in a bin.

To match 5 records, Warehouse Builder must perform 10 comparisons. To match 50 records, Warehouse Builder must perform 1,225 comparisons. To match 500 records, Warehouse Builder must perform 124,750 comparisons.

Match Rules

You can define match rules for a single attribute or multiple attributes in the operator. On the Match Rules tab, create match rules at the top of the page. In the lower portion of Match Rules tab, specify the details for each match rule.

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. This is indicated on the Match Rules tab by the OR icon in the left most column. For more information, see "Understanding Matching Concepts" on page 8-44.

Warehouse Builder assigns a position number and creates a default name such as MM_MA_0 for each match rule you add. You can edit and drag the rule to a new position in the rule list. You can designate a match rule as either Active Match Rules or Passive Match Rules. Warehouse Builder does not execute a passive rule unless you call it through a custom match rule. Assign one of the rule types listed in Table 8–7 on page 8-44. When you select a rule type, the lower portion of the Match Rules tab activates and you can enter details for the match rule.

When you add details for conditional rule types, Warehouse Builder prompts you to add details for one or more attributes. When you add details for multiple attributes, Warehouse Builder displays the AND icon in the left most column. This indicates that Warehouse Builder matches rows only when all the condition details are satisfied.

Active Match Rules

Warehouse Builder executes a match rule if you designate as active. If more than one match rule is active, each is evaluated in turn until a match is found or until all rules are evaluated. If a match occurs, the records are considered a match.

Passive Match Rules

Warehouse Builder does not directly execute passive match rules. Instead, you can create custom match rules that are active and call passive match rules. For each match rule that you define, Warehouse Builder creates a corresponding function with the same name as the match rule. Custom rules can call the passive match rule using this function.

Custom Match Rule

Use this editor to create a custom comparison algorithm for the match rule. Select **Edit** to launch the Custom Match Rule Editor. Double click or drag and drop desired functions and parameters from the navigation tree on the left to the Implementation editor on the right. You can write a custom match rule that references active and passive match rules, functions, and parameters such as THIS_ and THAT_ which represent the two records from INGRP1 that are being compared.

You can also validate your custom rule by selecting **Test** and **Validation** from the Custom Match Rule Editor

Merge Rules

Use the Merge Rules tab to select values for the attributes in the merged record.

On the Merge Rules tab, create merge rules at the top of the page. In the lower portion of the Merge Rules tab, specify the details for each merge rule.

Warehouse Builder assigns a position number and creates a default name such as MM_ME_0 for each merge rule you add. Warehouse Builder executes merge rules in the order of their position numbers. You can edit and drag the rule to a new position in the rule list. Assign one of the rule types listed in Table 8–6.

Table 8–6 Merge Rule Types

Merge Rule Type	Select Single or Multiple Attributes	Description
Any	Single	Select an attribute and Warehouse Builder assigns the first non-blank value for that attribute.
Any Record	Multiple	In the lower portion of the tab, select multiple attributes and Warehouse Builder assigns those attributes based on any matched row.
Copy	Single	Select an attribute and Warehouse Builder assigns the value of another merged attribute.
Custom	Single	Select an attribute. Warehouse Builder assigns a value based on a PL/SQL code that you write. See Custom Merge Rule for details.
Custom Record	Multiple	In the lower portion of the tab, select multiple attributes. Warehouse Builder assigns a value based on a PL/SQL code that you write. See Custom Merge Rule for details.
Match Id	Single	Use this rule when you map the XREF group from one Match-Merge operator into the input of another Match-Merge operator. Select a sequence from the lower portion on the tab to generate the Match Id.
Min Max Record	Multiple	In the lower portion of the tab, select multiple attributes. Warehouse Builder assigns the merge attribute value from the match record that contains the relative value of the selecting attribute.
Min Max	Single	Select the first value out of the possible matches based on the order of another attribute.
Rank	Single	Warehouse Builder assigns values to records based on rank expressions you define. Select a record to use to populate a group of attributes. When you select the record, you have access to all attributes of all records in the matched record set.
Rank Record	Multiple	In the lower portion of the tab, select multiple attributes. Warehouse Builder assigns values based on rank expressions you define.
Sequence	Single	In the lower portion of the tab, select the sequence from the sequences defined in the project.

When you select a rule type, the lower portion of the Merge Rules tab activates and you can enter details for the merge rule.

When you define a merge rule, you can define one rule for all the attributes in the merged record or define a rule for each attribute. For instance, if the merged record is a customer record, it may have attributes such as ADDRESS1, ADDRESS2, CITY, STATE, and ZIP. You can write five rules, one for each attribute. Or, you can write one rule such that all five attributes come from the same matched row.

When you write a rule for a single attribute, specify the attribute in the **Attribute** list box at the top of the page. Then, if necessary, specify the details for the rule in the lower portion of the tab.

When you write a rule for multiple attributes, Warehouse Builder disables the **Attribute** list box at the top of the page and you must define the details in the lower portion of the tab.

Custom Merge Rule

Use this editor to create a custom merge rule that assigns a value based on a PL/SQL code that you write and returns the attribute type you previously selected.

Select Edit to launch the Custom Merge Rule Editor. Double click or drag and drop desired functions and attributes from the navigation tree on the left to the Implementation editor on the right. You can write a custom merge rule that references source attributes, other merge attributes, and functions.

You can also validate your custom rule by selecting **Test** and **Validation** from the Custom Merge Rule Editor.

Following is the Example of a Custom Merge Rule for an Attribute:

```
BEGIN
RETURN M_MATCHES(1)."TAXID";
END;
```

Following is the Example of a Custom Merge Rule for a Record:

```
BEGIN
RETURN M_MATCHES(1);
END;
```

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. Table 8–7 lists the match rules you can specify.

Table 8–7 Match Rule Types

Match Rule	Description
Address	Matches records based on postal addresses. Indicate the attribute(s) that compose the address. Assign input roles to each attribute. For details, see "Address Match Rule" on page 8-46.
All Match	Matches all the rows within the Match Bin.
Conditional	Matches rows based on an algorithm you select. Warehouse Builder matches rows only when all the condition details are satisfied. For details, see "Conditional Match Rule" on page 8-48.
Custom	Create a custom comparison algorithm. Select **Edit** to launch the Custom Match Rule Editor. For more information, see "Custom Match Rule".
Firm	Matches records based on business names. Indicate the attribute(s) that compose the firm name. Assign input roles to each attribute. For details, see "Firm Match Rule" on page 8-49.
None Match	Specifies that no rows match within the Match Bin.

Table 8–7 (Cont.) Match Rule Types

Match Rule	Description
Person	Matches records based on peoples names. Indicate the attribute(s) that compose the person name. Assign input roles to each attribute. For details, see "Person Match Rule" on page 8-50.
Weight	Matches rows based on scores that you assign to attributes. Warehouse Builder compares each attribute using a similarity algorithm that returns a score between 0 and 100 to represent the similarity between the rows. For two rows to be considered a match, the total counts must be greater than the overall score you designate. For details, see "Weight Match Rule" on page 8-51.

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 8–8:

Table 8–8 Two Match Rules

Name	Position	Rule Type	Usage	Description
Rule_1	1	Conditional	Active	Match SSN
Rule _2	2	Conditional	Active	Match LastName and PHN

In the lower portion of the tab, assign the details to Rule_1 as described in Table 8–9:

Table 8–9 Details for Rule_1

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 8–10:

Table 8–10 Details for Rule_2

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 8–11:

Table 8–11 Example Data

Row	FirstName	LastName	PHN	SSN
A	John	Doe	650-123-1111	NULL
B	Jonathan	Doe	650-123-1111	555-55-5555
C	John	Dough	650-123-1111	555-55-5555

According to Rule_1, rowsB 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 8–12:

Table 8–12 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 8–13:

Table 8–13 Sample Data

Row	FirstName	LastName	PHN	SSN
A	John	Jones	650-123-1111	NULL
B	Jonathan	James	650-123-1111	555-55-5555
C	John	Jamos	650-123-1111	555-55-5555

Jones matches James with a similarity of 80 and James matches Jamos with a similarity of 80. Jones does not match Jomos because the similarity is 60 which is less than 80. However, because Jones matches James, and James matches Jamos, all three records match.

Address Match Rule

Use the Address match rule to match records based on postal addresses. Matching by address is most effective when you first correct the address data using the Name-Address operator before the Match-Merge operator. The Name-Address operator identifies addresses as existing in a postal matching database and designates the records with the Is Found flag. The Match-Merge operator processes addresses with the Is Found role faster because the data is known to be syntactically correct, legal, and existing.

To define an Address match rule, complete the following steps:

1. On the Match Rules tab, select Address as the Rule Type.

   The Address Attributes tab and Details tab display at the bottom of the page.

2. In the left panel of the Address Attributes tab, select the attribute that represents the primary address and click the left to right arrow button.

3. Right-click **Roles** and designate that attribute as the Primary Address.

   You must perform this step. If you do not assign the Primary Address role, the match rule is ineffective and you cannot access the Details tab.

4. Add other attributes and designate their roles as necessary. If you used the Name-Address operator to cleanse the address data, assign the Is Found role to the appropriate attribute. See Table 8–14 for the types of roles you can assign.

5. Select the Details tab and select the applicable options as listed in Table 8–15.

Table 8–14 describes the Address Roles you can assign for the Address match rule.

Table 8–14 Address Roles

Role	Description
Primary Address	For the match rule to be valid, you must assign this role to one attribute. The primary address can be the street address such as 100 Main Street or post office box such as PO Box 100.
Unit Number	For addresses with matching primary addresses, the operator compares unit numbers such as suite numbers, floor numbers, or apartment numbers. When the unit numbers are blank, the operator considers them a match. If only one unit number is blank, it is not considered a match unless you select the **Match on blank secondary address** option.
PO_Box	The operator compares the post office box number, the number portion of the primary address when it represents a PO Box. When the primary address represents a street address, the PO Box number is blank.
Dual_primary_address	For addresses with matching primary addresses, the operator compares Dual_primary_addresses which is an address that contains both a street address and a post office box.
Dual_unit_number	Assign this role only if also assigning the dual_primary_address role. The operator compares the Dual_unit_number in one record with the Unit_number and Dual_unit_number of another record. Unit numbers are considered a match if one or both are blank.
Dual_PO_Box	Assign this role only if also assigning the dual_primary_address role. The operator compares the Dual_PO_Box in one record with the PO_Box and Dual_PO_Box of another record.
City	Assign this role only if also assigning the State role. The matching behavior for this role depends on whether you previously used the Name-Address operator to cleanse the address data. For uncorrected address data, the operator compares each City. For corrected addresses, the operator only compares Cities when the postal codes do not match. If both City and State match, then the operator compares the address roles. Cities are considered a match if both are blank but not if only one is blank.
State	Assign this role only if also assigning the City role. The matching behavior for this role depends on whether you previously used the Name-Address operator to cleanse the address data. For uncorrected address data, the operator compares each State. For corrected addresses, the operator only compares States when the postal codes do not match. If both City and State match, then the operator compares the address roles. States are considered a match if both are blank but not if only one is blank.
Postal_code	The matching behavior for this role depends on whether you previously used the Name-Address operator to cleanse the address data. For uncorrected address data, the operator does not use the Postal_code. For corrected addresses, the operator only compares each Postal_code. If the Postal_codes match, then the operator compares the address roles. If the Postal_codes do not match, then the operator compares City and State to determine if it should compare address roles such as the Primary_address.
Is_found	Assign this role for address data that you previously cleansed and standardized with Name-Address operator. The Name-Address operator marks records with the Is_found flag when it identifies the address as part of a country postal matching database.

Table 8–15 describes the options you can assign to Address Roles.

Table 8–15 Options for Address Roles

Option	Description
Allow differing secondary address	Addresses match despite different unit numbers.
Match on blank secondary address	Addresses match despite one blank unit number.
Match on either street or post office box	The operator matches records if either the street address or post office box match.
Address line similarity	The operator ignores all spaces and non-alphanumeric characters in the address lines and calculates the similarity. Records with a similarity greater than or equal to the score you assign are considered a match.
Last line similarity	The operator ignores all spaces and non-alphanumeric characters in City and State and calculates the similarity. Records with a similarity greater than or equal to the score you assign are considered a match.

Conditional Match Rule

Use the Conditional Match Rule to combine multiple attribute comparisons into one composite rule. When you assign multiple attributes for comparison, all the comparisons must be true for the records to be considered a match.

To define a Conditional match rule, complete the following steps:

1. On the top portion of the Match Rules tab, select Conditional for the rule type.

   The operator displays a Details section at the bottom of the tab.
2. Click **Add** to add and select an attribute.
3. For **Algorithm,** select one of the options as listed in Table 8–16.
4. If you select Similarity or Standardized Similarity, specify a similarity score.
5. In **Blank Matching,** specify how the operator should handle blank values.

Table 8–16 Algorithms for Match Rules

Algorithm	Description
Exact	The 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	The operator standardizes the values of the attribute before comparing for an exact match. With standardization, the comparison ignores case, spaces, and non-alphanumeric characters. Using this algorithm, "Dog" and "dog!" do match.
Similarity	Enter a similarity score between 0-100. If the similarity of the two attributes is equal or greater to the score, then the attribute values are considered matched. The similarity algorithm computes the edit distance between two strings. Edit distance is the number of deletions, insertions, or substitutions required to transform one string into another. A similarity value of 100 indicates that the two values are identical. A value similarity 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.

Table 8–16 (Cont.) Algorithms for Match Rules

Algorithm	Description
Standardized Similarity	The operator 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.
Soundex	The operator converts the data to a Soundex representation and compares the text strings. If the Soundex representations match, then the two attribute values match.
Partial Name	The values of a string attribute are considered a match if the value of one 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. Before attempting to match a partial name, this algorithm performs a Standardized Exact comparison on the entire string.
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, the operator performs a Standardized Exact comparison on the entire string. The comparison ignores case and non-alphanumeric characters. For each word, the operator looks for abbreviations, as follows. If the larger of the words contains all of the letters from the shorter word and the letters appear in the same order as the shorter word, then the words are considered a match. For example, "Intl. Business Products" 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, the operator performs a Standardized 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.

Firm Match Rule

Use the Firm match rule to match records by business name. Matching by business name is most effective when you first correct the address data using the Name-Address operator before the Match-Merge operator.

To define a Firm match rule, complete the following steps:

1. On the Match Rules tab, select Firm as the Rule Type.

   The Firm Attributes tab and Details tab display at the bottom of the page.

2. In the left panel of the Firm Attributes tab, select the attribute that represents the firm name and click the left to right arrow button.

3. Right-click **Roles** and designate that attribute as Firm 1.

   By default, the operator compares the values in Firm 1 for exact matches. You can change this default behavior by making selections on the Details tab.

4. Add another attributes and designate it role as Firm 2, if necessary.

   For the match rule to be valid, you must assign at least one attribute either as Firm 1 or Firm 2.

5. Select the Details tab and select the applicable options.

   If you select **Strip noise words,** the operator ignores words in the business names such as “the” and “and”. If you select **Cross match firm 1 and firm 2,** the operator compares business names in firm 1 with business names in firm 2.

   See Table 8–16 on page 8-48 for descriptions of the remaining options on the Details tab.

Person Match Rule

Use the Person match rule to match records based on names. Matching by names is most effective when you first correct the address data using the Name-Address operator before the Match-Merge operator.

To define a Person match rule, complete the following steps:

1. On the Match Rules tab, select Name as the Rule Type.

   The Person Attributes tab and Details tab display at the bottom of the page.

2. In the left panel of the Person Attributes tab, select the attribute that represents the last name and click the left to right arrow button.
3. Right-click **Roles** and designate that attribute as the Last Name.
4. Add other attributes and designate their roles as necessary. You must define either the Last Name or First Name Standardized for the match rule to be effective. See Table 8–17 for the types of roles you can assign.
5. Select the Details tab and select the applicable options as listed in Table 8–18.

Table 8–17 describes the Name Roles you can assign for the Name match rule.

Table 8–17 Name Roles

Role	Description
Prename	The operator compares prenames only if First Name Standardized is blank for one of the records, the “Mrs.” option is selected, and the Last Name and any Middle Name role match. Given that criteria, the operator would match, for example, the record “Mrs. William Webster” with “Mrs. Webster”.
First Name Standardized	First names match if both are blank. A blank first name will not match a non-blank first name unless the Prename role has been assigned and the "Mrs. Match" option is set.
Middle Name Standardized, Middle Name 2 Standardized, Middle Name 3 Standardized	The operator compares and cross compares any of the middle names assigned. By default, the middle names must match exactly. Middle names match if either or both are blank. To assign any of the middle name roles, you must also assign the First Name Standardized role.
Last Name	The operator assigns last names as matching if both are blank and not matching if only one is blank.
Maturity Post Name	This is the same as post names such as “Jr.” and “Sr.”. The operator assigns these as matching if the values are exact or if either is blank.

Table 8–18 lists the options you can select from the Details tab in the Name match rule.

Table 8–18 Options for the Name Role

Option	Description
Match on initials	The operator matches initials to names such as "R." and "Robert". You can select this option for first name and middle name roles.
Match on substrings	The operator matches substrings to names such as "Rob" to "Robert". You can select this option for first name and middle name roles.
Similarity	See Table 8–16.
Soundex	See Table 8–16.
Detect compound name	The operator matches compound names to names such as "De Anne" to "Deanne". You can select this option for the first name role.
"Mrs" Match	The operator matches prenames to first and last names such as 'Mrs. Washington" to "George Washington". You can select this option for the prename role.
Match hyphenated names	The operator matches hyphenated last names to unhyphenated last 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.
Detect switched name order	The operator 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.

Weight Match Rule

Use this rule to match rows based on a weight value that you assign. A weighted match rule is most useful when comparing a large number of attributes, without having a single attribute that is different causing a non-match, as can happen with conditional rules.

To use the Weight match rule, complete the following steps:

1. On the Match Rules tab, select Weight as the Rule Type.

   The Details tab display at the bottom of the page.

2. Select **Add** at the bottom of the page to add an attribute to the rule.

   In **Maximum Score,** assign a weight to each attribute you want to include in the comparison. Warehouse Builder compares each attribute using a similarity algorithm that returns a score between 0 and 100 to represent the similarity between the rows. A value of 100 indicates that the two values are identical. A value of zero indicates there is no similarity.

3. In **Required score to match,** assign an overall score for the match.

   For two rows to be considered a match, the total counts must be greater than the Required score you designate.

Example of a Weight Match Rule

Assume you want to apply the Weight match rule to the data in Table 8–19:

Table 8–19 Example Records for Matching

Record Number	Attr_1	Attr_2
Rec_1	CA	QQ
Rec_2	CA	QQ
Rec_3	CA	QR

For **Maximum score,** you assign a value of 50 to both Att_1 and Att_2. You assign a value of 80 for the **Required score to match.** You can expect the following results:

- Rec_1 is the driver record. The operator reads it first.
- In Rec_2, the value for Attr_1 is CA. That value has a similarity of 100 with the value in the driver record, Rec_1. Since the weight value for Attr_1 is 50, its score is 50 (100% of 50).
- In Rec_2, the value for Attr_2 is QQ and has a similarity of 100. The weight value for Attr_2 is also 50 and its score is therefore 50 (100% of 50). The total maximum score is 100 (50 + 50). This equals or exceeds the value of the **Required score for match** and Rec_2 and Rec_1 are considered matched.
- In Rec_3, Attr_1 is CA and has a similarity of 100 with Rec_1. Since the weight value for Attr_1 is 50, its weighted score is 50 (100% of 50).
- In Rec_3, the value for Attr_2 is QR and that has a similarity of 50. The maximum value for Attr_2 is 50, so its score is 25 (50% of 50). The total weighted score is 75 (50+25). This is less than the value of the **Required score to match.** Therefore, Rec_3 and Rec_1 do not match.

Refining Data from Match-Merge Operators

Once you pass data through a Match-Merge operator, you may want to further refine the data. 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 either a staging table or to another Match-Merge operator. Although mapping to a staging table is relatively easy to design, it can lead to significant loss of performance. Map the XREF group directly to another Match-Merge operator to avoid loss of performance.

Figure 8–19 shows a mapping that relies on a staging table between two Match-Merge operators MM and MM_1.

Figure 8–19 Householding Data: Mapping with Staging Table

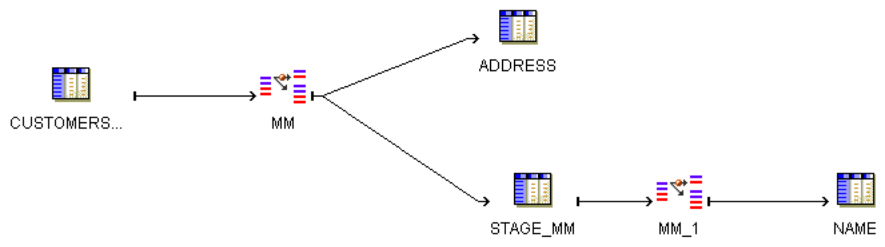
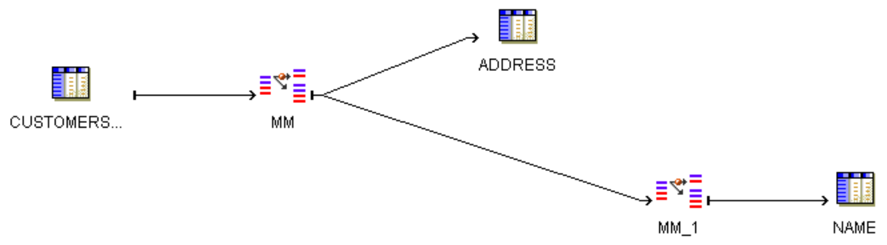


Figure 8–20 shows a mapping that achieves the same results with better performance. The XREF group from MM is mapped directly to MM_1. For this mapping to be valid, you must assign the Match Id generated for the first XREF group as the Match Bin rule on the second Match-Merge operator.

Figure 8–20 Householding Data: XREF Group Mapped to Second Match-Merge Operator



Name and Address Operator

This section includes the following topics:

- About the Name and Address Operator on page 8-53
- Example: Following a Record Through the Name and Address Operator on page 8-54
- Using the Name and Address Operator in a Mapping on page 8-57
- Postal Reporting on page 8-65

This section contains introductory material followed by instructions for using the operator. For advanced information on the Name and Address operator, see Chapter 20, "Data Quality: Name and Address Cleansing".

About the Name and Address Operator

Oracle Warehouse Builder enables you to perform name and address cleansing on your data with 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 the third-party name and address cleansing software vendors. You can purchase the data libraries directly from these vendors.

Note: Taking advantage of the Name and Address operator requires separate licensing and additional installation steps. Refer to the Oracle Warehouse Builder Installation and Configuration 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, or transposed names. The operator fixes these errors and inconsistencies by:

- Parsing, or separating, name and address input data into individual elements.

- Standardizing name and address data, such as standardized versions of nicknames and business names or standard abbreviations of address components, as approved by the Postal Service of the appropriate country. Standardized versions of names and addresses facilitate matching, householding, and ultimately help you obtain a single view of your customer.
- Correcting address information such as street names and city names. Filtering out incorrect or undeliverable addresses can lead to savings on marketing campaigns.
- Augmenting names and addresses with additional data such as gender, ZIP+4, country code, apartment identification, or business and consumer identification. You can use this and other augmented address information, such as census geocoding, for marketing campaigns that are based on geographical location.

  Augmenting addresses with geographic information facilitates geography-specific marketing initiatives, such as marketing only to customers in large metropolitan areas (for example, within an *n*-mile radius from large cities); marketing only to customers served by a company's stores (within 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 in conjunction with this feature.

Finally, the Name and Address operator enables you to generate postal reports for countries that support address correction and postal matching. Using postal reports in countries that support this feature often qualifies you for mailing discounts. For more information, see "Postal Reporting" on page 8-65.

Example: Following a Record Through the Name and Address Operator

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. For more details about how data is processed by the Name and Address operator, see Chapter 20, "Data Quality: Name and Address Cleansing".

Example Input

In this example, your source data contains a Customer table with the row of data shown in Table 8–20.

Table 8–20 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

This 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 for distance calculations for truckload shipping. In order to complete the name and address data, you can use the Name and Address operator.

Example Steps

This example uses a mapping with a Name and Address operator followed by a Splitter operator to cleanse name and address records and then load them into separate targets depending on whether they were successfully parsed. This section explains the general steps required to design such a mapping. For detailed information on each type of operator, refer to that operator's description in this chapter.

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 your data source. It contains the data in "Example Input" on page 8-54.

   A Name and Address operator. This action launches the Name and Address Wizard. Complete it following the instructions in "Using the Name and Address Operator in a Mapping" on page 8-57.

   A Splitter operator. For information on using this operator, see "Splitter Operator" on page 8-76.

   Three target operators to which you load (respectively):

   The successfully parsed records

   The records with parsing errors

   The records whose addresses are parsed but not found in the postal matching software

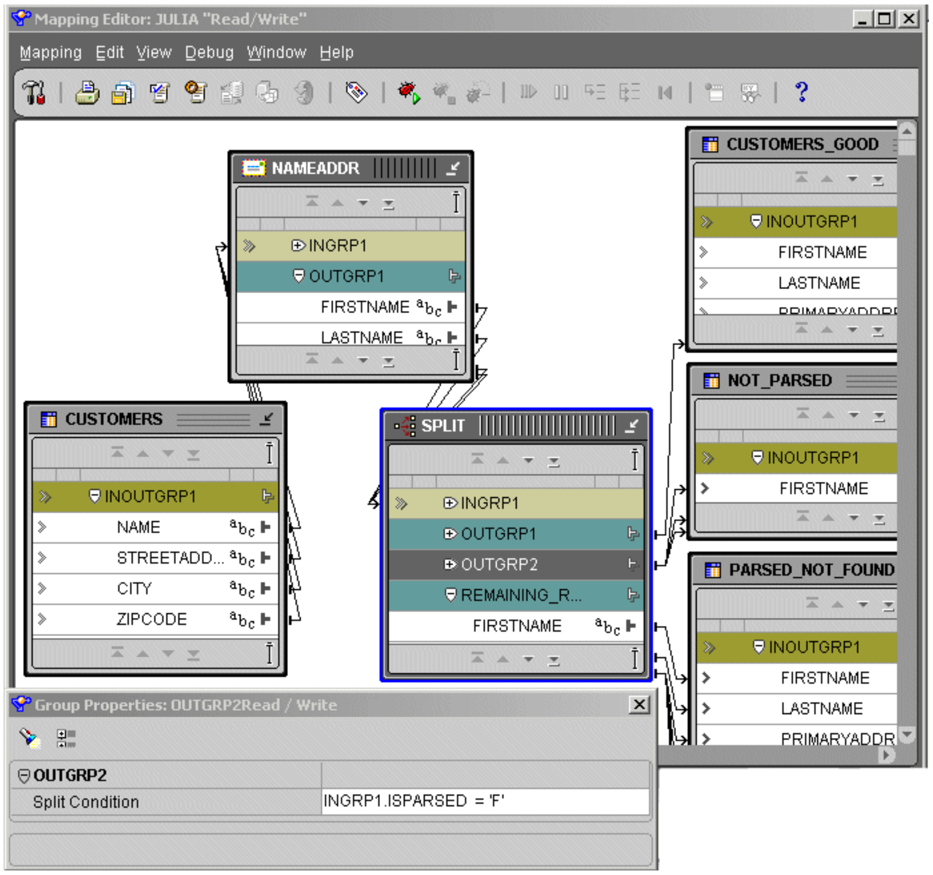
2. Map the attributes from the CUSTOMERS table to the Name and Address operator ingroup. Map the attributes from the Name and Address operator outgroup to the Splitter operator ingroup.

   You are not required to use the Splitter operator in conjunction with the Name and Address operator, but the combination highlights the Name and Address error handling capabilities.

3. Define the split conditions for each of the outgroups in the Splitter operator and map the outgroups to the targets.

Figure 8–21 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 shown at the bottom of the screen: records whose Is Parsed flag is False are loaded to the NOT_PARSED target. 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 8–21 Name and Address Operator Used with a Splitter Operator in a Mapping



OUTGRP2

Split Condition	INGRP1.ISPARSED = 'F'

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 8–21 (compare it with the input record from Table 8–20 on page 8-54).

Table 8–21 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

Table 8–21 (Cont.) Sample Output from Name and Address Operator

Address Column	Address Component
Is Parsed	This field, indicating whether or not the record is parsed successfully, is added for error handling. Based on the value, the record will be loaded into the target operator for successfully parsed records, or the target operator for records with errors.
Is Good Name	This field is added for more detail on error handling.
Name Warning	This field is added to facilitate manual error correction.
Is Good Address	This field is added for more detail on error handling.
Is Found	This field, indicating whether or not the postal matching software found the address, can still be False for a successfully parsed address. It is added here to move those records to a separate target.
Street Warning	This field is added to facilitate manual error correction.
City Warning	This field is added to facilitate manual error correction.

In this example, the following changes were made to the input data:

- Joe Smith was separated into separate columns for a 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.
- Flags were added to distinguish records that parsed successfully from those that did not, and a separate target was loaded with records that had errors.

Using the Name and Address Operator in a Mapping

You have the following options for using the Name and Address operator:

- **Define a new Name and Address operator:** Drag the Name and Address operator from the Toolbox onto the mapping. The Mapping Editor launches a wizard.
- **Edit an existing Name and Address operator:** Right-click the operator and select **Edit.** The Mapping Editor opens the Name and Address Editor.

Whether you are using the operator wizard or the Operator Editor, complete the following pages:

- General
- Definitions
- Groups
- Input Connections
- Input Attributes
- Output Attributes
- Postal Report

General

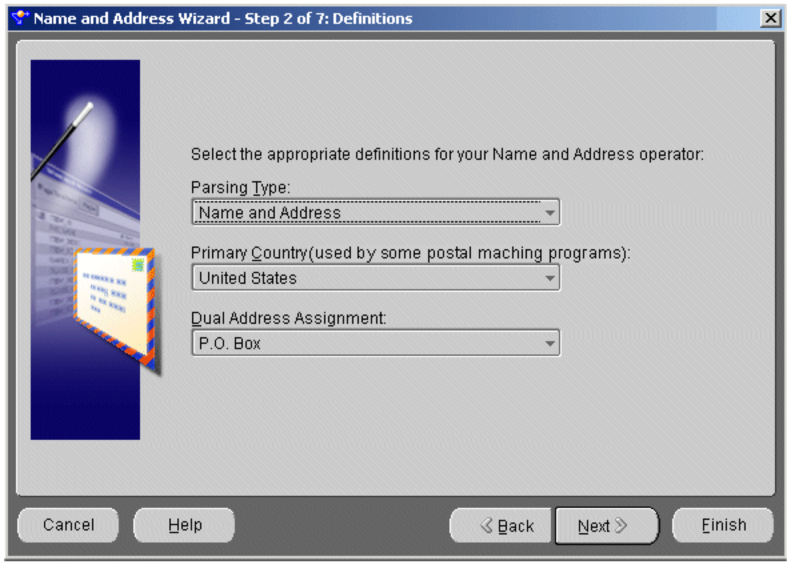
Use the General page to specify a name and optional description for the operator. By default, the wizard names the Name and Address operator "NAMEADDR."

Definitions

Characterize the nature of your input data by assigning general definitions to this Name and Address operator.

Figure 8–22 shows the Definitions page containing sample values for the data described in “"Example: Following a Record Through the Name and Address Operator" on page 8-54. Here, the Parsing Type is set to ‘Name and Address’, Primary Country is set to ‘United States’, and Dual Address Assignment is set to ‘P.O. Box’.

Figure 8–22 Name and Address Operator Definitions Page



Select the appropriate definitions for your Name and Address operator:

Parsing Type:
Name and Address

Primary Country(used by some postal maching programs):
United States

Dual Address Assignment:
P.O. Box

Parsing Type Select one of the following parsing types from the drop-down list:

Note: You can only specify the Parsing Type when you first add the Name and Address operator to your mapping. When you edit a Name and Address operator in the operator editor, you cannot modify the Parsing Type.

- **Name Only:** Select this 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 increases performance and accuracy.
- **Address Only:** Select this when the input data contains only address data and no name data. Selecting this option instead of the more generic Name and Address option increases performance and accuracy.
- **Name and Address:** Select this when the input data contains both name and address data. If your input data only contains one or the other, selecting one of the other options for optimal performance and accuracy.

Primary Country Select the country which 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 your 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 considered the *normal address* and the PO Box address is considered the *dual address*. This means that the address component is assigned the street address. In the preceding example, it is assigned 4439 MORMON COULEE RD. This will cause the postal code to be corrected to 54601-8220.
- **PO Box Assignment:** The PO Box address is considered the *normal address* and the street address is considered the *dual address*. This means that the address component is assigned the Post Office (PO) box address. In the preceding example, it is assigned PO BOX 2589. This will cause the postal code to be corrected to 54602-2589.
- **Closest to Last Line:** Whichever address occurs closest to the last line is considered the *normal address*; the other is considered the *dual address*. This means that the address component is assigned the address line closest to the last line. In the preceding example, it is assigned the street address, 4439 MORMON COULEE RD. This will cause the postal code to be corrected to 54601-8220.

This option has no effect for records having a single street or PO box address. Note that this option may not be supported by all name and address cleansing software providers.

Groups

By definition, the Name and Address operator has one input group and one output group. You cannot edit, add, or delete groups in the Name and Address operator. The input group is called INGRP1 and the output group is OUTGRP1. You can edit these names. If your input data requires multiple groups, create a separate Name and Address operator for each group.

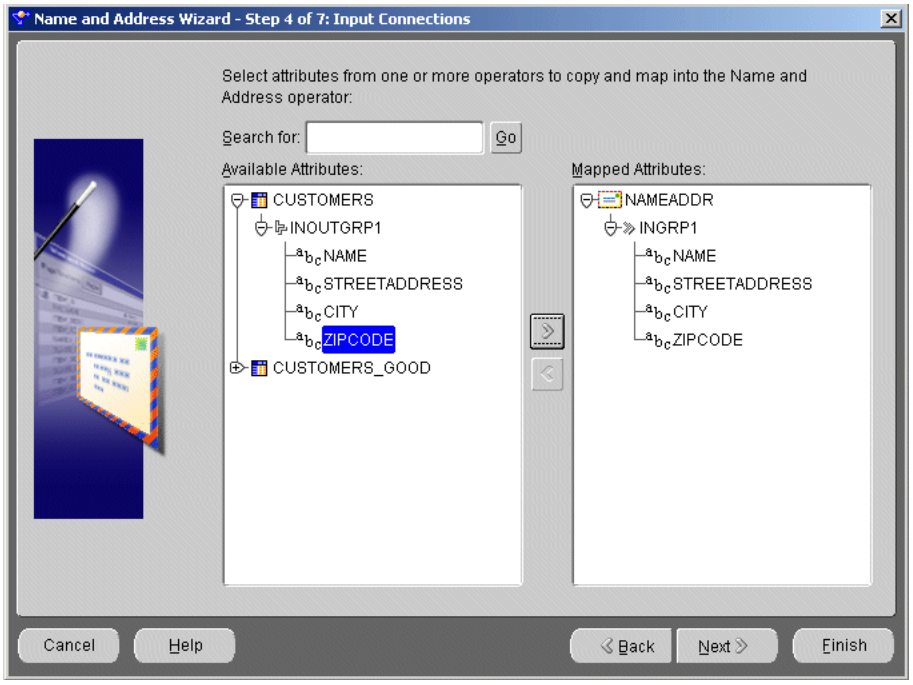
You assign attributes to the INGROUP on the Input Connections page and then edit those attributes on the Input Attributes page. You assign attributes to the OUTGRP1 group on the Output Attributes page.

Input Connections

Use the Input Connections page to select attributes from any operator in your mapping that you want to copy and map into the operator. The Available Attributes panel enables you to select attributes from any operator in your mapping. The Mapped Attributes panel represents the Name and Address operator. When you move attributes from the left panel to the right, you map them to the operator.

Figure 8–23 shows the Input Connections page containing sample values for the example described in "Example: Following a Record Through the Name and Address Operator" on page 8-54. Notice that the CUSTOMERS table columns are mapped as the input attributes.

Figure 8–23 Name and Address Operator Input Connections Page



To complete the Input Connections page for an operator:

1. Select complete groups or individual attributes from the Available Attributes panel. The Available Attributes panel enables you to select attributes from any operator in your mapping.

   To search for a specific attribute or group by name, type the text in **Search for** and click **Go.** To find the next match, click **Go** again.

   Hold the **Shift** key down to select multiple groups or attributes. If you want to select attributes from different groups, you must first combine the groups with a Joiner or Set operator.

2. Use the left to right arrow button between the two panels to move your selections to the Mapped Attributes panel.

Input Attributes

Use the Input Attributes page to further modify the attributes you selected in the Input Connections page, and to assign input roles to each input attribute.

You can perform the following tasks from the Name and Address Input Attributes page:

- **Add attributes:** Use the **Add** button to add input attributes.
- **Change attribute properties:** You can change the following properties: attribute name, input role, length. Leave data type as VARCHAR2. You cannot change Precision or Scale. You must assign an Input Role for every input attribute.

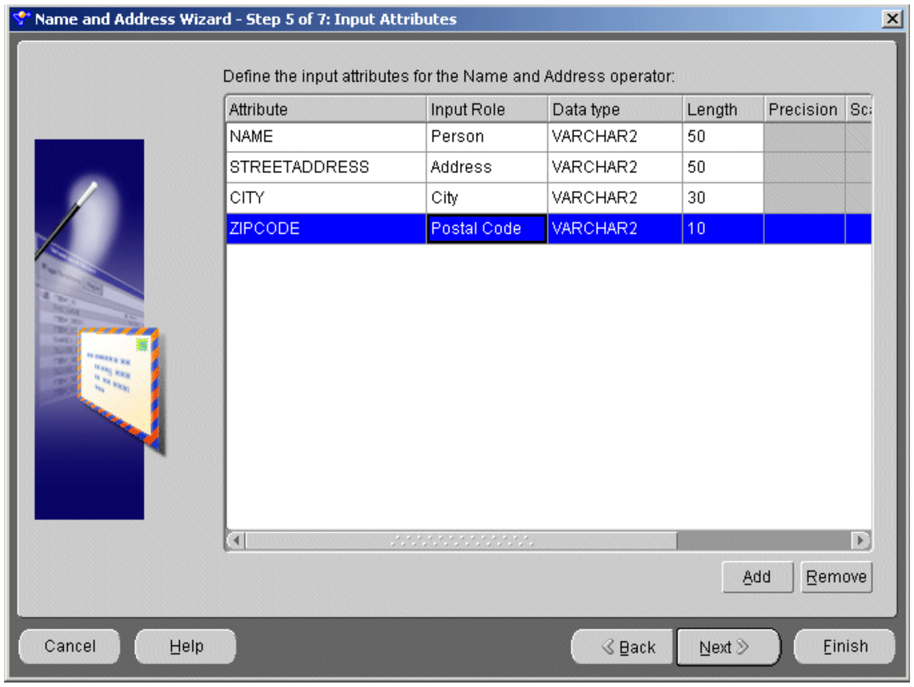
  Input roles indicate what kind of name or address information resides in a line of data. For each attribute, select the Input Role that most closely matches the data contained in the source attribute. Refer to Table 20–1 on page 20-1 for a complete list of input roles and their descriptions.

You can select either non-discrete (line oriented) input roles for free-form data, or discrete roles (such as first name, primary address, or city) for specific input attributes. Whenever possible, choose discrete input roles (such as 'Person'), rather than non-discrete roles (such as 'Line1'). Discrete roles give the Name and Address operator more information about the content of the source attribute.

- **Add an optional description:** Type a description for the input attributes.

Figure 8–24 shows the Input Attributes page containing sample values for the data described in "Example: Following a Record Through the Name and Address Operator" on page 8-54. In this example, the source table has only one NAME column containing both the first and last names, such as Joe Smith, so this column is assigned the 'Person' input role. In the example, the source table also contains the entire street address portion of the mailing address in the STREETADDRESS column, so that column is assigned the 'Address' input role.

Figure 8–24 Name and Address Operator Input Attributes Page



Define the input attributes for the Name and Address operator:

Attribute	Input Role	Data type	Length	Precision	Sc
NAME	Person	VARCHAR2	50		
STREETADDRESS	Address	VARCHAR2	50		
CITY	City	VARCHAR2	30		
ZIPCODE	Postal Code	VARCHAR2	10		

Output Attributes

Use the Output Attributes page to define output attributes that determine how the Name and Address operator handles parsed data. Specifically, the output attribute properties characterize the data that is extracted from the parser output.

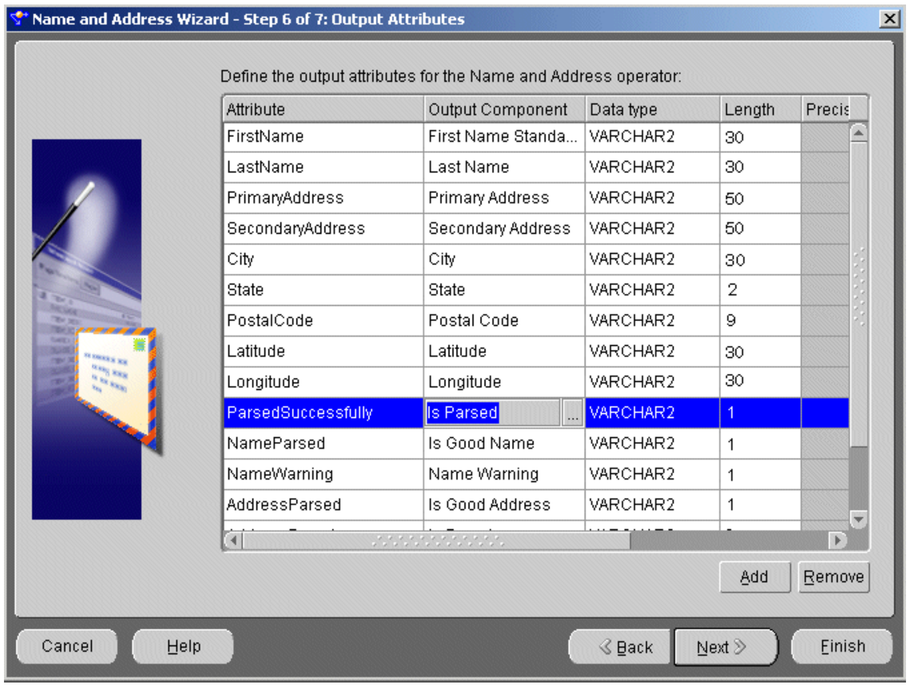
The output attribute collection is initially empty. You can create and edit attributes.

- To create output attributes, select **Add.**
- To edit an attribute name, click the appropriate cell and overwrite the default name.
- You must specify an output component for every output attribute. To do this, click the **...** button to the right of each attribute to open the Output Components dialog, where you can assign a component. See Table 20–2 on page 20-3 for a complete list of the output components and descriptions of their functions.

- You can also adjust the field length to match the length of the target attribute to which you intend to map the output attribute. Adjusting the length to match the target attribute helps avoid data truncation warnings during code generation, or errors during execution.
- You cannot change the data type.

Figure 8–25 shows the Output Attributes page containing sample values for the data described in "Example: Following a Record Through the Name and Address Operator" on page 8-54.

Figure 8–25 Name and Address Operator Output Attributes Page



Define the output attributes for the Name and Address operator:

Attribute	Output Component	Data type	Length	Precis
FirstName	First Name Standa...	VARCHAR2	30	
LastName	Last Name	VARCHAR2	30	
PrimaryAddress	Primary Address	VARCHAR2	50	
SecondaryAddress	Secondary Address	VARCHAR2	50	
City	City	VARCHAR2	30	
State	State	VARCHAR2	2	
PostalCode	Postal Code	VARCHAR2	9	
Latitude	Latitude	VARCHAR2	30	
Longitude	Longitude	VARCHAR2	30	
ParsedSuccessfully	Is Parsed	VARCHAR2	1	
NameParsed	Is Good Name	VARCHAR2	1	
NameWarning	Name Warning	VARCHAR2	1	
AddressParsed	Is Good Address	VARCHAR2	1	

Notice that every output attribute is assigned an output component. For example, the FirstName attribute is assigned the 'First Name Standardized' component. Notice also that 'Latitude' and 'Longitude' attributes are added to augment the address information.

Finally, notice that several error handling flags are added, such as Is Parsed, Is Good Name, and Is Good Address. These flags can be used with the Splitter operator to separate successfully parsed records from records with errors and load them into different targets.

Output Components Use the Output Attributes components dialog to define components for the output attributes you create. Each output component represents a discrete name or address entity, such as a title, a standardized first name, a street number, a street name, or a street type, and indicates which component of a name or address an attribute constitutes. The component you select is assigned to the output attribute.

- **Output Component:** Use the component tree on the left side of the Output Components dialog to expand the category housing your component, and then select the name or address component you want to assign to the attribute in question. You can select any node on the tree that is denoted by the envelope icon with a green border, even if that component expands to reveal other nodes. See

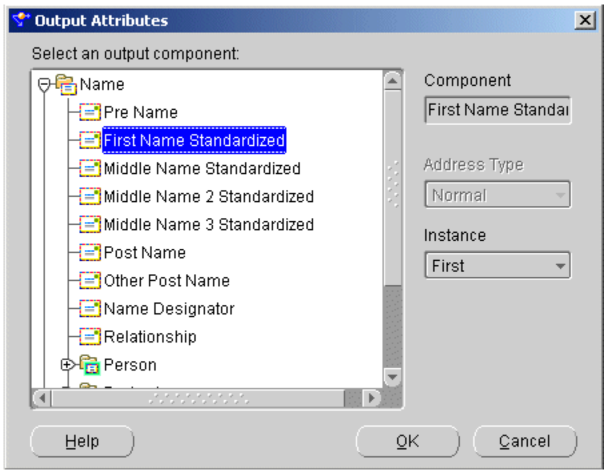
Table 20–2, " Name and Address Operator Output Components" on page 20-3 for a description of these components.

- **Address Type:** Like Dual Address Assignment, this option may not be supported by all name and address cleansing software providers.This option only applies for dual addresses—addresses containing both a street address and a Post Office (PO) box or route-box address. The Dual Address Assignment option you specified in the Definitions page determines which address—the street address or the PO box address—is used as the dual address. Select either the normal or dual address. For more information on dual addresses, see "Dual Address Assignment" on page 8-59.
- **Instance:** Specify which instance of an output component to use when there are multiple occurrences of the same attribute in a single record. The instance control applies to all name components and several address components, such as Miscellaneous Address and Complex. Instance enables you to extract numerous attributes of the same nature.

  For example, an input record containing John and Jane Doe would have two name occurrences: John Doe and Jane Doe. You can extract the first person with any name component by assigning Instance 1 to that component. Similarly, you can also extract the second person using any name component by assigning Instance 2. The number of instances allowed for various components depends on the vendor of the name and address cleansing software you use. Miscellaneous address may also have multiple instances, for example, if both an email address and phone number are present.

Figure 8–26 shows the Output Components page selecting the component for the first sample output attribute used in "Example: Following a Record Through the Name and Address Operator" on page 8-54.

Figure 8–26 Name and Address Operator Output Attributes Components Dialog


Output Attributes

Select an output component:

- Name
  - Pre Name
  - First Name Standardized
  - Middle Name Standardized
  - Middle Name 2 Standardized
  - Middle Name 3 Standardized
  - Post Name
  - Other Post Name
  - Name Designator
  - Relationship
  - Person

Component: First Name Standar

Address Type: Normal

Instance: First

In "Example: Following a Record Through the Name and Address Operator" on page 8-54, the address type is Normal. The example calls for the following output components: First Name Standardized, Last Name, Primary Address, Secondary Address, City, State, Postal Code, Latitude, Longitude, Is Parsed, Is Good Name, Name Warning, Is Good Address, Is Found, Street Warning, and City Warning.

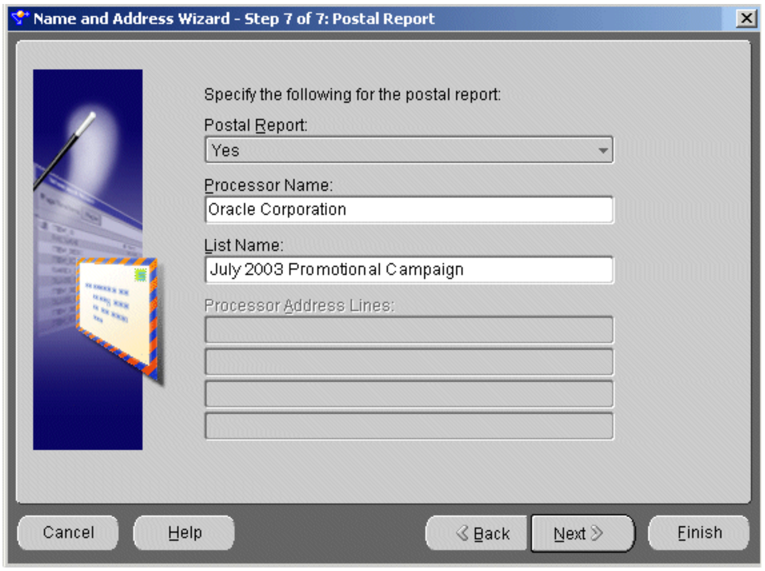
Postal Report

Postal reporting applies only to countries that support address correction and postal matching. Country certification varies with different vendors of name and address cleansing software. The most common country certifications are United States, Canada, and Australia. The process provides mailers a common platform to measure the quality of address-matching software, focusing on the accuracy of postal codes (in the case of the United States, of five-digit ZIP Codes and ZIP+4 Codes), delivery point codes, and carrier route codes applied to all mail. Some vendors of name and address cleansing software may ignore these parameters and require external setup for generating postal reports. For more information, see "Postal Reporting" on page 8-65.

- **Postal Report:** If you select Yes for Postal Report, the Primary Country you chose in the Definitions page determines the country for which the postal report is generated. Only one postal report can be active.
- **Processor Name:** The use of this field varies with vendors of name and address cleansing software. Typically, the value supplied here appears on the United States CASS report.
- **List Name:** The list name is an optional reference field that appears on the United States and United Kingdom reports under the List Name section, but is not output on other reports. The list name provides a reference for tracking multiple postal reports; for example, 'July 2003 Promotional Campaign'.
- **Processor Address Lines:** The four address lines may appear on various postal reports. Various name and address cleansing software vendors use these fields differently. These lines often contain the full address of your company.

Figure 8–27 shows the Postal Report page containing sample values.

Figure 8–27 Name and Address Operator Output Attributes Page



Name and Address Wizard - Step 7 of 7: Postal Report

Specify the following for the postal report:

Postal Report: Yes

Processor Name: Oracle Corporation

List Name: July 2003 Promotional Campaign

Processor Address Lines:

To exit the Name and Address Wizard, click **Finish**. To exit the Name and Address Editor, click **OK**.

Postal Reporting

You can specify a postal report using the Postal Report page in the Name and Address operator editor or wizard. The postal report is generated when the mapping containing the Name and Address operator is executed.

Postal reporting applies to only those countries that support address correction and postal matching. Those countries vary among vendors of name and address cleansing software. The most common certifications are for 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.

All address lists used to produce mailings for automation rates must be matched by postal report-certified software. Oracle Warehouse Builder Name and Address is built on name and address software and data supplied by third-party software vendors specializing in name and address cleansing. Certifications therefore depend on the vendor, and may include the following:

- **United States:** Coding Accuracy Support System (CASS) certification with the United States Postal Service. The CASS report is a text file specified by the USPS and produced by Oracle Warehouse Builder Name and Address. To meet USPS requirements, the mailer must submit a CASS report in its original form to the USPS.
- **Canada:** Software Evaluation and Recognition Program (SERP) certification with Canada Post. Customers who utilize Incentive Lettermail, Addressed Admail, or Publications Mail must meet the Address Accuracy Program requirements. Customers can obtain a Statement of Accuracy by comparing their database to Canada Post's address data.
- **Australia:** Address Matching Approval System (AMAS®) certification with Australia Post. PreSort Letter Service prices require that customers use AMAS Approved Software with unique Delivery Point Identifiers (DPIDs) being current against the latest version of the Postal Address File (PAF) .

Accessing the Postal Report Files

To access the postal report, you must have access to the file system where the Name and Address Server resides. The reports are processed by the Name and Address Server and are written to the `owb/bin/admin/reports` folder, located in the Oracle home path specified during the Warehouse Builder Server-Side installation. For more information on installation parameters, see the Oracle Warehouse Builder Installation and Configuration Guide.

For each report, Warehouse Builder creates a unique file name using the Country Code, Group Name, and the date and time the file is created, for example: `p_CAN_TESTGROUP1_20021219_0130.txt`. This naming convention may not apply to all vendors of postal matching software, because in some cases, the file naming is controlled through external configuration of the vendor installation.

Postal Report Restrictions for International Data

A postal report-enabled mapping should only process that country's data, specified as the Primary Country in the Definitions page of the Name and Address wizard. If your source contains international data, and if the source records contain country codes, map the Country Code source column to an input group attribute of the Name and Address operator. Assign the 'Country Code' input role to the attribute.

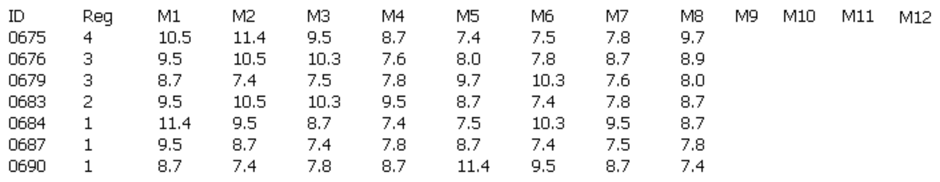
Pivot Operator

The pivot operator enables you to transform a single row of attributes into multiple rows. Use this operator in a mapping when you want to transform data that is contained across attributes instead of rows. This situation can arise when you extract data from non-relational data sources such as data in a crosstab format.

Example: Pivoting Sales Data

The external table `SALES_DAT`, shown in Figure 8–28, contains data from a flat file. There is a row for each sales representative and separate columns for each month. For more information on external tables, see "Using External Tables" on page 3-18.

Figure 8–28 SALES_DAT



ID	Reg	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
0675	4	10.5	11.4	9.5	8.7	7.4	7.5	7.8	9.7				
0676	3	9.5	10.5	10.3	7.6	8.0	7.8	8.7	8.9				
0679	3	8.7	7.4	7.5	7.8	9.7	10.3	7.6	8.0				
0683	2	9.5	10.5	10.3	9.5	8.7	7.4	7.8	8.7				
0684	1	11.4	9.5	8.7	7.4	7.5	10.3	9.5	8.7				
0687	1	9.5	8.7	7.4	7.8	8.7	7.4	7.5	7.8				
0690	1	8.7	7.4	7.8	8.7	11.4	9.5	8.7	7.4				

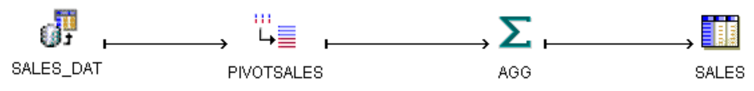
Table 8–22 shows a sample of the data after Warehouse Builder performs a pivot operation. The data that was formerly contained across multiple columns (M1, M2, M3...) is now contained in a single attribute (Monthly_Sales). A single ID row in `SALES_DAT` corresponds to 12 rows in pivoted data.

Table 8–22 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 8–29.

Figure 8–29 Pivot Operator in a Mapping



In this mapping, Warehouse Builder reads the data from the external table once, pivots the data, aggregates the data, and writes it to a target in set based mode. It is not necessary to load the data to a target directly after pivoting it. You can use the pivot operator in a series of operators before and after directing data into the target operator. You can place operators such as filter, joiner, and set operation before the pivot operator. Since pivoted data in Warehouse Builder is not a row-by-row operation, you can also execute the mapping in set based mode.

The Row Locator

In the pivot operator, the row locator is an output attribute that you create to correspond to the repeated set of data from the source. When you use the pivot operator, Warehouse Builder transforms a single input attribute into multiple rows and generates values for a row locator. In this example, since the source contains attributes for each month, you can create an output attribute named ‘`MONTH`’ and designate it as the row locator. Each row from `SALES_DAT` then yields 12 rows of pivoted data in the output.

Table 8–21 shows the data from the first row from `SALES_DAT` after Warehouse Builder pivots the data with ‘`MONTH`’ as the row indicator.

Table 8–23 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

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 Toolbox onto the mapping. The Mapping Editor launches the Pivot Wizard.
- **Edit an existing pivot operator:** Use the Pivot Editor to edit a pivot operator you previously created. Right click the operator and select **Edit.** The Mapping Editor opens the Pivot Editor.

Whether you are using the Pivot operator 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.

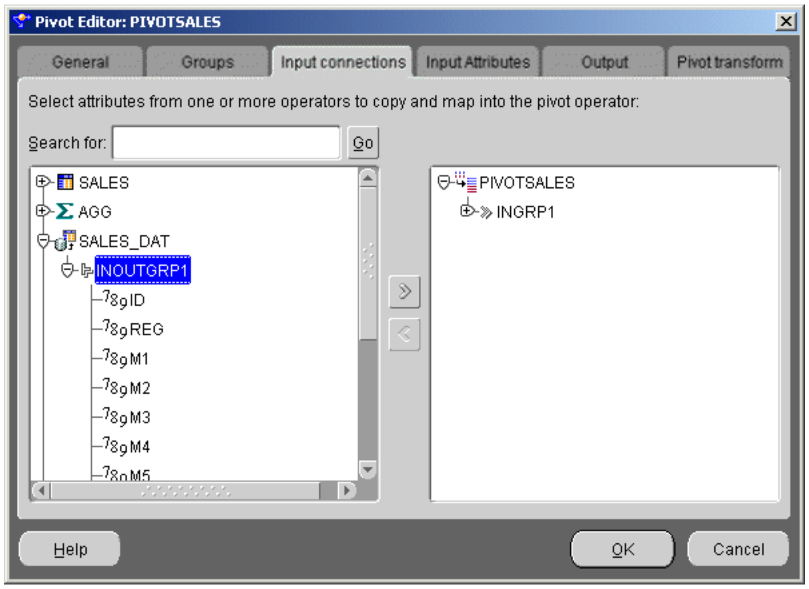
In a pivot operator, the input group represents the data from the source that is contained across multiple attributes. The output group represents that data transformed into rows.

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 8–30 shows a group from the external table SALES_DAT selected as input for the pivot operator.

Figure 8–30 Pivot Operator Input Connections Page


Pivot Editor: PIVOTSALES

General | Groups | Input connections | Input Attributes | Output | Pivot transform

Select attributes from one or more operators to copy and map into the pivot operator:

Search for: [ ] Go

- SALES
- AGG
- SALES_DAT
  - INOUTGRP1
    - ID
    - REG
    - M1
    - M2
    - M3
    - M4
    - M5

- PIVOTSALES
  - INGRP1

Help | OK | Cancel

To complete the Input Connections page for a pivot operator:

1. Select complete groups or individual attributes from the left panel.

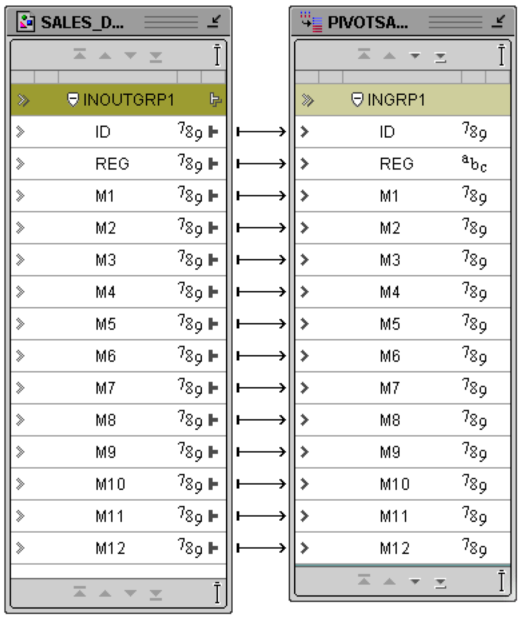
   To search for a specific attribute or group by name, type the text in **Search for** and select **Go.** To find the next match, select **Go** again.

   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 left to right arrow button in the middle of the page to move your selections to the right side of the wizard page.

   Use the right to left arrow to remove groups or attributes from the input connections list. Warehouse Builder removes the selection from the input group and removes the mapping line between the source operator and the pivot operator. Figure 8–31 shows a group from `SALES_DAT` copied and mapped into the `PIVOTSALES` operator.

Figure 8–31 Attributes Copied and Mapped into Pivot In Group



Input Attributes

Use the Input Attributes page to modify the attributes you selected in the Input connections tab or wizard page.

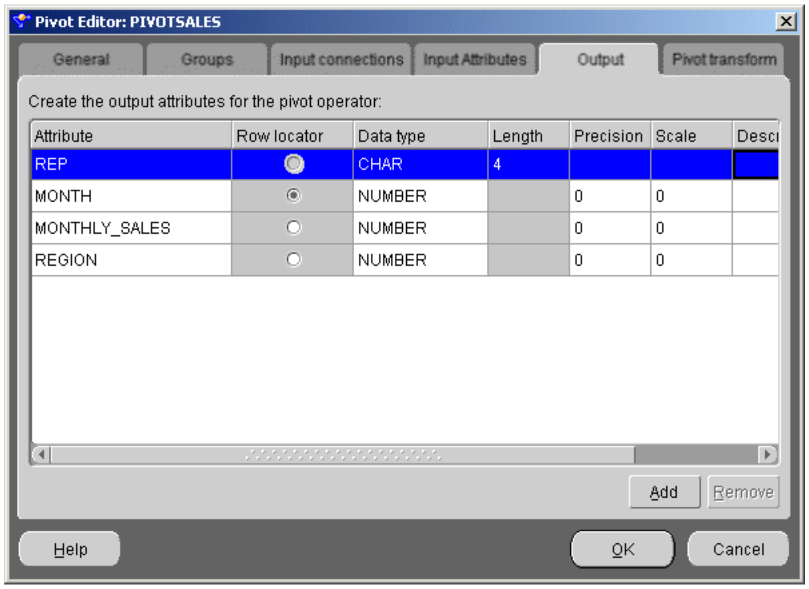
You can perform the following tasks from the pivot Input Attributes page:

- **Add attributes:** Use the Add button to add input attributes.
- **Change attribute properties:** You can change the attribute name, data type, length, precision, and scale.
- **Add an optional description:** Type a description for the input attributes.
- **Designate attribute keys:** As an option, use the **Key** check box to indicate an attribute that uniquely identifies the input group.

Output Attributes

Use the Output Attributes page to create the output attributes for the pivot operator. If you designated any input attributes as keys on the Input Attributes tab or wizard page, Warehouse Builder displays those input attributes as output attributes that you cannot edit or delete. Figure 8–32 displays the output attributes with `MONTH` selected as the row locator.

Figure 8–32 Pivot Output Attributes Page



Create the output attributes for the pivot operator:

Attribute	Row locator	Data type	Length	Precision	Scale	Desc
REP	○	CHAR	4			
MONTH	●	NUMBER		0	0	
MONTHLY_SALES	○	NUMBER		0	0	
REGION	○	NUMBER		0	0	

You can perform the following tasks from the pivot Output Attributes Page:

- **Change attribute properties:** Except for attributes you designated as keys on the previous tab or wizard page, you can change the attribute name, data type, length, precision, and scale.
- **Add an optional description:** Type a description for the output attributes.
- **Designate a row locator:** Although you are not required to designate a row locator for the pivot operator, it is recommended. When you identify the row locator on the Output Attributes page or tab, it is easier for you to match your output data to the input data.

  In the pivot operator, the row locator is an output attribute that corresponds to the repeated set of data from the source. For example, if the source data contains separate attributes for each month, create an output attribute 'MONTH' and designate it as the row locator.

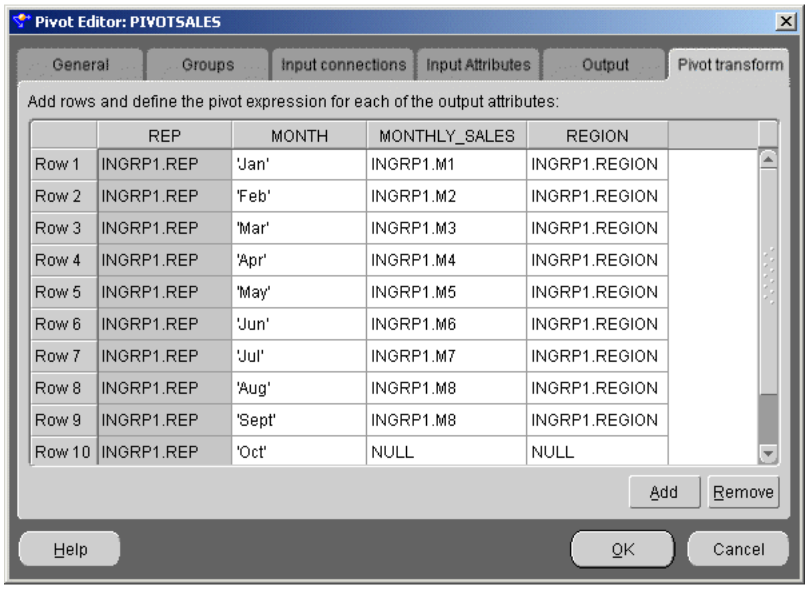
Pivot Transform

Use the Pivot Transform page to write expressions for each output attribute.

By default, Warehouse Builder displays two rows. Use **Add** to specify how many rows of output you want from a single row in the source. For example, if your source contains an attribute for each quarter in a year, you can specify 4 rows of output for each row in the source. If the source data contains an attribute for each month in the year, you can specify 12 rows of output for each row in the source.

Figure 8–33 shows the Pivot Transform tab with the pivot expressions defined for a source with an attribute for each month.

Figure 8–33 Pivot Transform Page



Add rows and define the pivot expression for each of the output attributes:

	REP	MONTH	MONTHLY_SALES	REGION
Row 1	INGRP1.REP	'Jan'	INGRP1.M1	INGRP1.REGION
Row 2	INGRP1.REP	'Feb'	INGRP1.M2	INGRP1.REGION
Row 3	INGRP1.REP	'Mar'	INGRP1.M3	INGRP1.REGION
Row 4	INGRP1.REP	'Apr'	INGRP1.M4	INGRP1.REGION
Row 5	INGRP1.REP	'May'	INGRP1.M5	INGRP1.REGION
Row 6	INGRP1.REP	'Jun'	INGRP1.M6	INGRP1.REGION
Row 7	INGRP1.REP	'Jul'	INGRP1.M7	INGRP1.REGION
Row 8	INGRP1.REP	'Aug'	INGRP1.M8	INGRP1.REGION
Row 9	INGRP1.REP	'Sept'	INGRP1.M8	INGRP1.REGION
Row 10	INGRP1.REP	'Oct'	NULL	NULL

Write pivot expressions based on the following types of output:

- **Row locator:** Specify a name for each row where the name is a value you want to load into the table. For example, if the row locator is 'MONTH', type 'Jan' for the first row.
- **Pivoted output data:** Select the appropriate expression from the list box. For example, for the row you define as 'Jan', select the expression that returns the set of values for January.
- **Attributes previously specified as keys:** Warehouse Builder defines the expression for you.
- **Unnecessary data:** If the Pivot Transform page contains data that you do not want as output, use the expression 'NULL'. Warehouse Builder outputs a repeated set of rows with no data for attributes you define as 'NULL'.

When using the wizard to create a new pivot operator, click **Finish** when you want to close the wizard. The Mapping Editor displays the operator you defined.

When using the Pivot Editor to edit an existing pivot operator, click **OK** when you have finished editing the operator. The Mapping Editor updates the operator with the changes you made.

Post-Mapping Process Operator

Use a Post-Mapping Process operator to define a procedure to be executed after running a mapping. For example, you can use a Post-Mapping Process operator to reenable and build indexes after a mapping completes successfully and loads data into the target.

The Post-Mapping Process operator calls a function or procedure that is defined in Warehouse Builder after the mapping is executed. The output parameter group provides the connection point for the returned value (if implemented through a function) and the output parameters of the function or procedure. There are no restrictions on the connections of these output attributes

The Post-Mapping Process operator contains groups corresponding to the number and direction of the parameters associated with the selected PL/SQL procedure or function. This list of groups and attributes can only be modified through reconciliation.

You can only define one Post-Mapping Process operator for a mapping. If you want to run more than one procedure after a mapping, you must wrap the procedures into one procedure.

You can map constants, data generators, mapping input parameters, and output from a Pre-Mapping Process into a Post-Mapping Process operator. The Post-Mapping Process operator is not valid for an SQL*Loader mapping.

After you add a Post-Mapping Process operator to the Mapping Editor, use the operator properties dialog to specify run conditions in which to execute the process.

To use a Post-Mapping Process operator in a mapping:

1. Drop a Post-Mapping Process operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Mapping Transformation dialog.

2. Use the Add Mapping Transformation dialog to select or create a transformation. For more information on how to use the Add Mapping Transformation dialog, see "Adding Bindable Operators" on page 6-7.
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, right-click a Post-Mapping Process operator and select **Operator Properties.**
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 navigation tree, select **Configure,** and expand **Runtime Parameters.**

Pre-Mapping Process Operator

Use a Pre-Mapping Process operator to define a procedure to be executed before running a mapping. For example, you can use a Pre-Mapping Process operator to truncate tables in a staging area before running a mapping that loads tables to that staging area. You can also use a Pre-Mapping Process operator to disable indexes before running a mapping that loads data to a target. You can then use a Post-Mapping Process operator to reenable and build the indexes after running the mapping that loads data to the target.

The Pre-Mapping Process operator calls a function or procedure whose metadata is defined in Warehouse Builder prior to executing a mapping. The output parameter group provides the connection point for the returned value (if implemented with a function) and the output parameters of the function or procedure. There are no restrictions on the connections of these output attributes.

When you drop a Pre-Mapping Process operator onto the Mapping Editor canvas, a dialog opens displaying the available libraries, categories, functions, and procedures. After you select a function or procedure from the tree, the operator displays with predefined input and output parameters.

The Pre-Mapping Process operator contains groups corresponding to the number and direction of the parameters associated with the selected PL/SQL procedure or function.

A mapping can only contain one Pre-Mapping Process operator. Only constants, mapping input parameters, and output from a Pre-Mapping Process can be mapped into a Post-Mapping Process operator.

After you add a Pre-Mapping Process operator to the Mapping Editor, use the operator property dialog to specify conditions in which to execute the mapping.

To use a Pre-Mapping Process operator in a mapping:

1. Drop a **Pre-Mapping Process** operator onto the Mapping Editor canvas.

   The Add Mapping Transformation dialog displays.

2. Use the Add Mapping Transformation dialog to select or create a transformation. For more information on how to use the Add Mapping Transformation dialog, see "Adding Bindable Operators" on page 6-7.

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, right-click a Pre-Mapping Process operator and select **Operator Properties.**

2. Click **Mapping Run Condition** and select one of the following run conditions:

   Always: Warehouse Builder runs the mapping after the process completes, regardless of the errors.

   On Success: Warehouse Builder runs the mapping only if the process completes without errors.

   On Error: Warehouse Builder runs the mapping only if the process completes with errors.

Set Operation Operator

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. Mapping attributes to a Set Operation input group creates corresponding attributes with the same name and data type in the Set Operation output group. 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.
- 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}.

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. Right-click the operator header and select **Operator Properties...**

   The Set Operation properties window displays.
4. Click the field to the right of the **Set Operation** property and select an operation from the drop-down list.
5. Close the Set Operation properties window.
6. Connect the Set Operation output group to a target input group.

Sorter Operator

You can produce a sorted row set using the Sorter operator. The Sorter operator enables you to specify which input attributes are sorted and whether the sorting is performed in ascending or descending order. Warehouse Builder sorts data by placing an ORDER BY clause in the code generated by the mapping.

The Sorter operator has one input/output group. You can use the Sorter operator to sort data from any relational database source. You can place any operator after the Sorter operator.

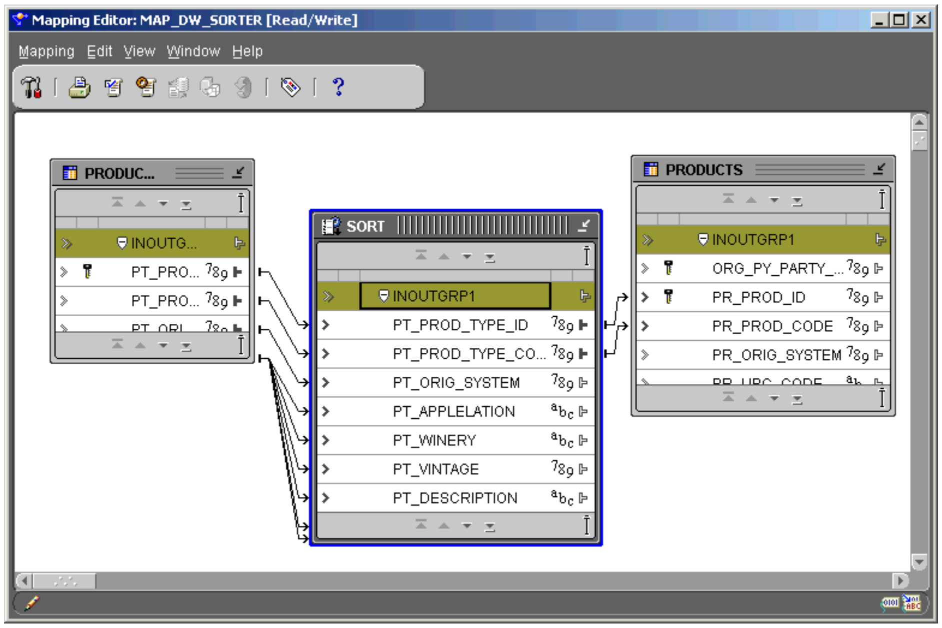
The Sorter operator contains the following property:

- **Order By Clause:** 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.

To use the Sorter operator in a mapping:

1. 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 8–34.

Figure 8–34 Mapping Editor with a Sorter Operator



3. Right-click the Sorter operator header and select **Operator Properties.**

   The Sorter properties window displays.

4. Click the **...** button in the Order By Clause field.

   The Order By Clause dialog displays.

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 drop-down list.

7. Click **OK.**

Splitter Operator

You can use the Splitter operator to split data from one source to several targets. The 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.

You can configure Warehouse Builder mappings that split data from one source to multiple targets to take advantage of Oracle9*i* functionality that optimizes SQL code and improves performance. For more information, see "Example: Creating Mappings with Multiple Targets" on page 8-79.

The Splitter operator creates an output group called REMAINING_ROWS containing all input rows not included in any of the other output groups. You can delete this output group, but you cannot edit it.

The Splitter Operator contains the following properties:

- **Split Condition:** The text expression template for 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.
- **Data Type:** The data type of the attribute.
- **Precision:** The precision of the attribute, used for numeric type attributes only.
- **Scale:** The scale of the attribute, used for numeric type attributes only.
- **Length:** The length of the attributes, used for string-type attributes only.

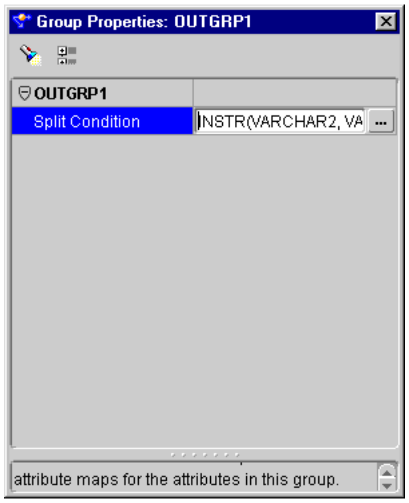
To use the Splitter operator in a mapping:

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. Right-click the Splitter operator header and select **Operator Properties.**

   The Splitter properties window displays as shown in Figure 8–35.

Figure 8–35 Group Properties Window for a Split Condition



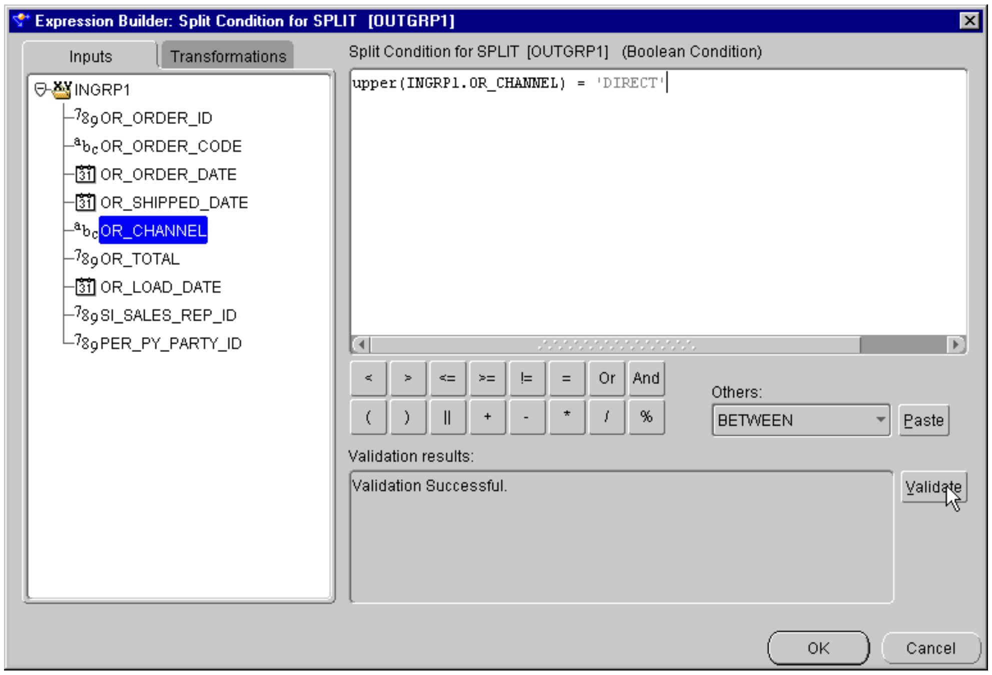
OUTGRP1

Split Condition	INSTR(VARCHAR2, VA

attribute maps for the attributes in this group.

4. Enter an expression in the Split Condition field. Or click **...** to define an expression using the Expression Builder as shown in Figure 8–36.

Figure 8–36 Expression Builder Showing A Split Condition



Split Condition for SPLIT [OUTGRP1] (Boolean Condition)

```
upper(INGRP1.OR_CHANNEL) = 'DIRECT'
```

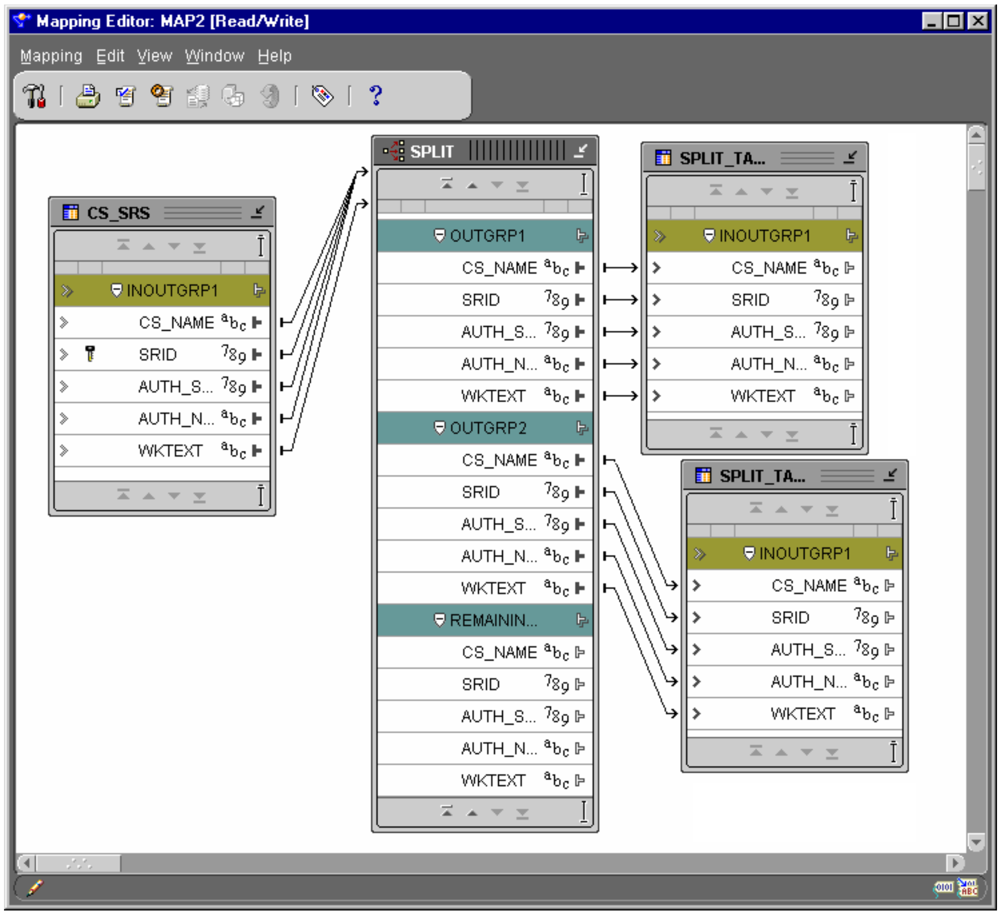
Others: BETWEEN

Validation results:

Validation Successful.

5. Close the Splitter Properties window.
6. Define expressions for each of the output groups except for the REMAINING ROWS group.
7. Connect the output groups to the targets.

Figure 8–37 Mapping with a Single Source and Multiple Targets



Example: Creating Mappings with Multiple Targets

When you design and configure a mapping with the Splitter operator, Warehouse Builder generates a `multi_table_insert` statement. This SQL statement takes advantage of parallel query and parallel DML services in the Oracle9*i* database server.

To create a mapping with multiple targets:

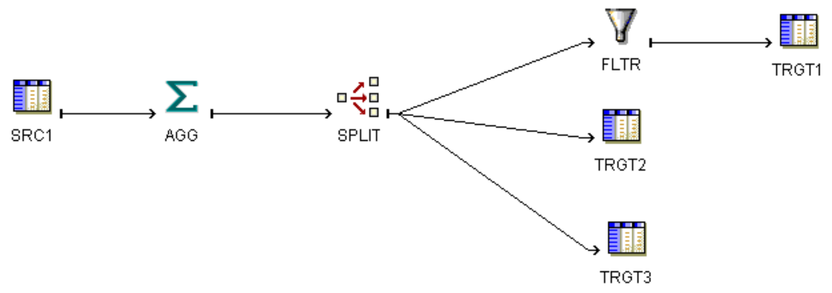
1. Configure an Oracle target module that contains the mapping to validate and generate Oracle9*i* SQL.

   From Warehouse Builder, right-click the target module on the navigation tree and select **Configuration Properties.** Under **Target Database Type,** select Oracle9*i*.

2. In the Mapping Editor, design a mapping with a single source, a Splitter operator, and multiple targets.

   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 8–38, but not a Joiner or Aggregator operator.

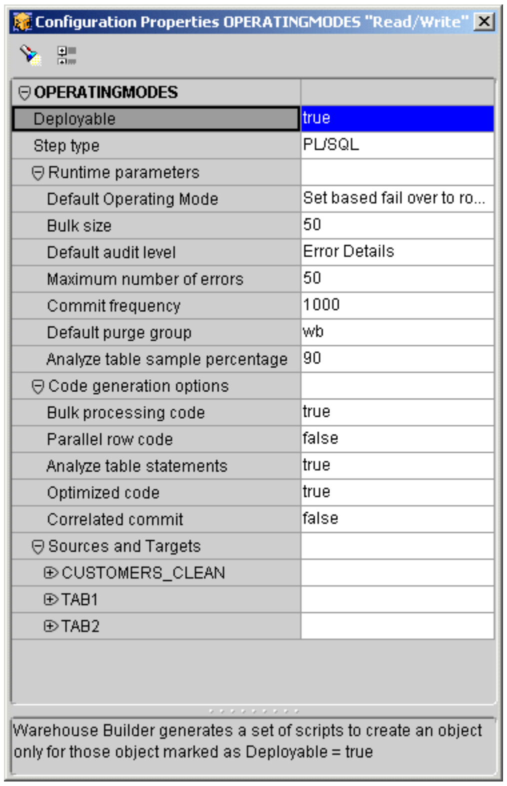
Figure 8–38 Example Mapping with Multiple Targets



3. From the Warehouse Builder console, select the mapping from the navigation tree, select **Object** from the menu bar, and select **Configure.** You can also right-click the mapping you want to configure and select **Configure.**

   Warehouse Builder displays the configuration properties dialog for a mapping as shown in Figure 8–39.

Figure 8–39 Configuration Properties Window for Mapping Tables



OPERATINGMODES	
Deployable	true
Step type	PL/SQL
Runtime parameters	
Default Operating Mode	Set based fail over to ro...
Bulk size	50
Default audit level	Error Details
Maximum number of errors	50
Commit frequency	1000
Default purge group	wb
Analyze table sample percentage	90
Code generation options	
Bulk processing code	true
Parallel row code	false
Analyze table statements	true
Optimized code	true
Correlated commit	false
Sources and Targets	
CUSTOMERS_CLEAN	
TAB1	
TAB2	

Warehouse Builder generates a set of scripts to create an object only for those object marked as Deployable = true

4. Expand **Runtime Parameters Reference** and set **Default Operating Mode** to set based.

5. Expand **Code Generation Options Reference** and set **Optimize Code** to true.

   When you run this mapping and view the generation results, Warehouse Builder returns one total SELECT and INSERT count for all targets.

Table Function Operator

Table function operators enable you to manipulate a set of input rows and return another set of rows possibly of different cardinality. Unlike conventional functions, table functions can return a set of output rows that can be queried like a physical table.

Using table functions can greatly improve performance when loading your data warehouse.

Table Functions have the following characteristics:

- 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 Mapping Table Function operator in a mapping, you need to create the table function in your target, external to Warehouse Builder. The table functions in the database that are supported by the unbound table function operator must meet the following requirements:

Input

- Ref Cursor returning PLS Record (the fields of the PLS Record) must be scalar data types supported by Warehouse Builder (0..n).
- There must be at least one input parameter.

Output

TABLE OF:

- PLS Record (the fields of the PLS Record should be scalar data types supported by Warehouse Builder).
- Object Type (the attributes of the Object Type should be scalar data types supported by Warehouse Builder).
- Scalar data types supported by Warehouse Builder.
- ROWTYPE

For an unbound Mapping 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 mapping 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

The Mapping Table Function operator contains the following properties:

General

General properties include the name and description. Specify the name of the table function located in the target database. The description is optional.

Input Parameter Group

The table function operator accepts the following types of input parameters:

- **Input Parameter Type:** Valid input parameter types are REF_CURSOR_TYPE or SCALAR_TYPE.
- **REF_CURSOR_TYPE:** Returns a PLS Record {0...N}. The fields of the PLS Record must be a scalar data type supported by Warehouse Builder.
- **SCALAR_TYPE:** Scalar data types supported by Warehouse Builder.
- **Parameter Position:** Indicates the position of the parameter in the table function signature corresponding to this parameter group.

Input Parameter

- **Parameter Position:** The position of the parameter in the table function signature. This property is only applicable to scalar parameters.

Output Parameter Group

- **Return Table of Scalar:** This property specifies whether the return of the table function is a TABLE of SCALAR or not. This information is required because the select list item for TABLE of SCALAR must be Select COLUMN_VALUE while in the other cases it should be an appropriate name.

Output Parameter

- **Type Attribute Name:** The name of the field of the PLS Record, attribute of the Object Type, or column of the ROWTYPE. This property is not applicable if the return type is TABLE of SCALAR. This name is used to invoke the table function.

To use a Table Function operator in a mapping:

1. Drag and drop a **Mapping Table Function** operator onto the Mapping Editor canvas.
2. Connect the appropriate source attributes to the input group of the mapping table function operator.

   This automatically creates the input attributes.
3. Right-click the Table Function operator and select **Edit.**
4. From the Groups tab, select **Add** to add an output group.

Before you deploy the mapping containing the mapping table function operator, you must manually create the table function in the target warehouse. The mapping table function operator is bound to the actual table function object through the code generated by the mapping.

Transformation Operator

You use the Mapping Transformation operator to transform the column value data of rows within a row set using a PL/SQL function, while preserving the cardinality of the input row set.

The Mapping Transformation operator must be bound to a function or procedure contained by one of the modules in the repository. The inputs and outputs of the Mapping Transformation operator correspond to the input and output parameters of the bound repository function or procedure. If the Mapping 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 Mapping Transformation operator onto a mapping. In addition, you can choose a function or procedure from the Global Shared Library.

The Mapping Transformation operator contains the following properties:

- **Function Call:** The text template for the function call that is generated by the code generator with the attribute names listed as the calling parameters. For the actual call, the attribute names are replaced with the actual source or target columns that are connected to the attributes.
- **Function Name:** The name of the function or procedure, to which this operator is bound.
- **Procedure:** A boolean value indicating, if true, that the bound transformation is a procedure rather than a function with no returned value.
- **Data Type:** Indicates the data type of the input, output, or result parameter of the bound function that corresponds to the given attribute. If the output of a mapping transformation is of CHAR data type, then Warehouse Builder applies an RTRIM on the result before moving the data to a target. This ensures that no extra spaces are contained in the output result.
- **Default Value:** The default value (blank if none) for the given attribute.
- **Optional Input:** A boolean value indicating, if true, that the given attribute is optional. If the attribute is optional, it need not be connected in the mapping.
- **Function Return:** A boolean value indicating, if true, that the given output attribute is the result attribute for the function. The result attribute is a named result. Use this property if another output is a named result, or if you change the name of the result output.

To use a Mapping Transformation operator in a mapping:

1. Drag and drop a **Mapping Transformation** operator onto the Mapping Editor canvas.

   The Add Mapping Transformation dialog displays.
2. Use the Add Mapping Transformation dialog to create a new transformation or select one or more transformations. Warehouse Builder adds a transformation

operator for each transformation you select. For more information on these options, see "Adding Bindable Operators" beginning on page 6-7.

3. Connect the source attributes to the inputs of the Mapping Transformation operator.
4. (Optional step) Right-click one of the inputs and select **Attribute Properties.**

   The Mapping Transformation properties window displays.
5. Select an input attribute. If the Procedure property is set to True**,** then do not connect the input parameter.
6. Close the attribute property window.
7. Connect the Transformation operator output attributes to the target attributes.

Unpivot Operator

The unpivot operator converts multiple input rows into one output row. The unpivot operator enables you to extract from a source once and produce one row from a set of source rows that are grouped by attributes in the source data. Like the pivot operator, the unpivot operator can be placed anywhere in a mapping.

Example: Unpivoting Sales Data

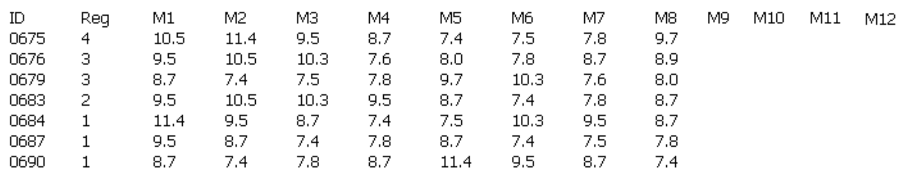
Table 8–24 shows a sample of data from the SALES relational table. In the crosstab format, the ‘MONTH’ column has 12 possible character values, one for each month of the year. All sales figures are contained in one column, ‘MONTHLY_SALES’.

Table 8–24 Data in a Crosstab Format

REP	MONTH	MONTHLY_SALES	REGION
0675	Jan	10.5	4
0676	Jan	9.5	3
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 8–40 depicts data from the relational table ‘SALES’ after Warehouse Builder unpivoted the table. The data formerly contained in the ‘MONTH’ column (Jan, Feb, Mar...) corresponds to12 separate attributes (M1, M2, M3...). The sales figures formerly contained in the ‘MONTHLY_SALES’ are now distributed across the 12 attributes for each month.

Figure 8–40 Data Unpivoted from Crosstab Format



ID	Reg	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
0675	4	10.5	11.4	9.5	8.7	7.4	7.5	7.8	9.7				
0676	3	9.5	10.5	10.3	7.6	8.0	7.8	8.7	8.9				
0679	3	8.7	7.4	7.5	7.8	9.7	10.3	7.6	8.0				
0683	2	9.5	10.5	10.3	9.5	8.7	7.4	7.8	8.7				
0684	1	11.4	9.5	8.7	7.4	7.5	10.3	9.5	8.7				
0687	1	9.5	8.7	7.4	7.8	8.7	7.4	7.5	7.8				
0690	1	8.7	7.4	7.8	8.7	11.4	9.5	8.7	7.4				

The Row Locator

When you use the unpivot operator, Warehouse Builder transforms multiple input rows into a single row based on the row locator. In the unpivot operator, the row locator is an attribute that you must select from the source to correspond with a set of output attributes that you define. A row locator is required in an unpivot operator. In this example, the row locator is ‘MONTH’ from the ‘SALES’ table and it corresponds to attributes M1, M2, M3... M12 in the unpivoted output.

Using the Unpivot Operator

You have the following options for using an unpivot operator:

- **Define a new unpivot operator:** Drag an unpivot operator from the Toolbox onto the mapping. The Mapping Editor launches a wizard.
- **Edit an existing unpivot operator:** Right-click the operator and select **Edit.** The Mapping Editor opens the Unpivot Editor.

Whether you are using the Unpivot operator wizard or the Unpivot Editor, complete the following pages:

- General
- Groups
- Input Connections
- Input Attributes
- Row Locator
- Output Attributes
- Unpivot Transform

General

Use the General page to specify a name and optional description for the unpivot operator. By default, the wizard names the operator “Unpivot.”

Groups

Use the Groups page to specify one input and one output group.

In an unpivot operator, the input group represents the source data in crosstab format. The output group represents the target data distributed across multiple attributes.

You can rename and add descriptions to the input and output groups. Since each unpivot operator must have exactly one input and one output group, the wizard prevents you from adding or removing groups or changing group direction.

Input Connections

Use the Input Connections page to select attributes to copy and map into the unpivot operator.

To complete the Input connections page for an unpivot operator:

1. Select complete groups or individual attributes from the left panel.

   To search for a specific attribute or group by name, type the text in **Search for** and click **Go.** To find the next match, click **Go** again.

   Hold the Shift key down to select multiple groups or attributes. If you want to select attributes from different groups, you must first combine the groups with a Joiner or Set operator.

2. Use the left to right arrow button in the middle of the page to move your selections to the right side of the wizard page.

   You can use the right to left arrow to move groups or attributes from the input connections list. Warehouse Builder removes the selection from the input group and removes the mapping line between the source operator and the pivot operator.

Input Attributes

Use the Input Attributes page to modify the attributes you selected in the Input connections tab or wizard page.

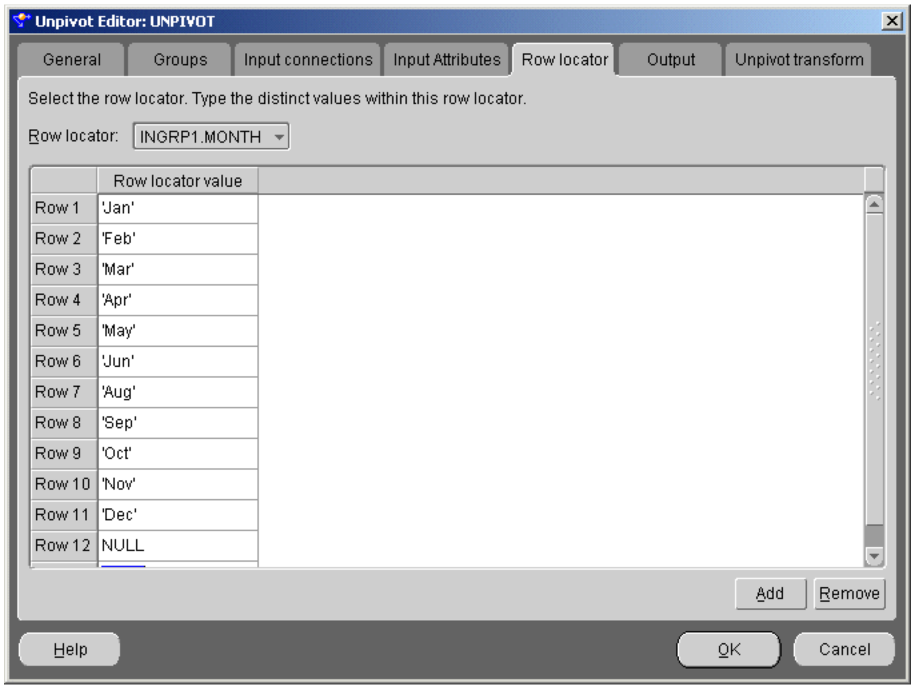
You can perform the following tasks from the Unpivot Input Attributes Page:

- **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 8–41 shows the attribute MONTH selected as the row locator with values such as ‘Jan’, ‘Feb’, or ‘Mar’.

Figure 8–41 Unpivot Row Indicator Page



Select the row locator. Type the distinct values within this row locator.

Row locator: INGRP1.MONTH

	Row locator value
Row 1	'Jan'
Row 2	'Feb'
Row 3	'Mar'
Row 4	'Apr'
Row 5	'May'
Row 6	'Jun'
Row 7	'Aug'
Row 8	'Sep'
Row 9	'Oct'
Row 10	'Nov'
Row 11	'Dec'
Row 12	NULL

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.

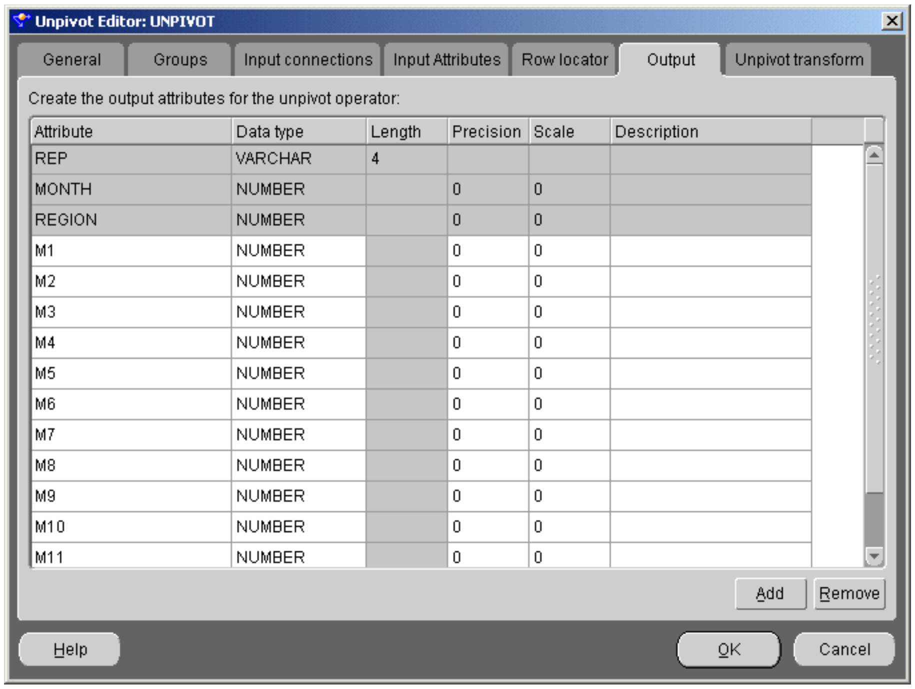
3. For each row locator value, type in the value as it appears in your source dataset.

   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 8–24 are 'Jan', 'Feb', and 'Mar.'

Output Attributes

Use the Output Attributes page shown in Figure 8–42 to create the output attributes for the unpivot operator.

Figure 8–42 Unpivot Output Attributes Page



Create the output attributes for the unpivot operator:

Attribute	Data type	Length	Precision	Scale	Description
REP	VARCHAR	4			
MONTH	NUMBER		0	0	
REGION	NUMBER		0	0	
M1	NUMBER		0	0	
M2	NUMBER		0	0	
M3	NUMBER		0	0	
M4	NUMBER		0	0	
M5	NUMBER		0	0	
M6	NUMBER		0	0	
M7	NUMBER		0	0	
M8	NUMBER		0	0	
M9	NUMBER		0	0	
M10	NUMBER		0	0	
M11	NUMBER		0	0	

If you designated any input attributes as keys on the Input Attributes tab or wizard page, Warehouse Builder displays those input attributes as output attributes that you cannot edit or remove.

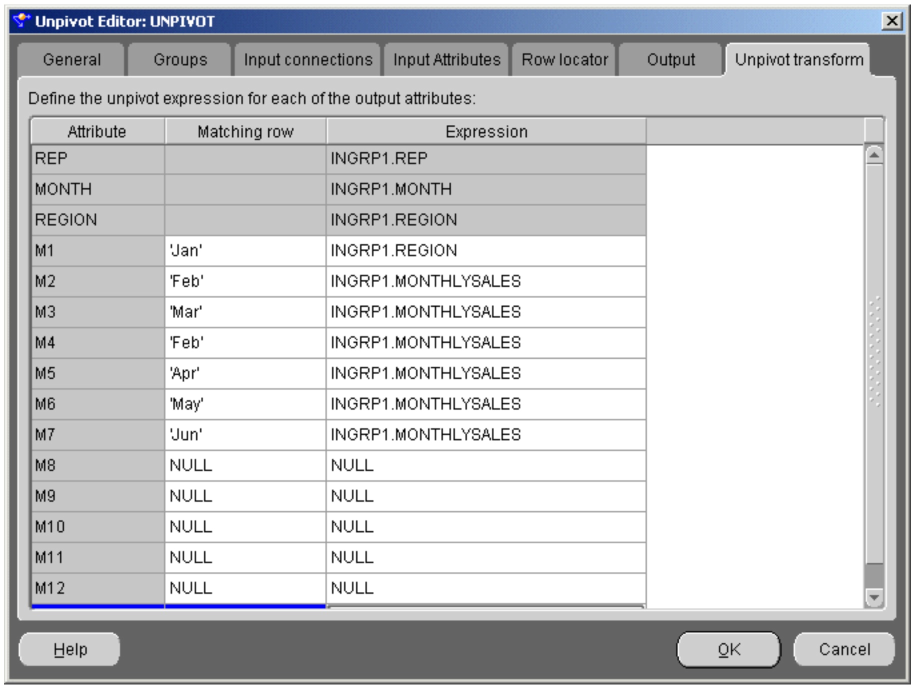
You can perform the following tasks from the Pivot Output Attributes page:

- **Add attributes:** Use **Add** to increase the number of output attributes to accommodate the rows you specified on the Row locator tab or wizard page. If you specified 12 rows, specify 12 output attributes plus attributes for any other input attributes that you did not designate as a key.
- **Change attribute properties:** Except for attributes you designated as keys on the Input Attributes tab or wizard page, you can change the attribute name, data type, length, precision, and scale.
- **Add an optional description:** Type a description for the output attributes.

Unpivot Transform

Use the Unpivot Transform page shown in Figure 8–43 to write expressions for each output attribute.

Figure 8–43 Unpivot Transform Page



Define the unpivot expression for each of the output attributes:

Attribute	Matching row	Expression
REP		INGRP1.REP
MONTH		INGRP1.MONTH
REGION		INGRP1.REGION
M1	'Jan'	INGRP1.REGION
M2	'Feb'	INGRP1.MONTHLYSALES
M3	'Mar'	INGRP1.MONTHLYSALES
M4	'Feb'	INGRP1.MONTHLYSALES
M5	'Apr'	INGRP1.MONTHLYSALES
M6	'May'	INGRP1.MONTHLYSALES
M7	'Jun'	INGRP1.MONTHLYSALES
M8	NULL	NULL
M9	NULL	NULL
M10	NULL	NULL
M11	NULL	NULL
M12	NULL	NULL

For attributes you designated as keys, Warehouse Builder defines the matching row and expression for you. Warehouse Builder displays the first row as the match for a key attribute. For all other output attributes, specify the matching row and the expression.

- **Matching row:** Select the appropriate option from the list box. For example, for the attribute you define as the first month of the year, 'M1', select 'Jan' from the list box.
- **Expression:** Select the appropriate expression from the list box. For all the new attributes you created to unpivot the data, select the same input attribute that contains the corresponding data. For example, the unpivot attributes M1, M2, M3... M12 would all share the same expression, INGRP1.MONTHLY_SALES. For all other output attributes, select the corresponding attribute from the list of input attributes.

9 Using Transformations

As you design mappings and process flows, you may want to use specialized transformations to transform data. This chapter describes how to import transformation definitions and how to create custom transformations.

This chapter includes the following topics:

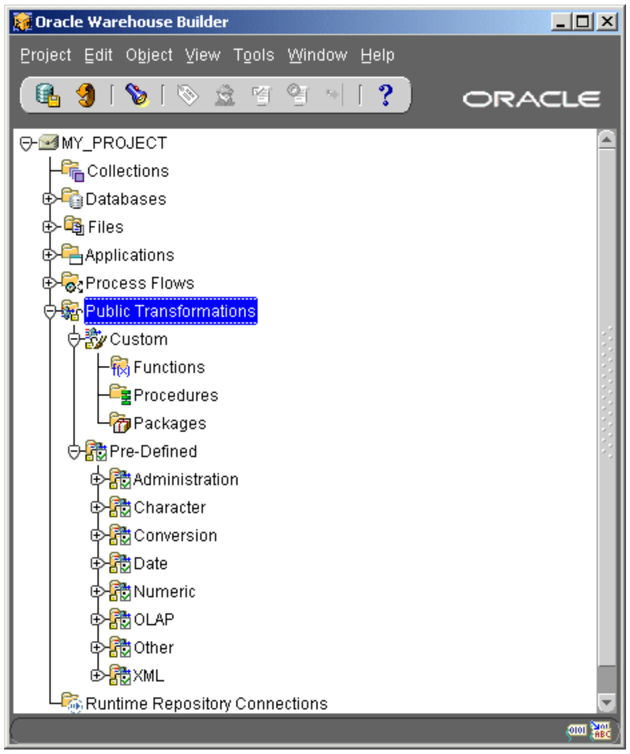
- About Transformations on page 9-1
- Defining Custom Transformations on page 9-3
- Importing PL/SQL on page 9-5
- Editing Transformation Properties on page 9-8

About Transformations

Transformations are PL/SQL functions, procedures, and packages that enable you to transform data. Warehouse Builder provides you a set of pre-defined transformations from the Oracle Library. You can also use the New Transformation Wizard to create custom transformations that define a standalone function, procedure, or package. Use transformations when designing mappings and process flows that define ETL processes.

From the Warehouse Builder navigation tree, expand the Public Transformations node to display the following types of transformations available in Warehouse Builder, as shown in Figure 9–1.

Figure 9–1 Public Transformations Folder



About Custom Transformations

The Custom library stores transformations that can be shared across different warehouse modules in a repository. It contains the following categories:

- **Functions:** The functions category is available under the Public Transformations node. This category contains any standalone functions. These functions can be defined by the user or imported from a database.
- **Procedures:** The procedures category is available under the Public Transformations node. This category contains any standalone procedures used as transformations. These procedures can be defined by the user or imported from a database. A procedure transformation takes 0-n input parameters and produces 0-n output parameters.
- **Packages:** PL/SQL packages can be created or imported in Warehouse Builder. The package body may be modified. The package header, which is the signature for the function or procedure, cannot be modified. The package can be viewed in the transformation library property sheet.

These transformation categories are also available within each warehouse module under the Transformations node. When you create functions and procedures within a warehouse module, you cannot share it across projects in the same repository.

About Pre-Defined Transformations

Warehouse Builder also provides pre-defined categories of transformations that enable you to perform common transformations quickly and easily. These built-in functions and procedures include a set of standard transformations organized into the following categories:

- Administration

- Character
- Conversion
- Date
- Numeric
- OLAP
- Other
- XML

For more information about pre-defined transformations, see the Oracle Warehouse Builder Transformation Guide.

Defining Custom Transformations

You can create custom transformations using the New Transformation Wizard.

To define a custom transformation:

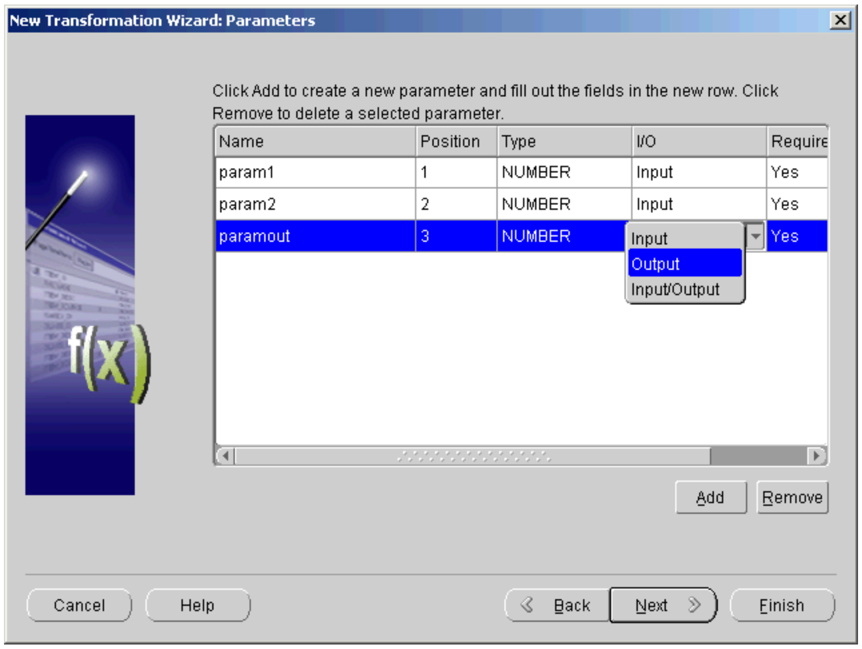
1. From the Warehouse Builder navigation tree, expand the Public Transformations node and then the Custom node. You can also expand the Oracle warehouse module node and then the Transformations node.
2. Right-click the category and select **Create Function, Create Procedure,** or **Create Package.**

   Warehouse Builder opens the New Transformation Wizard Welcome page.
3. Click **Next.**

   Warehouse Builder displays the Name page.
4. Type a name and a description for the new transformation.
5. Select a return type for the function from the drop-down list.
6. Click **Next.**

   Warehouse Builder displays the Parameters page, as shown in Figure 9–2.
7. Define each parameter for the transformation:
   a. Click **Add.**
   b. Type a name for the Parameter in the **Name** column.
   c. Specify the type, the order, whether it is an Input, Output, or Input/Output parameter, and whether the parameter is required.

Figure 9–2 Transformation Parameter Page



Click Add to create a new parameter and fill out the fields in the new row. Click Remove to delete a selected parameter.

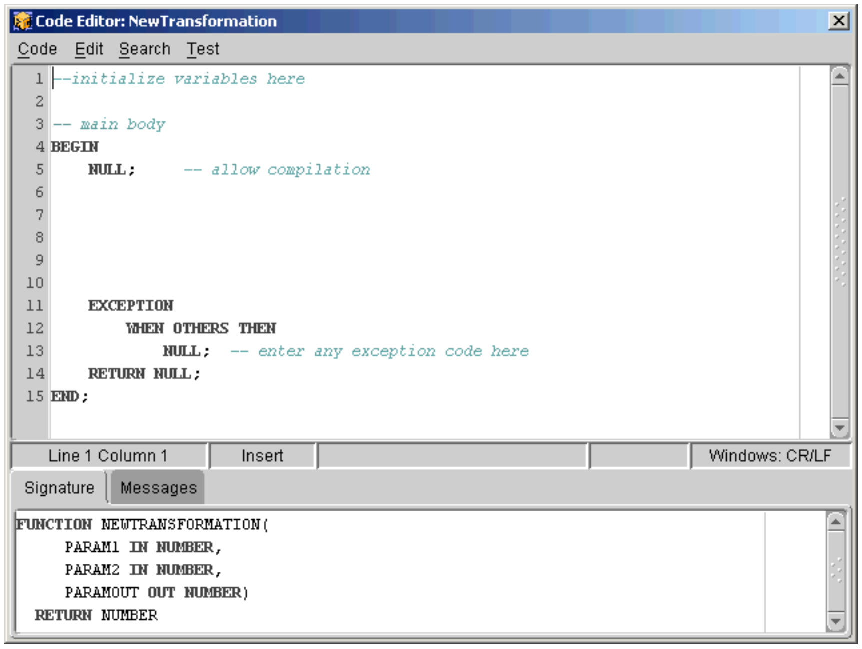
Name	Position	Type	I/O	Require
param1	1	NUMBER	Input	Yes
param2	2	NUMBER	Input	Yes
paramout	3	NUMBER	Input	Yes

8. Click **Next.**

   Warehouse Builder displays the Implementation page, as shown in Figure 9–3.

9. Click **Code Editor** to display the code editor. The code editor has line numbers, find, deploy, and syntax checking.

Figure 9–3 Code Editor for a New Transformation



```
--initialize variables here

-- main body
BEGIN
    NULL;      -- allow compilation





    EXCEPTION
        WHEN OTHERS THEN
            NULL;  -- enter any exception code here
    RETURN NULL;
END;
```

```
FUNCTION NEWTRANSFORMATION(
     PARAM1 IN NUMBER,
     PARAM2 IN NUMBER,
     PARAMOUT OUT NUMBER)
  RETURN NUMBER
```

10. Close the code editor and click **Next.**

    Warehouse Builder displays the Finish page.

11. Click **Finish.**

    Warehouse Builder creates the function, procedure, or package and displays it under the corresponding folder under the Public Transformations and Custom nodes in the navigation tree.

Importing PL/SQL

Using the Import Wizard, you can import PL/SQL functions, procedures, and packages into a Warehouse Builder project.

The following steps describe how to import PL/SQL packages from other sources into Warehouse Builder.

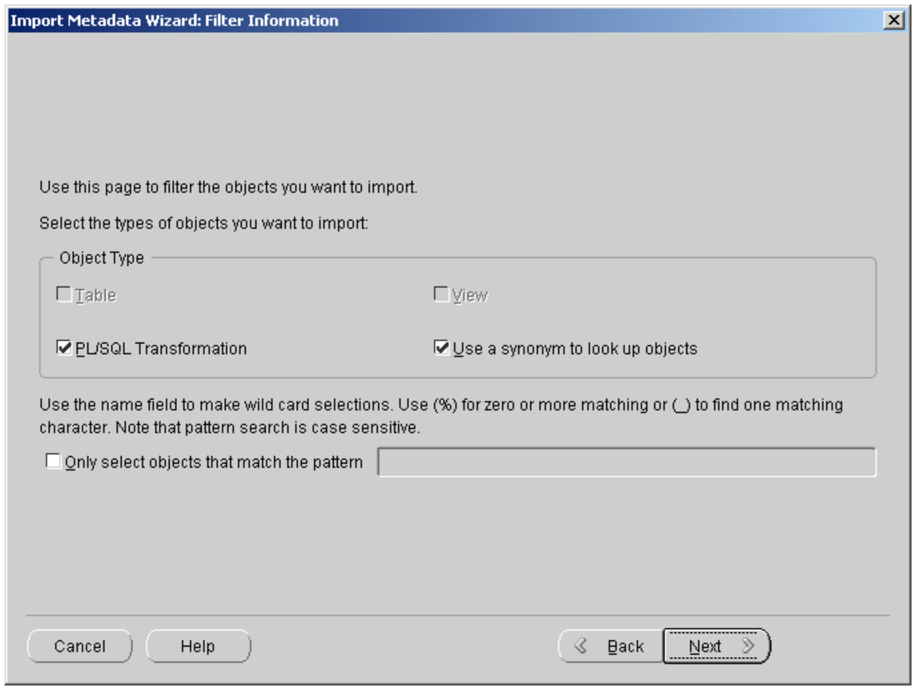
To import a PL/SQL function, procedure, or package:

1. From the Warehouse Builder navigation tree, expand the Public Transformations node.
2. Right-click the Custom node and select **Import**. Or right-click the warehouse module name and select **Import.**

   Warehouse Builder displays the Database Link Information dialog.
3. Create a new database link to the system from where you are importing the PL/SQL package. Or select a database link from the list.
4. Click **OK.**

   Warehouse Builder displays the Import Metadata Wizard Welcome page.
5. Click **Next.**
6. Select **PL/SQL Transformation** in the Object Type field of the Filter Information page, as shown in Figure 9–4.

Figure 9–4 PL/SQL Transformation Selection



Import Metadata Wizard: Filter Information

Use this page to filter the objects you want to import.

Select the types of objects you want to import:

Object Type

☐ Table ☐ View

☑ PL/SQL Transformation ☑ Use a synonym to look up objects

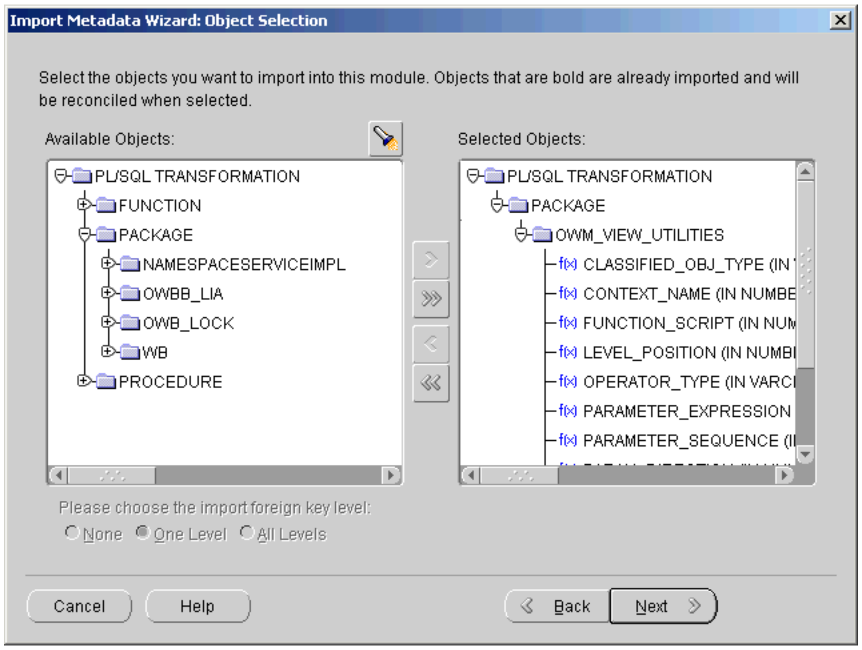
Use the name field to make wild card selections. Use (%) for zero or more matching or (_) to find one matching character. Note that pattern search is case sensitive.

☐ Only select objects that match the pattern

7. Click **Next.**

   The Import Metadata Wizard displays the Object Selection page, as shown in Figure 9–5.

Figure 9–5 Object Selection Page



Import Metadata Wizard: Object Selection

Select the objects you want to import into this module. Objects that are bold are already imported and will be reconciled when selected.

Available Objects:

```
PL/SQL TRANSFORMATION
  FUNCTION
  PACKAGE
    NAMESPACESERVICEIMPL
    OWBB_LIA
    OWB_LOCK
    WB
  PROCEDURE
```

Selected Objects:

```
PL/SQL TRANSFORMATION
  PACKAGE
    OWM_VIEW_UTILITIES
      f(x) CLASSIFIED_OBJ_TYPE (IN
      f(x) CONTEXT_NAME (IN NUMBE
      f(x) FUNCTION_SCRIPT (IN NUM
      f(x) LEVEL_POSITION (IN NUMBI
      f(x) OPERATOR_TYPE (IN VARC
      f(x) PARAMETER_EXPRESSION
      f(x) PARAMETER_SEQUENCE (I
```

Please choose the import foreign key level:
None [✓ One Level] All Levels

8. Select a function, procedure, or package from the Available Objects list. Move the objects to the Selected Objects list by clicking the single arrow button to move a single object or the double arrow button to move multiple objects.

   Check if 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.

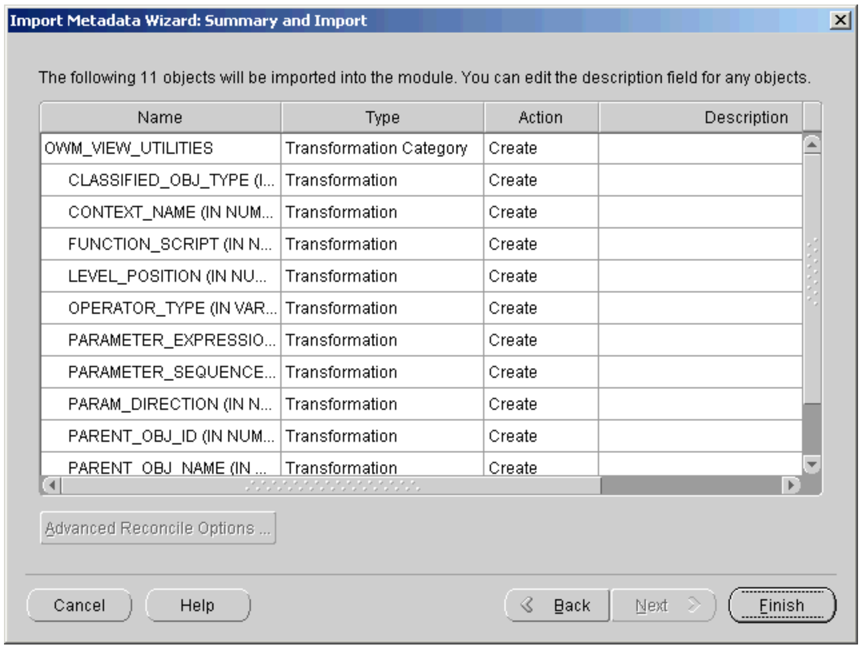
   Check if function should be enabled for parallel execution:

   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.

9. Click **Next.**

   The Import Metadata Wizard displays the Summary and Import page, as shown in Figure 9–6.

Figure 9–6 Summary and Import Page


Import Metadata Wizard: Summary and Import
Advanced Reconcile Options ...
Cancel
Help
Back
Next
Finish

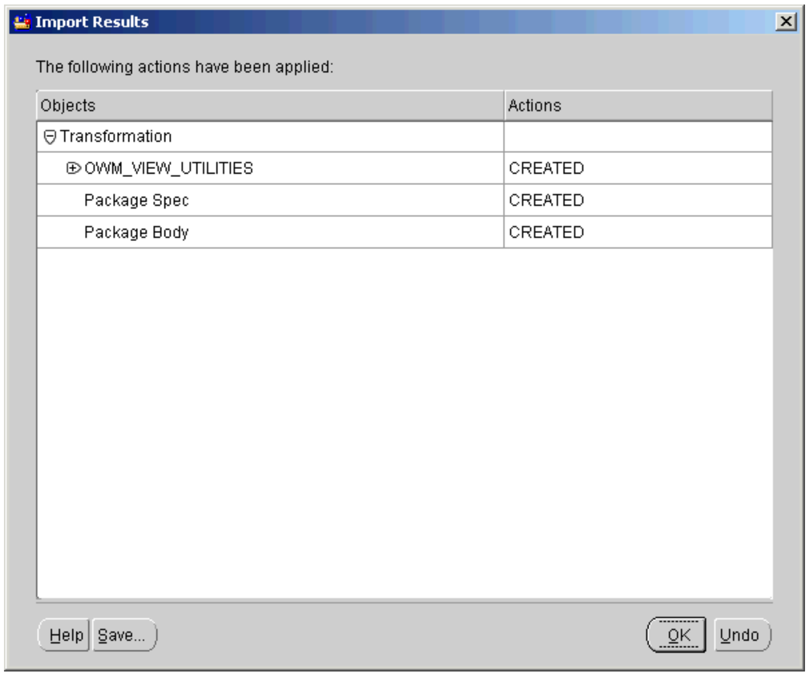
The following 11 objects will be imported into the module. You can edit the description field for any objects.

Name	Type	Action	Description
OWM_VIEW_UTILITIES	Transformation Category	Create	
CLASSIFIED_OBJ_TYPE (I...	Transformation	Create	
CONTEXT_NAME (IN NUM...	Transformation	Create	
FUNCTION_SCRIPT (IN N...	Transformation	Create	
LEVEL_POSITION (IN NU...	Transformation	Create	
OPERATOR_TYPE (IN VAR...	Transformation	Create	
PARAMETER_EXPRESSIO...	Transformation	Create	
PARAMETER_SEQUENCE...	Transformation	Create	
PARAM_DIRECTION (IN N...	Transformation	Create	
PARENT_OBJ_ID (IN NUM...	Transformation	Create	
PARENT_OBJ_NAME (IN ...	Transformation	Create	

10. Verify the import information. Click **Back** to revise your selections.
11. Click **Finish** to import.

    The Import Results dialog displays, as shown in Figure 9–7.

Figure 9–7 Import Results



The following actions have been applied:

Objects	Actions
Transformation	
OWM_VIEW_UTILITIES	CREATED
Package Spec	CREATED
Package Body	CREATED

12. Click **OK** proceed with the import. Click **Undo** to cancel the import process.

    The imported PL/SQL information appears under the Custom node in the navigation tree.

When you use imported PL/SQL:

- You can edit, save, and deploy the imported PL/SQL functions and procedures.
- Wrapped PL/SQL objects are not readable.
- Imported packages can be viewed and modified in the category property sheet.
- You can edit the imported package body but not the imported package specification.

Editing Transformation Properties

You can edit a function, procedure, or package from the Transformation Properties Sheet. 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.

To edit a function, procedure, or package:

1. From the Warehouse Builder navigation tree, 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 **Properties.**

   Warehouse Builder displays the Transformation Properties Sheet. For Packages, you can only edit the name and description of the package. For Functions and Procedures the Properties Sheet displays three tabs: Name, Parameters, and Implementation.

   Name: You can edit the name and description of the function or procedure. For functions, you can also edit the return data type.

   Parameters: You can edit, add, or delete new parameters for a function or procedure. You can also edit and define the attributes of the parameters.

   Implementation: Review the PL/SQL code for the parameter. Click **Code Editor** to edit the code.

10

Designing Process Flows

After you create mappings that define the operations for moving data from sources to targets, you can create and define process flows. Process flows interrelate mappings and activities external to Warehouse Builder such as email, FTP commands, and operating system executables.

This chapter contains the following topics:

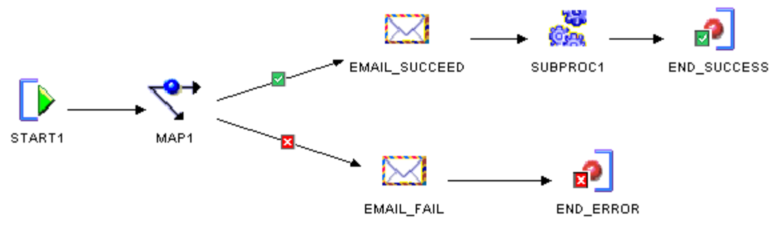
- About Process Flows on page 10-1
- Procedure for Defining Process Flows on page 10-2
- About the Process Flow Editor on page 10-4
- Process Flow Editor Menu and Toolbar on page 10-22

About Process Flows

A process flow describes dependencies between Warehouse Builder mappings and external activities such as email, FTP, and operating system commands.

Each process flow begins with a Start activity and concludes with an End activity for each stream in the flow. You can design a process flow to launch other process flows. Figure 10–1 shows an example of a process flow that launches a mapping MAP1. If the mapping completes successfully, Warehouse Builder sends an email notification EMAIL_SUCCEED and launches another process flow SUBPROC1. If the mapping fails, Warehouse Builder sends an email EMAIL_FAIL and ends the process flow.

Figure 10–1 Sample Process Flow



When you design a process flow in Warehouse Builder, you use an interface known as the Process Flow Editor. Alternatively, you can create and define process flows using the Warehouse Builder scripting language, OMB Scripting Language. For information on how to create and define process flows in OMB Plus, the scripting interface, see the Oracle Warehouse Builder Scripting Reference .

Warehouse Builder process flows comply with the XML Process Definition Language (XPDL) standard set forth by the Workflow Management Coalition (WfMC). When you generate a process flow, Warehouse Builder generates an XML file in the XPDL format. You can plug the generated XML file into any workflow engine that follows the XPDL standard.

About Process Flow Modules

Like all modules Warehouse Builder, process flow modules allow you to group objects to be deployed to a common Location. For instructions, see "Creating Process Flow Modules" on page 10-2.

From the Warehouse Builder Deployment Manager, you can deploy process flows to Oracle Workflow or any XPDL complaint process flow engine. For information about Oracle Workflow, see the *Oracle Workflow Guide.* You can also execute and schedule process flows using Oracle Enterprise Manager. For information about Oracle Enterprise Manager, see the *Oracle Enterprise Manager Administrator's Guide.*

About Process Flow Packages

Process flow modules contain process flow packages. The process flow package is the grouping mechanism for process flows you plan to deploy as one unit. At runtime, you can launch one process flow that launches other process flows that exist in the same process flow package. For example, Figure 10–2 depicts a process flow PROC1 that includes process flow SUBPROC1. In order to add SUBPROC1 to PROC1, the process flows must be contained within the same process flow package.

For instructions on how to create a process flow package, see "Creating Process Flow Packages" on page 10-3.

Procedure for Defining Process Flows

To define a process flow, complete the following steps:

1. Creating Process Flow Modules on page 10-2
2. Creating Process Flow Packages on page 10-3
3. Creating Process Flow Definitions on page 10-3
4. Adding Activities on page 10-7
5. Connecting Activities on page 10-10
6. Configuring Process Flows on page 11-28
7. Validating and Generating Process Flows. For more information on validating code for process flows, see Chapter 12, "Validating Objects".

Creating Process Flow Modules

Before working with process flows, create a process flow module. The module is the 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 Flows** node in the navigation tree and select **Create Process Flow Module.**

   Warehouse Builder displays the Welcome page for the New Module Wizard.
2. Click **Next.**

The New Module Wizard displays the Name page.

3. On the Name page, type:

   Module name: Module names must be unique within a project.

   Module status: Specify the status as Development, Quality Assurance, or Production. This status is for descriptive purposes only.

   Description: Enter an optional text description.

4. Click **Next.**

   The New Module Wizard displays the Location page.

   You can select a location from the drop down box or leave the location unspecified. If you do not specify a location in the wizard, you must specify a location on the process flow module properties sheet before you can deploy process flows packages in the module.

5. Click **Next.**

   The New Module Wizard displays the Finish page. Verify the name and location of the new process flow module.

6. Click **Finish.**

   Warehouse Builder stores the definition for the module and inserts its name in the warehouse module navigation tree. You can now create a process flow package.

Creating Process Flow Packages

After you create a module for process flows, you can create the process flow package. The process flow package is the grouping mechanism that determines which process flows you can interrelate. At runtime, you can launch one process flow that launches other process flows that exist in the same process flow package.

To create a process flow package:

1. Right-click a process flow module in the navigation tree and select **Create Process Flow Package.**

   Warehouse Builder displays the Create Process Flow Package dialog.

2. Type a name and optional description for the process flow package.
3. Select **OK.**

   You can now create a process flow definition.

Creating Process Flow Definitions

After you create a module and package for process flows, you can create the process flow definition.

To create a process flow definition:

1. Right-click a process flow package in the navigation tree and select **Create Process Flow.**

   Warehouse Builder displays the Create Process Flow dialog.

2. Type a name and optional description for the process flow.
3. Select **OK.**

Warehouse Builder launches the Process Flow and displays the Graph View tab with a Start activity and an End activity.

You can now model the process flow with activities and transitions.

About the Process Flow Editor

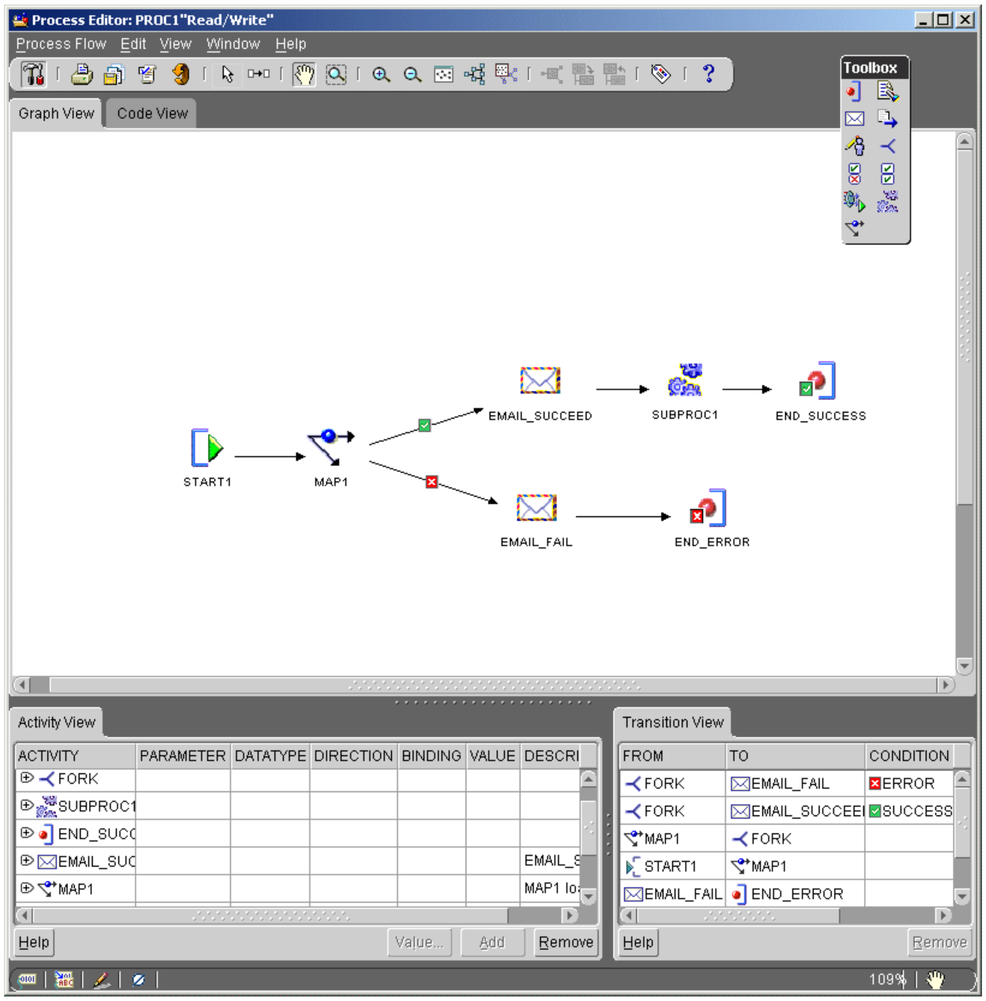
Once you create a process flow module and package, use the Process Flow Editor to create, design, and edit process flows. The Process Flow Editor includes a variety of activities that you add and then connect with transitions to design a flow.

Activity represents units of work in a process flow. These unit of work can involve components internal or external to Warehouse Builder.

Transitions indicate the sequence and conditions in which to launch activities.

Figure 10–2 shows the Process Flow Editor with a sample process flow. Activities display as icons on the canvas. In this example, the activities include a start, a mapping activity, 2 email activities, a subprocess, and two end activities. Arrows between the activities represent the transitions.

Figure 10–2 Process Flow Editor



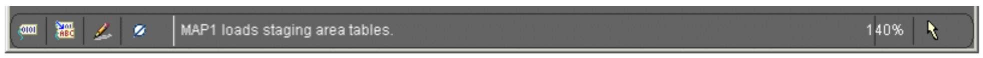
ACTIVITY	PARAMETER	DATATYPE	DIRECTION	BINDING	VALUE	DESCRI
FORK						
SUBPROC1						
END_SUCC						
EMAIL_SUC						EMAIL_S
MAP1						MAP1 lo

FROM	TO	CONDITION
FORK	EMAIL_FAIL	🟥ERROR
FORK	EMAIL_SUCCEE	🟩SUCCESS
MAP1	FORK	
START1	MAP1	
EMAIL_FAIL	END_ERROR	

The Process Flow Editor has of the following components:

- **Title Bar:** At the top of the editor, the title bar displays the name of the process flow and the access privileges you have on the process flow. In Figure 10–2, the name is PROC1 and the user has Read/Write privileges.
- **Menu Bar:** Below the title bar, the menu bar provides access to the process flow editor commands. For a description of each command on the menu bar, see Menu Bar on page 10-5.
- **Toolbar:** Below the menu bar, the toolbar provides icons for commonly used commands. For a description of each command on the menu bar, see Table 10–14 on page 10-24.
- **Toolbox:** When you first launch a Process Flow Editor, Warehouse Builder displays the toolbox in the upper right corner. You can relocate the toolbox anywhere on the editor. You can choose to hide or display the toolbox by clicking on the Toggle Tool Palette icon on the toolbar. The toolbox contains activity icons that you can drag and drop onto the canvas. For a description of each activity in the toolbox, see Table 10–2 on page 10-11.
- **Canvas:** The canvas provides the work space where you design and modify process flows. There are two tabs on the process flow canvas. The Graph View provides a graphical representation of the process flow. The Code View displays code. When you first create a new process, the Process Flow displays the Graph View tab with a Start activity and an End activity.
- **Activity View and Transition View:** These editing dialogs appear under the canvas. To view and edit the properties for all the activities and transitions, click the white space in the canvas. To view and edit the properties for a single activity or transition, choose the Select mode from the toolbar and select a transition or activity.
- **Indicator Bar:** In the lower panel, under the Activity panel and Transition panel, you can see mode icons, indicators, and descriptions as shown in Figure 10–3.

Figure 10–3 Indicator Bar on the Process Flow Editor



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

In the right corner are the percent zoom indicator and the navigation mode. In the preceding figure, the zoom level is at 140% and the navigation mode is set to Select Mode.

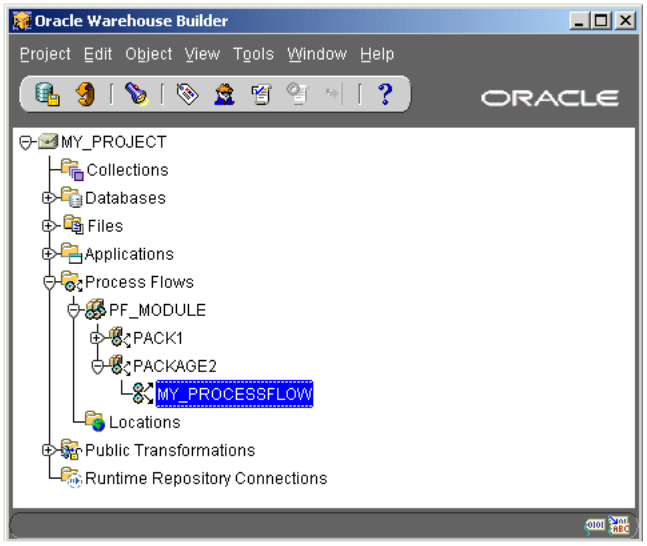
When you select an activity or transition from the canvas, the text you entered for the description appears on the indicator bar. In the preceding figure, the description reads, “MAP1 loads staging area tables.”

To display the Process Flow Editor:

1. From **Process Flows** on the navigation tree, select a process flow module. If no process flow modules are listed, create a process flow module. For instructions, see "Creating Process Flow Modules" on page 10-2.
2. Select a process flow package from a process flow module. If no process flow packages are listed, create a process flow package. For instructions, see "Creating Process Flow Packages" on page 10-3.

3. Select a process flow from the navigation tree as shown in Figure 10–4. If no process flows are listed in the process flow package, right-click the process flow package and select **Create Process Flow.**

Figure 10–4 Process Flow in the Navigation Tree



4. After you create a process flow definition, you can open it by double-clicking on the process flow in the navigation tree.

   Or, select a process flow and then from the **Object** menu, select **Editor.** Or, select a process flow and type **Ctrl** and the letter **O.** Or, Right-click a process flow, and select **Editor...** from the pop-up menu.

   Warehouse Builder displays the Process Flow Editor. By default, the Process Flow Editor displays the Graph View in the select mode.

Navigating in the Process Flow Editor

The Process Flow Editor includes a variety of tools to assist you in navigating, selecting, and viewing objects on the canvas. Commands you use frequently when designing process flows include the following:

Select mode: Use Select mode to select objects on the canvas. When you select an activity, the editor displays a blue border around the activity and you can edit, move, or delete the activity. You can edit the activity in the Activity View or you can right click for more options. When you select a transition, the editor changes the arrow from black to blue. You can edit the transition in the Transition View or right click for more options. To activate the Select mode, click the icon in the toolbar or select **Edit** and **Select mode** from the menu.

Create transition: Use the create transition mode to connect activities on the canvas. Create transition is a drawing tool only. To edit or delete a transition, switch to the select mode and select the transition. To activate the create transition mode, click the icon in the toolbar or select **Edit** and **Create transition** from the menu.

For navigating and displaying complex process flows, you may find the following tools useful:

- Panning
- Interactive zoom
- Zoom in
- Zoom out
- Fit in window
- Auto layout
- Birds eye view
- Zoom
- Center

For more information on these and other tools available in the Process Flow Editor, see "Process Flow Editor Menu and Toolbar" on page 10-22.

About Activities

The basic design elements for a process flow include activities and transitions. Activities represent units of work for the process flow. When you design a process flow in Warehouse Builder, you select activities from the Process Flow Editor toolbox and drag them onto the canvas.

For a description of each activity, see "Using Activities in Process Flows" on page 10-11.

Adding Activities

To add an activity to a process flow:

1. Select an activity from the Toolbox and drag it onto the canvas.

   When you add transformations, mappings, and subprocesses, select the object from the Add Activity dialog.

   The Process Flow Editor displays the activity on the canvas and lists the activity on the **Activity Panel,** which is located at the lower left corner of the Process Flow Editor.

2. In the **Activity Panel,** select the activity.

   You can rename the activity or accept the default name.

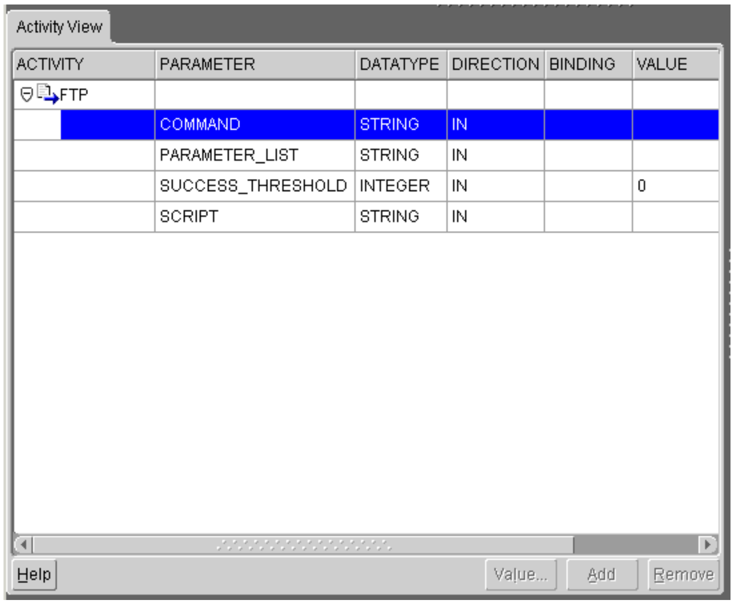
   You can add an optional description for the activity. This description appears on the indicator bar at the bottom of the Process Flow Editor when you select the activity in the canvas.

   To navigate and modify complex process flows easily, add descriptions for each activity.

3. In the **Activity Panel,** expand the activity by clicking on the plus icon to the left of the activity name.

   The Process Flow Editor displays the parameters for the activity. These properties vary according to the type of activity. Figure 10–5, for example, shows the properties for an FTP activity.

Figure 10–5 Parameters for an FTP activity



ACTIVITY	PARAMETER	DATATYPE	DIRECTION	BINDING	VALUE
FTP					
	COMMAND	STRING	IN		
	PARAMETER_LIST	STRING	IN		
	SUCCESS_THRESHOLD	INTEGER	IN		0
	SCRIPT	STRING	IN		

- **Parameter:** The name of the activity parameter, such as COMMAND and PARAMETER in Figure 10–5. For information about a specific parameter, look up the activity by name under "Using Activities in Process Flows" on page 10-11.
- **Datatype:** For all activities except the Start activity, the parameter datatype is read-only. Warehouse Builder assigns the appropriate datatype for all default parameters.
- **Direction:** The parameter direction is read-only. A direction of IN indicates that the parameter is an input parameter for the activity.
- **Binding:** Use binding to pass in parameters from outside the process flow. If you assign a parameter in **Binding,** it overrides any text you assign to **Value.** For more information on binding, see "Binding and Passing Parameters to Activities" on page 10-8.
- **Value:** Warehouse Builder assigns default values to some parameters. To override the default, type in a new value.
- **Description:** You can type an optional description for each property.

Binding and Passing Parameters to Activities

You can pass in external values into a process flow. To dynamically change the values passed to various activities at runtime, execute the process flow with different parameter values.

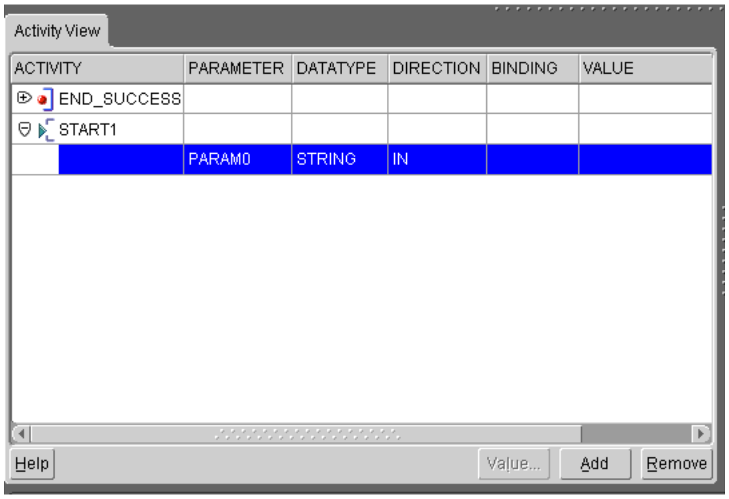
You can input values to activity parameters based on parameters you defined for the Start activity. This is referred to as binding. To bind an activity, the activity parameter must match the datatype and direction of the input parameter in the Start activity. If you want to pass in a date into the process flow, verify that the date is in the format 'mm-dd-yyyy'.

To pass values to an activity:

1. To add an input parameter to the process flow, select the Start activity and select **Add** at the bottom of the Activity Panel.

The Process Flow Editor adds a parameter under the Start activity as shown in Figure 10–6.

Figure 10–6 Adding a parameter to a Start activity



ACTIVITY	PARAMETER	DATATYPE	DIRECTION	BINDING	VALUE
END_SUCCESS					
START1					
	PARAM0	STRING	IN		

2. Set the properties for the parameter.

   Change the parameter name and datatype as necessary. You cannot alter the direction and binding. The direction is IN, indicating that the parameter is an input parameter only. For value, type the parameter value.

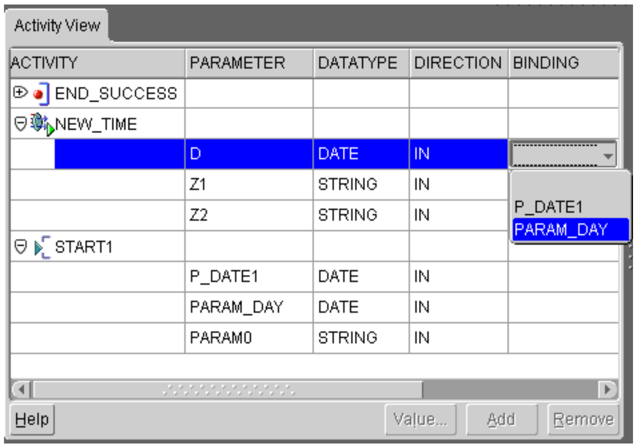
3. Select the activity you want to receive the parameter.

   In this step, select an activity other than the start activity you used in the previous steps of these instructions.

4. Expand the activity to receive the parameter, select its parameter, and select **Binding.**

   The Process Flow Editor displays a list of start activity input parameters matching the datatype and direction of the activity parameter you selected. Figure 10–7 shows PARM_DAY defined in START1 and selected as input for parameter Z2 in the transformation activity NEW_TIME.

Figure 10–7 Binding an Activity Parameter



ACTIVITY	PARAMETER	DATATYPE	DIRECTION	BINDING
END_SUCCESS				
NEW_TIME				
	D	DATE	IN	
	Z1	STRING	IN	
	Z2	STRING	IN	
START1				
	P_DATE1	DATE	IN	
	PARAM_DAY	DATE	IN	
	PARAM0	STRING	IN	

About Transitions

Use transitions to indicate the sequence and conditions in which activities occur in the process flow. You can use transitions to execute an activity based on the completion state of the preceding activity.

Table 10–1 lists the types of conditions you can specify for a transition and shows the associated icons on the canvas.

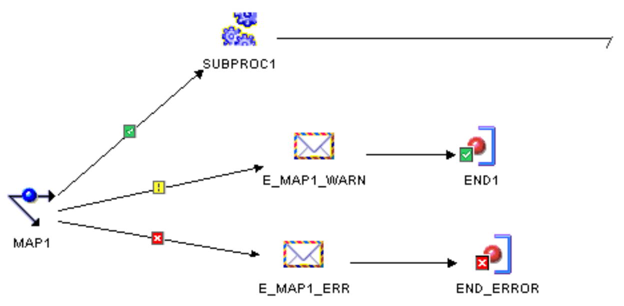
Table 10–1 Types of Conditions for Transitions

Icon	Transition	Description
↗	Unconditional	When you add a transition to the canvas, by default, the transition has no condition applied to it. The Process Flow Editor continues the flow once the preceding activity completes. The process flow continues regardless of the ending state of the previous activity.
✓ ↗	Success	The Process Flow Editor continues the flow only if the preceding activity ends in success.
! ↗	Warning	The Process Flow Editor continues the flow only if the preceding activity ends with warnings.
✗ ↗	Error	The Process Flow Editor continues the flow only if the preceding activity ends in error.

If the activity has only one outgoing activity, you can specify any of the four conditions listed in Table 10–1. When you add multiple outgoing transitions to an activity, ensure that the conditions do not conflict. A conflict occurs when the process flow logic can evaluate to more than one outgoing transition being true. For a single activity, you can specify either an unconditional outgoing transition OR one of each of the following transition conditions: SUCCESS, WARNING, and ERROR. Transition conditions are evaluated in the following order: SUCCESS, WARNING, ERROR, and UNCONDITIONAL.

Figure 10–8 shows a portion of a process flow in which different activities are triggered based on the three possible completion states of MAP1. Since only one of these conditions can be satisfied at a time, there is no conflict. If you attempt to add an unconditional transition or another conditional transition, two transition conditions would be true and the process flow would be invalid.

Figure 10–8 Outgoing Transition Conditions



Connecting Activities

To create dependencies using transitions:

1. From the Process Flow Editor, select **Edit** and **Create Transition** or select the Create Transition icon from the menu bar.

   The Process Flow Editor displays the cursor as a small horizontal arrow, indicating that you can now use the mouse button to connect activities.

2. Place the cursor over an activity you want to connect from and press the left mouse button. Continue to press the left mouse button until you place the cursor over the second activity.

   The Process Flow Editor displays an arrow between the two activities and adds an entry in the Transition Panel.

3. In the Transition Panel, select the transition. The Transition Panel displays the following properties for each transition:

   From: This property is read-only and indicates the first activity in the connection.

   To: This property is read-only and indicates the second activity in the connection.

   Condition: You can specify a success, error, or warning condition for a transition. When you select a condition, the Process Flow Editor displays the associated icon inserted into the transition line on the canvas.

   Description: You can type an optional description for the transition. When you select the transition on the canvas, the description displays on the indicator bar.

Using Activities in Process Flows

This section begins with a table that summarizes all activities followed by detailed explanations for using each activity. Activities are presented in alphabetical order.

Table 10–2 lists each activity and shows the associated icon in the Process Flow Editor. It also lists the number of incoming and outgoing transitions allowed for each activity.

Table 10–2 Process Flow Activities

Icon	Activity	Brief Description	Incoming Transitions	Outgoing Transitions
☑	AND	Use the AND activity to specify the completion of all incoming activities before launching another activity.	2 or more allowed	1 unconditional transition only
✉	Email	Use the Email activity to send email. For example, send notifications about the status of activities in the process flow.	1 or more allowed	1 unconditional transition OR up to 3 conditional transitions
	End	By default, Warehouse Builder adds an END_SUCCESS to each process flow. You can include up to 3 End activities in the flow: success, error, or warning.	1 or more allowed	not allowed
	External Process	Use the external process activity to represent an activity not defined by Warehouse Builder and to incorporate it into a process flow.	1 or more allowed	1 unconditional transition OR up to 3 conditional transitions
	File Exists	Use the File Exists activity to check if a file is located on a specified drive or directory. The File Exists activity waits for the file to arrive before resuming the process flow.	1 or more allowed	1 unconditional transition OR up to 3 conditional transitions

Table 10–2 (Cont.) Process Flow Activities

Icon	Activity	Brief Description	Incoming Transitions	Outgoing Transitions
	FORK	Use the Fork activity to launch two or more activities after completing an activity.	1 or more allowed	2 or more unconditional transitions
	FTP	Use the FTP activity to launch a file transfer protocol command during a process flow. For example, use FTP to move data files to the machine where a mapping executes.	1 or more allowed	1 unconditional transition OR up to 3 conditional transitions
	Mapping	Use the Mapping activity to add an existing mapping to the process flow.	1 or more allowed	1 unconditional transition OR up to 3 conditional transitions
	OR	Use the OR activity to launch an activity after the completion of any of two or more specified activities.	2 or more allowed	1 unconditional transition only
	Start	By default, Warehouse Builder adds a Start activity to each process flow. You can pass parameters in a process flow through the Start activity.	not allowed	1 unconditional transition only
	Subprocess	Use the Subprocess activity to embed an existing process flow within the process flow.	1 or more allowed	1 unconditional transition OR up to 3 conditional transitions
	Transform	Use the Transform activity to add an existing transformation to the process flow.	1 or more allowed	1 unconditional transition OR up to 3 conditional transitions

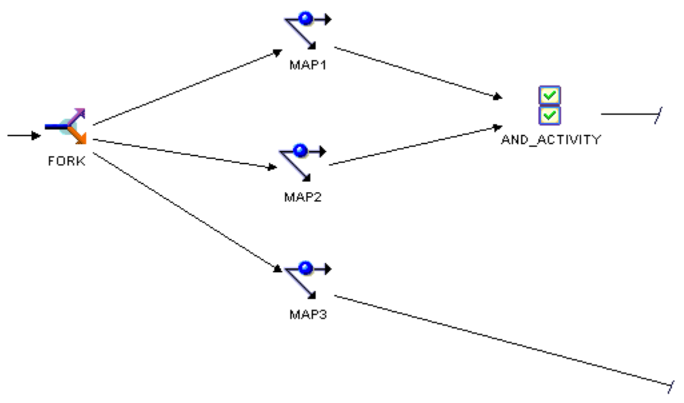
AND

Use the AND activity to specify the completion of two or more activities before resuming the process flow. The AND activity can have two or more incoming transitions but can have only one unconditional outgoing transition.

To correctly design process flows with an AND activity, you must place a FORK activity upstream of the AND. Also, the number of transitions going into the AND must be less than or equal to the number of outgoing transitions from the upstream FORK. The FORK is the only activity that enables you to assign multiple unconditional transitions and therefore ensure the completion of multiple activities as required by the AND activity.

Figure 10–9 shows the AND and FORK activities in a process flow. In this example, AND_ACTIVITY triggers downstream activities based on the completion of MAP1 and MAP2. The process flow is valid because the FORK activity has 3 outgoing transitions while AND_ACTIVITY has 2 incoming transitions. The process flow would also be valid if the transition and activities associated with MAP3 were deleted.

Figure 10–9 AND Activity in a Process Flow



Email

You can instruct Warehouse Builder to send email notifications after the completion of an activity in a process flow. You may find this useful, for example, for notifying administrators when activities such as mappings end in error or warnings.

Table 10–3 lists the parameters you set for the email activity.

Table 10–3 Email Activity Parameters

Parameter	Description
SMTP Server	The name of outgoing mail server. The default value is `Local Host`.
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 that recipients should respond.
To_Address	The email address(es) or mailing list(s) that receive the process flow notification. Use a comma or a semi-colon to separate multiple email addresses.
CC_Address	The email address(es) or mailing list(s) that receive a copy of the process flow notification. Use a comma or a semi-colon to separate multiple email addresses
BCC_Adress	The email address(es) or mailing list(s) that receive a blind copy of the process flow notification. Use a comma or a semi-colon to separate multiple email addresses
Importance	The level of importance for the notification. You can type in any text such as, for example, `High`, `Medium`, or `Low`.
Subject	The text that appears in the email subject line.
Message_Body	The text that appears in the body of the email. To type in or paste text, select **Value** at the bottom of the Activity panel. The Process Flow Editor does not limit you on the amount of text you can enter.

For email addresses, you can enter the email address with or without the display name. For example, the following entries are correct:

```
jack.emp@oracle.com
```

```
Jack Emp<jack.emp@oracle.com>
```

```
Jack Emp[jack.emp@oracle.com]
Jack Emp[jack.emp@oracle.com],Jill Emp[jill.emp@oracle.com]
Jack Emp[jack.emp@oracle.com];Jill Emp[jill.emp@oracle.com]
```

End

Every path in the process flow must terminate in an End activity.

When you first create a process flow, Warehouse Builder includes a success type End activity by default. Use end types to indicate the type of logic contained in a path. Since a given activity such as a mapping has three possible outcomes, the Process Flow Editor includes three ending types, as shown in Table 10–4. You can use these ending types to design error handling logic for the process flow.

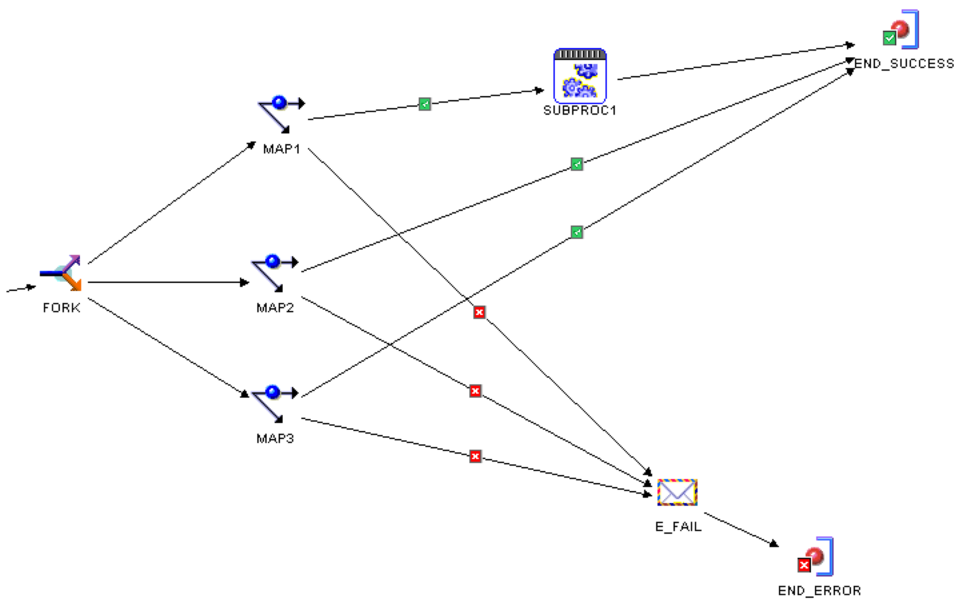
Table 10–4 Types of End Activities

Icon	End Type	Description
✓	Success	Indicates that the path or paths contain logic dependent upon the successful completion of an upstream activity.
!	Warning	Indicates that the path or paths contain logic dependent upon an upstream activity completing with warnings.
✗	Error	Indicates that the path or paths contain logic dependent upon an upstream activity completing with errors.

You can design a process flow to include one, two, or all three types of endings. You can use each ending type only once; duplicate ending types are not allowed. Each End activity can have a single or multiple incoming transitions.

In Figure 10–10, END_SUCCESS has three incoming transitions, each dependent upon the successful completion of upstream activities. END_ERROR has one incoming transition from an email activity that runs when any of the upstream mapping activities completes with errors.

Figure 10–10 End Activities in a Process Flow



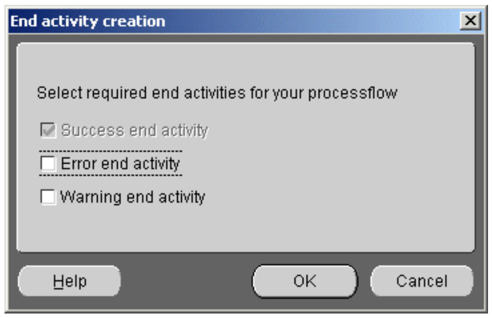
By default, every process flow includes an END_SUCCESS. Although you cannot change an end activity to another type, you can add others of different types.

To add end activities to a process flow:

1. From the toolbox on the Process Flow Editor, drag and drop the End icon onto the canvas.

   The Process Flow Editor displays a dialog for adding end activities, as shown in Figure 10–11.

Figure 10–11 Add End Activity Dialog



End activity creation

Select required end activities for your processflow

[✓] Success end activity
[ ] Error end activity
[ ] Warning end activity

2. Select from the list of available end types.

   Warehouse Builder does not allow you to select ending types already present in the process flow.

3. Select **OK.**

   Warehouse Builder adds the End activity or activities to the canvas.

External Process

The external process activity enables you to incorporate into a process flow an activity not defined within Warehouse Builder.

You can specify one or more incoming transitions to launch a external process activity. For outgoing transitions, you can either specify one unconditional transition or one of each of the three conditional transitions.

If you specify conditional outgoing transitions, you can configure the activity to base its status on it return value. For more information on **Use Return as Status,** see"Configuring Process Flows" on page 11-28.

During code generation, Warehouse Builder generates a job control code (TCL script) for external processes that you can deploy as part of the Warehouse Builder workflow.

Table 10–5 lists the parameters you set for the FTP activity.

Table 10–5 External Process Activity Parameters

Parameter	Description
Command	The command to execute the external process you defined. Type the path and file name such as `c:\winnt\system32\cmd.exe` .

Table 10–5 (Cont.) External Process Activity Parameters

Parameter	Description
Parameter List	The list of parameters to be passed to the external 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 10–6.
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 filename for a script. If you type in a filename, use the ${Task.Input} variable in the paramter list to pass the filename. To type in or paste text, select **Value** at the bottom of the Activity panel. The Process Flow Editor does not limit you on the amount of text you can enter. Each carriage return in the script is equivalent to pressing the *`Enter`* key. Therefore, end the script with a carriage return to ensure that the last line is sent.

Table 10–6 lists the substitute variables you can enter for the FTP activity.

Table 10–6 Substitute Variables for the External Process Activity

Variable	Value
${Working.Host}	The host value for the working location.
${Working.User}	The user value for the working location.
${Working.Password}	The password value for the working location.
${Working.Rootpath}	The root path value for the working location.
${Task.Input}	Use this variable when you type in a filename for the script parameter. ${Task.Input} passes the filename you entered for the script parameter. Under parameter list, you could type the following, using the question mark as the separator: `?”-s:${Task.Input}?`

File Exists

Use the File Exists activity to verify the existence of a file before executing the next activity. In the Activities panel, type a string value for the path of the file.

The time out setting for the workflow engine determines how long the File Exist activity waits for a file. If the file does not arrive before the time out setting expires, the File Exists activity terminates with errors.

FORK

Use the FORK activity to launch multiple, concurrent activities after the completion of an activity.

You can assign multiple incoming transitions to a FORK. The FORK is the only activity that enables you to assign multiple unconditional outgoing transitions for parallel execution. For example, in Figure 10–12, the process flow executes the activities named FTP, FDS, and EMAIL in parallel after completing MAP1.

Figure 10–12 FORK Activity Ensures Parallel Execution

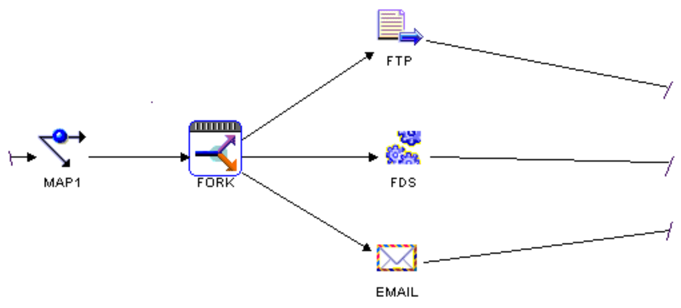
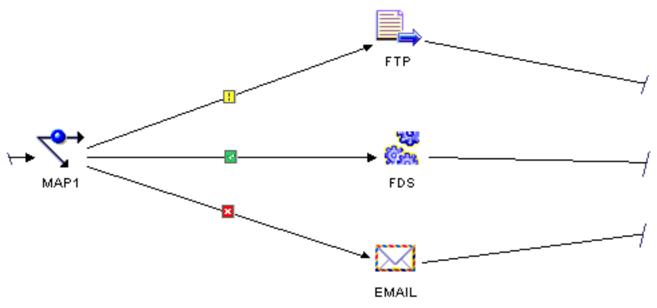


Figure 10–13 shows the same activities without the FORK activity. In this case, only one of the activities runs based on the completion state of MAP1.

Figure 10–13 Absence of FORK Activity Results in Conditional Execution



The Process Flow Editor does not limit you on the number of outgoing transitions or concurrent activities you can assign from a FORK. When designing for concurrent execution, design the FORK based on limitations imposed by the workflow engine or server you use to run the process flow.

FTP

Use the FTP activity to transfer files from one location to another.

You can specify one or more incoming transitions to launch an FTP 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, configure the FTP activity to base its status on it return value. For example, the operating system returns a value of 1 when the FTP command completes successfully. If you set **Use Return as Status** to true on the configuration properties window, the process flow continues along the success transition. For more information on **Use Return as Status,** see"Configuring Process Flows" on page 11-28.

Table 10–7 lists the parameters you set for the FTP activity.

Table 10–7 FTP Activity Parameters

Parameter	Description
Command	The file transfer protocol command such as `c:\WINNT\System32\ftp.exe` .
Parameter List	The list of parameters to be passed to the FTP command such as `/usr/bin/ftp` . 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/\// You can also enter the substitution variables listed in Table 10–8.
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	You can enter a script here or type in a filename for a script. If you type in a filename, use the ${Task.Input} variable in the paramter list to pass the filename. To type 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 10–8 lists the substitute variables you can enter for the FTP activity. The variables related to working and remote locations depend on values you set when you configure the process flow. For more information, see"Configuring Process Flows" on page 11-28.

Table 10–8 Substitute Variables for the FTP Activity

Variable	Value
${Working.Host}	The host value for the working location.
${Working.User}	The user value for the working location.
${Working.Password}	The password value for the working location.
${Working.Rootpath}	The root path value for the working location.
${Remote.Host}	The host value for the remote location.
${Remote.User}	The user value for the remote location.
${Remote.Password}	The password value for the remote location.
${Remote.RootPath}	The root path value for the remote location.

Table 10–8 (Cont.) Substitute Variables for the FTP Activity

Variable	Value
${Task.Input}	Use this variable when you type in a filename for the script parameter. ${Task.Input} passes the filename you entered for the script parameter. Under parameter list, you could type the following, using the question mark as the separator: `?”-s:${Task.Input}?`

Mapping

Use the mapping activity to add an existing mapping that you defined and configured in the Mapping Editor.

You can assign multiple incoming transitions to a mapping activity. For outgoing transitions, assign 1 unconditional transition or up to one of each of the unconditional transitions.

When you add a mapping to a process flow, you can view its configuration properties in the Activities panel. The mapping activity in the Process Flow Editor inherits its properties from the mapping in the Mapping Editor. In the Process Flow Editor, you cannot change a property datatype or direction.

You can, however, assign new values that affect the process flow only and do not change the settings for the mapping in the Mapping Editor. For example, if you change the operating mode from set-based to row-based in the Process Flow Editor, the process flow executes in row-based mode. The original mapping retains set-based mode as its operating mode. If want to change the properties for the underlying mapping, see "Configuring Mappings Reference" on page 11-1.

If a mapping contains a Mapping Input Parameter operator, specify a value according to its datatype. 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.

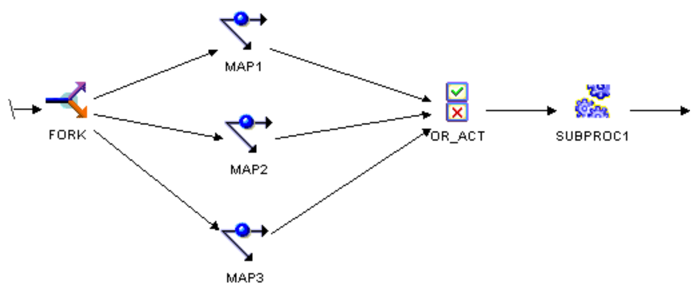
If you want to update a process flow with changes you made to a mapping in the Mapping Editor, delete the mapping activity from the process flow and add the mapping activity again.

OR

Use the OR activity to launch an activity based on the completion of one of a multiple number of upstream activities. You can assign multiple incoming transitions and only one unconditional outgoing transition to an OR activity.

An OR activity in a process flow ensures that downstream activities are triggered only once for each run of a process flow. For example, in Figure 10–14, the SUBPROC1 launches once any one of the three upstream Mapping activities completes.

Figure 10–14 The OR activity in an process flow



The Process Flow Editor enables you to omit the OR activity and assign transitions from each of the three Mapping activities to SUBPROC1 shown in Figure 10–14. However, this logic would launch SUBPROC1 three times within the same run of a process flow. Avoid this by using an OR activity.

Start

START1

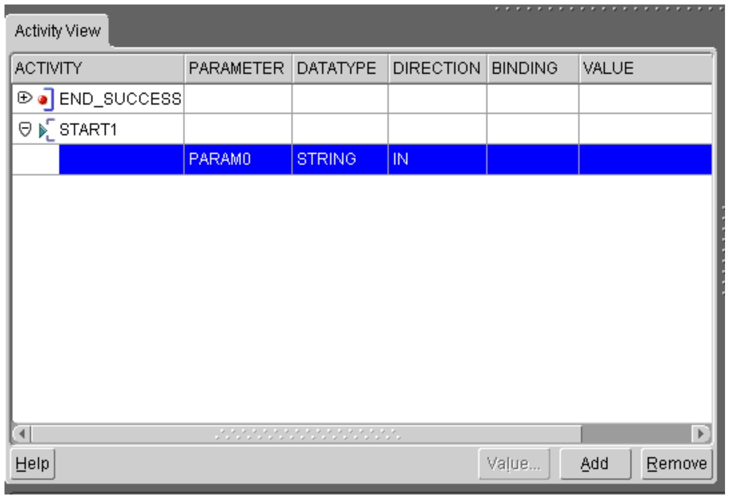
By default, each process flow includes one start activity. You can set input parameters for the start activity that become the input parameters for the complete process flow.

To add parameters to a Start activity:

1. In the Activity Panel on the Process Flow Editor, select the Start activity.
2. Select **Add** at the bottom of the Activity Panel and on top of the indicator bar.

   The Process Flow Editor adds a parameter under the Start activity as shown in Figure 10–15.

Figure 10–15 Parameter added to a Start activity


Activity View

ACTIVITY	PARAMETER	DATATYPE	DIRECTION	BINDING	VALUE
END_SUCCESS					
START1					
	PARAM0	STRING	IN		

3. Set the properties for the parameter.

   Change the parameter name and datatype as necessary. You cannot alter its direction and binding. The direction is IN, indicating that the parameter is an input parameter only. For value, type the parameter value.You can overwrite this value at runtime.
4. You can now use the parameter as input to other activities in the process flow. For instructions, see "Binding and Passing Parameters to Activities" on page 10-8.

Subprocess

Use a subprocess activity to launch a previously created process flow. From one process flow, you can launch any other process flow that is contained within the same process flow package.

Once you add a subprocess to a process flow, use it in your design similar to any other activity. You can assign multiple incoming transitions. For outgoing transitions, assign either 1 unconditional outgoing transition or up to 3 outgoing conditional transitions.

The END activities within the subprocess apply to the Subprocess only and do not function as a termination point in the process flow.

An important difference between a subprocess and other activities is that you can view the contents of a subprocess, but you cannot edit its contents in the parent process flow. To edit a subprocess, open its underlying process flow from the navigation tree. 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 toolbox in the Process Flow Editor, drag and drop the Subprocess activity icon onto the canvas.

   Warehouse Builder displays a dialog to select and add a process flow as a subprocess.

2. Expand the process flow module and select a process flow from the same process flow package as the parent process flow.

   Warehouse Builder displays the process flow as a subprocess activity on the parent process flow.

3. To view the contents of the subprocess, right-click the subprocess and select **Expand Node.**

   The Process Flow Editor displays the graph for the subprocess surrounded by a blue border.

Transform

When you add transformations from the Warehouse Builder transformation library to a process flow using the Transform activity, the Process Flow Editor displays the parameters for the transformation in the Activity panel.

You can specify one or more incoming transitions to launch a transform activity. For outgoing transitions, you can either specify one unconditional transition or one of each of the three conditional transitions.

If you specify conditional outgoing transitions, you can configure the activity to base its status on its return value. For more information on **Use Return as Status,** see"Configuring Process Flows" on page 11-28.

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.

Process Flow Editor Menu and Toolbar

This section contains reference information on the following design components in the Process Flow Editor:

- Menu Bar
- Toolbar

Menu Bar

Use the following menu options for designing a process flow within the Process Flow Editor:

- Process Flow
- Edit
- View
- Window
- Help

Process Flow

Table 10–9 lists the options available under **Process Flow** on the menu bar.

Table 10–9 Menu Bar: Process Flow

Menu Item	Description
Open	Opens another process flow editor within the current warehouse module. If you try to open a process flow that is already open, the Process Flow Editor switches to that process flow.
Validate	Validates the current process flow and all activities. After the validation completes, Warehouse Builder displays the Validation Details Dialog.
Generate	Use to display a sample of the code that will be generated for the current process flow. Warehouse Builder displays the generated script in the Code View tab.
Print Setup	Use to set printing options including scale, margins, and multi-page options.
Print Preview	Displays the print preview for the process flow. You can change the zoom and print setup properties.
Print	Prints the contents of the process flow canvas.
Save As	Saves the process flow diagram to the location you specify. You can use the diagram in documents or web pages to enhance reports and documentation. You can save mapping diagrams either as JPEG (.jpg) or Scalable Vector Graphics (.svg) files.
Properties	Launches the properties window for the process flow.
Close Window	Closes the Process Flow Editor.

Edit

Table 10–10 lists the options available under **Edit** on the menu bar.

Table 10–10 Menu Bar: Edit

Menu Item	Description
Select Mode	The select mode is the default mode when you launch the Process Flow Editor. Use the select mode when you select and move activities on the canvas.
Create transition	Use the create transition mode when you connect activities on the canvas or select existing transitions.
Panning	Use panning to move the objects on the canvas horizontally and vertically. When you select the panning option from the **Edit** menu, the cursor displays as a hand. While pressing the left mouse button, move the mouse horizontally and vertically to move the objects on the canvas.
Interactive zoom	Use interactive zoom to decrease or enlarge the size of the objects on the canvas. When you select the interactive zoom option from the **Edit** menu, the cursor displays as a magnifying glass with an up and down arrow at its center. To reduce the objects on the canvas, hold down the left mouse button and move the mouse upward. To enlarge the objects on the canvas, hold down the left mouse button and move the mouse downward. Release the left mouse button when the objects display at the desired size.
Rename	Launches the Rename dialog to rename the object selected on the process flow canvas. This menu option is enabled for activities.
Delete	Deletes the selected object from the process flow canvas. This menu option is enabled for activities and transitions.

View

Table 10–11 lists the options available under **View** on the menu bar.

Table 10–11 Menu Bar: View

Menu Item	Description
Zoom	Zooms the canvas to the selected level of magnification. When you select zoom from the **View** menu, you can select from the following magnification levels: 200%, 150%, 100%, 75%, 50%, and 25%.
Fit in Window	Resizes all objects to fit on the canvas.
Auto Layout	Resizes and rearranges all the objects on the canvas based on a default layout for process flows. Use auto layout to display objects in sequential order, from left to right. The start activity displays on the left and end activities on the right.
Center	Centers the objects on the canvas.
Expand Subprocess	Displays the graph for the subprocess on the canvas. The subprocess graph is enclosed by a blue border.
View Subprocess	Displays the subprocess in the Process Flow Editor.
Return to parent	After using the view subprocess command, use return to parent to activate the Process Flow Editor for the parent process flow.
View Validation Messages	Launches the Validation Status dialog.

Window

Table 10–12 lists the options available under **Window** on the menu bar.

Table 10–12 Menu Bar: Window

Menu Item	Description
Show/Hide Birds Eye View	Use to display or remove the Birds Eye View on the canvas.
Show/Hide Activities Panel	Use to display or remove the activity panel on the canvas. When you select to show the activity panel, Warehouse Builder displays the activity panel in the lower left corner of the Process Flow Editor.
Show/Hide Transition Panel	Use to display or remove the transition panel on the canvas. When you select to show the transition panel, Warehouse Builder displays the transition panel in the lower left corner of the Process Flow Editor.
Arrange All	Arranges all the viewers in the process flow editor.

Help

Table 10–13 lists the options available under **Help** on the menu bar.

Table 10–13 .Menu Bar: Help

Menu Item	Description
Contents	Launches the online help viewer with the “Contents” tab selected. Expand the content tree to view the table of contents.
Index	Launches the online help viewer with the Index tab selected.
Search	Launches the online help with the Search tab selected.
Topics	Launches the online help viewer for the selected topic.
About Oracle Warehouse Builder	Launches the About Warehouse Builder dialog.

Toolbar

Table 10–14 lists the buttons on the Process Flow Editor tool bar in order of appearance.

Table 10–14 Process Flow Editor Toolbar

Button	Description
Toggle Tool Palette	Toggles the palette display on and off.
Print	Prints the contents of the editor.
Save As	Saves the process flow diagram to the location you specify. You can use the diagram in documents or web pages to enhance reports and documentation. You can save mapping diagrams either as JPEG (.jpg) or Scalable Vector Graphics (.svg) files.
Validate	Validates the current process flow. After the validation completes, Warehouse Builder displays the Validation Results dialog.
Synchronize	Synchronizes the process flow with any updates to the repository since you opened the process flow. Other users may have made changes that affect the current process flow. This will check for changes and make the necessary updates.
Select mode	The select mode is the default mode when you launch the Process Flow Editor. Use the select mode to select objects on the canvas. When you select an activity, the editor displays a blue border around the activity and you can edit, move, or delete the activity. When you select a transition, the editor changes the arrow from black to blue and you can edit the transition.

Table 10–14 (Cont.) Process Flow Editor Toolbar

Button	Description
Create Transition	Use the create transition mode to connect activities on the canvas.
Panning	Use panning to move the objects on the canvas horizontally and vertically. When you select the panning option from the **Edit** menu, the cursor displays as a hand. While pressing the left mouse button, move the mouse horizontally and vertically to move the objects on the canvas.
Interactive zoom	Use interactive zoom to decrease or enlarge the size of the objects on the canvas. When you select the interactive zoom option from the **Edit** menu, the cursor displays as a magnifying glass with an up and down arrow at its center. To reduce the objects on the canvas, hold down the left mouse button and move the mouse upward. To enlarge the objects on the canvas, hold down the left mouse button and move the mouse downward. Release the left mouse button when the objects display at the desired size.
Zoom in	The zoom in option appears as a magnifying glass with a plus sign (+) at its center. Click the icon to increase the magnification of the objects on the canvas.
Zoom out	The zoom out option appears as a magnifying glass with a negative sign (-) at its center. Click the icon to decrease the magnification of the objects on the canvas.
Fit in window	Resizes all objects to fit on the canvas.
Auto layout	Resizes and rearranges all the objects on the canvas based on a default layout for process flows. Use auto layout to display objects in sequential order, from left to right. The start activity displays on the left and end activities on the right.
Birds eye view	Displays the Birds Eye View on the canvas.
Expand subprocess	Displays the graph for the subprocess on the canvas. The subprocess graph is enclosed by a blue border.
Visit child process	Launches the Process Flow Editor for the subprocess.
Return to parent	After using the view subprocess command, use return to parent to activate the Process Flow Editor for the parent process flow.
Properties	Launches the Process Flow Properties dialog.
Help	Invokes the online help set.

11

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 and recommendations for assigning physical properties to mappings and process flows.

This chapter includes:

Configuring Mappings Reference

When you correctly configure mappings, you can improve the ETL performance. Use this section as a reference for setting configuration parameters that govern how Warehouse Builder loads data and optimizes code for better performance.

This section includes the following topics:

Procedure for Configuring Mappings

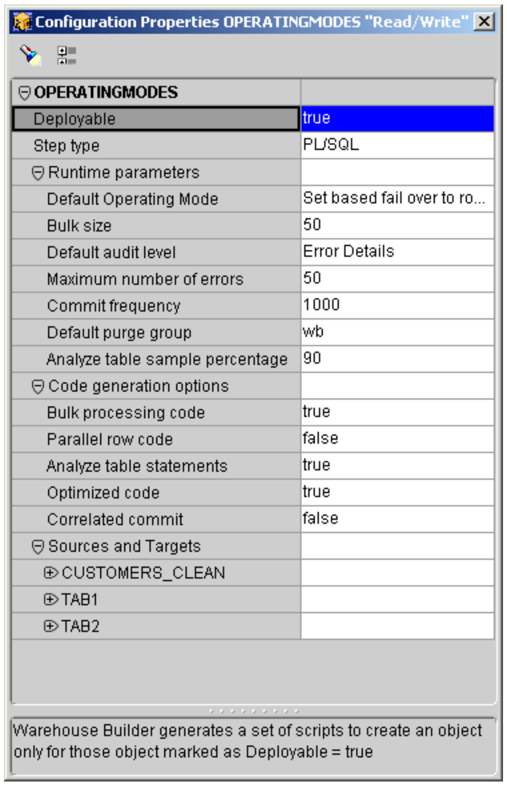
To configure physical properties for a mapping:

1. Select a mapping from the navigation tree, select **Object** from the menu bar, and select **Configure.**

   Or, right-click the mapping you want to configure and select **Configure** from the pop-up menu.

   Warehouse Builder displays the configuration Properties dialog for a mapping as shown in Figure 11–1.

Figure 11–1 Configuration Properties dialog For Mapping Tables



⊖ OPERATINGMODES	
Deployable	true
Step type	PL/SQL
⊖ Runtime parameters	
Default Operating Mode	Set based fail over to ro...
Bulk size	50
Default audit level	Error Details
Maximum number of errors	50
Commit frequency	1000
Default purge group	wb
Analyze table sample percentage	90
⊖ Code generation options	
Bulk processing code	true
Parallel row code	false
Analyze table statements	true
Optimized code	true
Correlated commit	false
⊖ Sources and Targets	
⊕ CUSTOMERS_CLEAN	
⊕ TAB1	
⊕ TAB2	

Warehouse Builder generates a set of scripts to create an object only for those object marked as Deployable = true

2. Set **Deployable** to true to enable Warehouse Builder to generate a set of scripts for mapping entities marked as deployable.

   The default setting is true. If you set deployable to false for a mapping, Warehouse Builder does not generate scripts for that mapping.

3. Set **Step Type** 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. Warehouse Builder sets the correct value depending on the mapping: PL/SQL, SQL*Loader, ABAP (for an SAP source mapping).

4. Expand the **Runtime Parameters Reference** to configure your mapping for deployment.

   For a description of each runtime parameter, see **"Runtime Parameters Reference"** on page 11-3.

5. Expand **Code Generation Options Reference** to enable performance options that optimize the code generated for the mapping.

   For a description of each option, see "Code Generation Options Reference" on page 11-4.

6. Expand **Sources and Targets Reference** to set the physical properties of the operators in the mapping.

   The Configuration Properties dialog displays all source and target operators in a mapping under **Sources and Targets.** Each operator contains a set of properties you can edit. For information on configuring sources and targets in a mapping, see "Sources and Targets Reference" on page 11-6.

7. If the mapping contains a flat file as a source or target, configure additional parameters. For information on configuring mappings with flat file sources and targets, see "Configuring Flat File Operators" on page 11-9.

Runtime Parameters Reference

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 Deployment Manager, the Process Flow Editor, or Oracle Enterprise Manager.

The Runtime Parameters include the following parameters:

- Default Operating Mode
- Bulk Size
- Default Audit Level
- Maximum Number of Errors
- Commit Frequency
- Default Purge Group
- Analyze Table Sample Percentage

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 "Selecting a Default Operating Mode" on page 11-13. You can select one of the following operating modes:

- **Set based:** Warehouse Builder generates a single SQL statement that inserts all data and performs all operations on the data. This increases the speed of Data Manipulation Language (DML) operations. Set based mode offers optimal performance but minimal auditing details.
- **Row based:** Warehouse Builder generates statements that process data row by row. The select statement is a SQL cursor. All subsequent statements are PL/SQL. Since data is processed row by row, the row based operating mode has the slowest performance but offers the most auditing details.
- **Row based (Target Only):** Warehouse Builder generates a cursor select statement and attempts to include as many operations as possible in the cursor. For each target, Warehouse Builder generates a PL/SQL insert statement and inserts each row into the target separately.
- **Set based fail over row based:** Warehouse Builder executes the mapping in set based mode. If an error occurs, the execution fails and Warehouse Builder starts the mapping over again in the row based mode.
- **Set based fail over row based (Target Only):** The mapping is first executed in set based mode. If any error occurs, the execution fails over to **Row based (Target Only)** mode.

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 Generate Bulk Processing Mode is set to true. For more information, see the *Oracle PL/SQL Reference Guide.*

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.

Maximum Number of Errors

Use **Maximum Number of Errors** to indicate the maximum number of errors allowed when executing the package. Execution of the package terminates when the number of errors reached is greater than the maximum number of errors value.

Commit Frequency

Commit frequency only applies to non-bulk mode mappings. Bulk mode mappings commit according to the bulk size.

When you set the **Default Operating Mode** to row based and Bulk Processing Code to false**,** Warehouse Builder uses the **Commit Frequency** parameter when executing the package. Warehouse Builder commits data to the database after processing the number of rows specified in this parameter.

If you set Bulk Processing Code to true, set the Commit Frequency equal to the Bulk Size. If the two values are different, Bulk Size overrides the commit frequency and Warehouse Builder implicitly performs a commit for every bulk size.

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.

Analyze Table Sample Percentage

When you set Analyze Table Statements to true, 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.

Code Generation Options Reference

The Code Generation Options include the following:

- Bulk Processing Code
- Row Based Mode
- Parallel Row Code
- Analyze Table Statements
- Optimized Code
- Correlated Commit

Bulk Processing Code

If this configuration parameter is set to true, 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.*

Row Based Mode

Row based mode is an operating mode option. Row based mode allows for the maximum amount of runtime auditing. A mapping that runs in row-based mode processes data from a source record by record.

Parallel Row Code

Set this parameter to true to generate code using the table function feature supported in Oracle Database. This setting improves performance by enabling table functions to use parallel execution in the Oracle Server for Row based and Row based (Target only) operating modes.

You can set this parameter to true when the mapping generates PL/SQL code and is executed by Oracle Database server.

If the mapping generates SQL* Loader code, set the parameter to **false** (default). This property is only available for mappings that will generate PL/SQL code. It will not be available as a configuration property for mappings that will generate SQL* Loader code. This property is not available for set-based mode, but for set-based-fail-over-rowbased and set-based-fail-over-rowbased-target modes, this property is available if the set based mode fails and the code executes in row-based/row-based-target mode. This property is not available for mappings that include a Mapping Output Parameter, a Flat File operator, or a Sequence operator as a source.

Scenarios when parallelism is lost for table functions:

1. When the input cursor refers to objects (such as tables or views) in remote schemas. For example, a mapping containing a table operator bound to a table in a remote database schema cannot be executed in parallel.
2. When the input cursor passed to table function refers to objects in non-Oracle databases. Oracle database server parallel engine is unable to extract parallelism when the objects are accessed using an Oracle gateway. For example, when a mapping contains a table operator that is bounded to a table in a DB2 database, the mapping will not be executed in parallel.
3. Parallel DML operation cannot be performed on tables with triggers. For example, a mapping with a table and a pre-map or post-map triggers that operate on the table.
4. Updates and deletes can only be parallelized on partitioned tables. Update and delete parallelism is not possible within a partition or on a non-partitioned table. For example, if a mapping contains a table operator with a delete/update load, the target table must be partitioned.

Analyze Table Statements

If you set this parameter to true, Warehouse Builder generates code for analyzing the target table after the target is loaded if the resulting target table only is double or half its original size.

Optimized Code

Set this parameter to true to improve performance for mappings that include the Splitter operator and inserts into multiple target tables. When this parameter is set to true and the mapping is executed by Oracle Database, Warehouse Builder generates a single SQL statement (`multi_table_insert`) that inserts data into multiple tables based on same set of source data.

Warehouse Builder performs the multiple table insert only if this parameter is set to true and the Oracle target module database is Oracle Database. The multiple tables insert is performed only for mappings in set based mode that include a Splitter operator and no active operators such as an Aggregator or Joiner operator between the Splitter and the target. Also, the multiple insert is available only for tables, not views, materialized views, dimensions, or cubes. Each target table must have fewer than 999 columns. For detailed instructions on how to create a mapping with multiple targets, see "Example: Following a Record Through the Name and Address Operator" on page 8-54.

Set this parameter to false for mappings run in row based mode or for mappings executed by Oracle*8i* server. You may also want to set this parameter to false when auditing is required. When set to true, Warehouse Builder returns one total SELECT and INSERT count for all targets.

Correlated Commit

This applies to mappings with multiple targets. When set to true, Warehouse Builder commits data based on the selected records. When set to false, Warehouse Builder commits data based on target by target processing (insert, update, delete).

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 11-16.

Sources and Targets Reference

The Configuration Properties dialog displays all source and target operators in a mapping under the **Sources and Targets.** The properties that are available for configuration vary according to the types of source and target operators used in the mapping.

For relational and dimensional sources and targets such as tables, views, and cubes, Warehouse Builder displays the following set of properties for each operator:

- Bound Name
- Schema
- Database Link
- Partition Exchange Loading
- Hints
- Constraint Management
- SQL*Loader Parameters

Bound Name

When Warehouse Builder generates code, it identifies each source and target operator by its bound name. By default, the bound name is the same name as the operator name.

Schema

This parameter maintained for backward compatibility only.

In previous version of Warehouse Builder, you could link the mapping to a particular schema by clicking on the Schema field and typing a name.

Database Link

This parameter maintained for backward compatibility only.

In previous version of Warehouse Builder, you could select a database link by name from the drop-down list. Source operators can be configured for schemas and database links, but targets can be configured for schemas only. Sources and targets can reside in different schemas, but they must reside in the same database instance.

Partition Exchange Loading

Set this parameter to true to enable partition exchange loading (PEL) into a target table. For information on how to design mappings for PEL, see "Using Partition Exchange Loading" on page 11-21.

Hints

Define extraction or loading hints. Application developers often develop insights into their data. For example, they know that a query runs much faster if a set of tables is joined in one order rather than another. Warehouse Builder can incorporate these insights into the generated SQL code packages as SQL Optimizer Hints.

When you select a hint from the Hints dialog, the hint appears in the Existing Hints field. Type additional text as appropriate in the Extra Text column. The editor includes the hint in the mapping definition as is. There is no validation or checking on this text.

For information on optimizer hints and how to use them, see *Oracle Database Designing and Tuning for Performance.*

Constraint Management

Configure the following Constraint Management parameters:

- **Enable Constraints:** If this property is set to true, Warehouse Builder maintains the foreign key constraints on the target table before loading data. If this property is set to false, Warehouse Builder disables the foreign key constraints on target tables before data loading and then re-enables the constraints after loading. Constraint violations found during re-enable are identified in the runtime audit error table and, if specified, in an exceptions table.

  When you disable constraints, loading time is decreased since constraint checking is not performed. However, if exceptions occur for any rows during re-enabling, the constraints for those rows will remain in a non-validated state. These rows are logged in the runtime audit error table by their ROWID. You must manually inspect the error rows to take any necessary corrective action.

  Setting the Enable Constraints to false is subject to the following restrictions:

  – For set based operating mode, the false setting disables foreign key constraints on the targets before loading, and then re-enables the constraints after loading. This property has no effect on foreign key constraints on other tables referencing the target table. If the load is done using SQL*Loader instead of a SQL or PL/SQL package, then a re-enable clause is added to the .ctl file.

  - For set based fail over to row based and set based fail over to row based (target only) operating modes, the false setting disables the foreign key constraints on the targets before loading and then re-enables them if the load succeeds in set based mode. This setting has no effect on foreign keys referencing other tables. If the load fails over to row-based, then loading will repeat in row based mode and all constraints remain enabled.

    Note: Constraint violations created during re-enabling will not cause the load to fail from set based over to row based mode.

  - For row based or row based (target only) operating modes, all foreign key constraints remain enabled even if the property is set to false.
  - For the TRUNCATE/INSERT DML type, the false setting disables foreign key constraints on other tables referencing the target table before loading, and then re-enables the constraints after loading, regardless of the default operating mode.

- **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.

SQL*Loader Parameters

Configure the following SQL*Parameters properties:

- **Partition Name:** Indicates that the load is a partition-level load. Partition-level loading enables you to load one or more specified partitions or subpartitions in a table. Full database, user, and transportable tablespace mode loading does not support partition-level loading. Because incremental loading (incremental, cumulative, and complete) can be done only in full database mode, partition-level loading cannot be specified for incremental loads. In all modes, partitioned data is loaded in a format such that partitions or subpartitions can be selectively loaded.
- **Sorted Indexes Clause:** Identifies the indexes on which the data is presorted. This clause is allowed only for direct path loads. Because data sorted for one index is not usually in the right order for another index, you specify only one index in the SORTED INDEXES clause. When the data is in the same order for multiple indexes, all indexes can be specified at once. All indexes listed in the SORTED INDEXES clause must be created before you start the direct path load.
- **Singlerow:** Intended for use during a direct path load with APPEND on systems with limited memory, or when loading a small number of records into a large table. This option inserts each index entry directly into the index, one record at a time. By default, SQL*Loader does not use SINGLEROW to append records to a table. Index entries are stored in a temporary area and merged with the original index at the end of the load. Although this method achieves better performance and produces an optimal index, it requires extra storage space. During the merge, the original index, the new index, and the space for new entries all simultaneously occupy storage space. With the SINGLEROW option, storage space is not required

for new index entries or for a new index. Although the resulting index may not be as optimal as a freshly sorted one, it takes less space to produce. It also takes more time because additional UNDO information is generated for each index insert. This option is recommended when the available storage is limited. It is also recommended when the number of records to be loaded is small compared to the size of the table. A ratio of 1:20 or less is considered small.

- **Trailing Nullcols:** Sets SQL*Loader to treat any relatively positioned columns that are not present in the record as null columns.
- **Records To Skip:** Invokes the SKIP command in SQL*Loader. SKIP specifies the number of logical records from the beginning of the file that should not be loaded. By default, no records are skipped. This parameter continues loads that have been interrupted for some reason. It is used for all conventional loads, for single-table direct loads, and for multiple-table direct loads when the same number of records are loaded into each table. It is not used for multiple-table direct loads when a different number of records are loaded into each table.
- **Database File Name:** Specifies the names of the export files to import. The default extension is .dmp. Because you can export multiple export files, you may need to specify multiple filenames to be imported. You must have read access to the imported files. You must also have the IMP_FULL_DATABASE role.

Configuring Flat File Operators

The Configuration Properties dialog contains additional settings for Mapping Flat File operators, depending on how the operators are used in the mapping.

- **Flat File Operators as a Target:** Warehouse Builder generates a PL/SQL deployment code package. For information on configuring the parameters associated with a Mapping Flat File operator used as a target, see "Flat File Operators as a Target" on page 11-9.
- **Flat File Operator as a Source:** Warehouse Builder generates SQL*Loader scripts. For information on the parameters associated with a Mapping Flat File operator used as a source, see "Flat File Operator as a Source" on page 11-11.

Flat File Operators as a Target

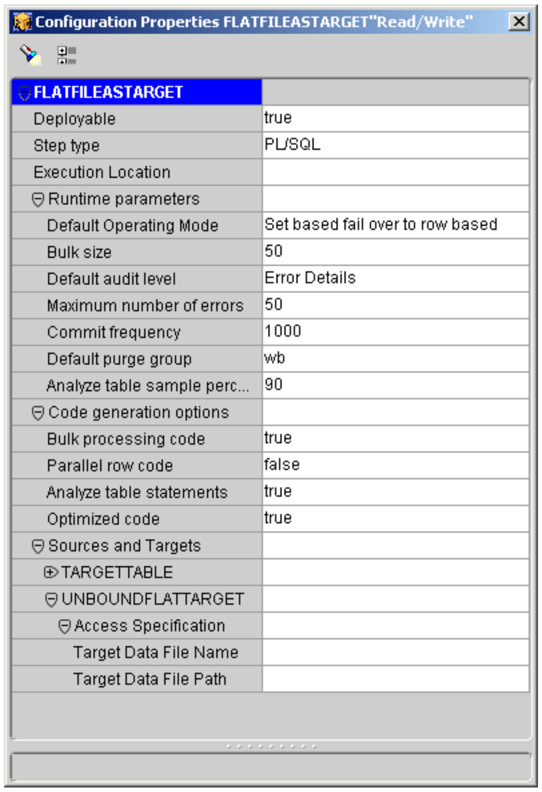
To configure properties unique to mappings with flat file targets:

1. Select a mapping from the navigation tree, select **Object** from the menu bar, and select **Configure.**

   Or, right-click the mapping you want to configure and select **Configure** from the pop-up menu.

   Warehouse Builder displays the Configuration Properties dialog as shown in Figure 11–2.

Figure 11–2 Configuration Properties dialog (Flat File Target)



Configuration Properties FLATFILEASTARGET"Read/Write"

FLATFILEASTARGET	
Deployable	true
Step type	PL/SQL
Execution Location	
Runtime parameters	
Default Operating Mode	Set based fail over to row based
Bulk size	50
Default audit level	Error Details
Maximum number of errors	50
Commit frequency	1000
Default purge group	wb
Analyze table sample perc...	90
Code generation options	
Bulk processing code	true
Parallel row code	false
Analyze table statements	true
Optimized code	true
Sources and Targets	
TARGETTABLE	
UNBOUNDFLATTARGET	
Access Specification	
Target Data File Name	
Target Data File Path	

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 **...** to display another properties dialog.

3. Set **Deployable** to true to enable Warehouse Builder to generate a set of scripts for mapping objects marked as deployable. The default setting is true. If you set deployable to false for a mapping, Warehouse Builder does not generate scripts for that mapping.

4. Set **Step Type** 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 Reference**, 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 Reference" on page 11-3.

7. Set the **Code Generation Options Reference** as described in "Code Generation Options Reference" on page 11-4.

8. Set the Sources and Targets Reference as described in "Sources and Targets Reference" on page 11-6.

9. For **Access Specification,** specify the name of the flat file target in **Target Data File Name.** For the **Target Data File Location,** specify a target file located on the machine where you installed the Warehouse Builder Runtime Platform.

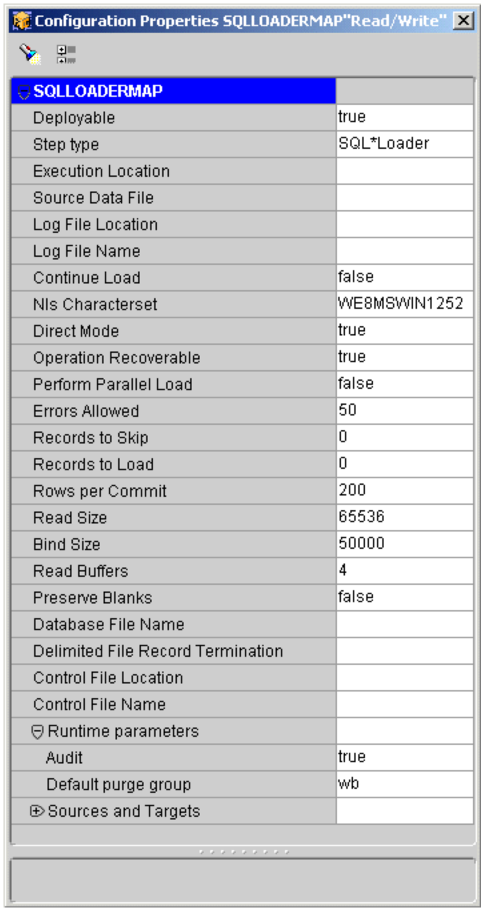
Flat File Operator as a Source

To configure a mapping with a flat file operator as a source:

1. Select a mapping from the navigation tree, select **Object** from the menu bar, and select **Configure.** Or, right-click the mapping you want to configure and select **Configure** from the pop-up menu.

   The Configuration Properties dialog appears as shown in Figure 11–3.

Figure 11–3 Configuration Properties Dialog (Flat File Source)



SQLLOADERMAP	
Deployable	true
Step type	SQL*Loader
Execution Location	
Source Data File	
Log File Location	
Log File Name	
Continue Load	false
Nls Characterset	WE8MSWIN1252
Direct Mode	true
Operation Recoverable	true
Perform Parallel Load	false
Errors Allowed	50
Records to Skip	0
Records to Load	0
Rows per Commit	200
Read Size	65536
Bind Size	50000
Read Buffers	4
Preserve Blanks	false
Database File Name	
Delimited File Record Termination	
Control File Location	
Control File Name	
Runtime parameters	
Audit	true
Default purge group	wb
Sources and Targets	

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 **...** to display another properties dialog.

3. Set **Deployable** to true to a generate SQL*Loader script.

4. Specify the **Execution Location,** the location where the mapping is executed.

5. For the **Source Data File,** enter the full physical filename of the data file to be loaded.

   Type a complete path name for the file in the syntax of the operating system on which the SQL*Loader script is deployed. The value you provide is placed in the INFILE clause. If you enter a value enclosed in single quote, the generated code will be LOAD. CONTINUE_LOAD is used when a direct load of multiple tables is discontinued and needs to be restarted. It is used in conjunction with the operator-level SKIP option.

6. Specify the **Log File Location** and **Log File Name.**
7. Set **Continue Load.**

   If SQL*Loader runs out of space for data rows or index entries, the load is discontinued. If **Continue Load** is set to true, Warehouse Builder attempts to continue discontinued loads.
8. In **Nls Characterset,** specify the character set to place in the CHARACTERSET clause.
9. In **Direct Mode,** specify the value of the DIRECT option as either TRUE or FALSE. True indicates that a direct path load will be done. False indicates that a conventional load will be done. In general, direct mode is faster.
10. In **Operation Recoverable,** select true to indicate that the load is recoverable. Select False to indicate that the load is not recoverable and records are not recorded in the redo log. This parameter controls the output of the RECOVERABLE clause.
11. Configure the following parameters that affect the OPTIONS clause in the SQL *Loader scripts Warehouse Builder generates for mappings with flat file sources.

    Perform Parallel Load: Specifies the value of the PARALLEL option as either =TRUE or =FALSE. True indicates that direct loads can operate in multiple concurrent sessions.

    Errors Allowed: If the value specified is greater than 0, then the ERRORS = n option is generated. SQL*Loader terminates the load at the first consistent point after it reaches this error limit.

    Records To Skip: If the value specified is greater than 0, then the SKIP = n option is generated. This value indicates the number of records from the beginning of the file that should not be loaded. If the value is not specified, no records are skipped.

    Records To Load: If the value specified is greater than 0, then the LOAD = n option will be generated. This value specifies the maximum number of records to load. If a value is not specified all of the records are loaded.

    Rows Per Commit: If the value specified is greater than 0, then the ROWS = n option is generated. For direct path loads, the value identifies the number of rows to read from the source before a data is saved. For conventional path loads, the value specifies the number of rows in the bind array.

    Read Size: If the value specified is greater than 0, then the READSIZE = n option is generated. The value is used to specify the size of the read buffer.

    Bind Size: If the value specified is greater than 0, then the BINDSIZE = n option is generated. The value indicates the maximum size in bytes of the bind array.

    Read Buffers: If the value specified is greater than 0, then the READBUFFERS n clause is generated. READBUFFERS specifies the number of buffers to use during a direct path load. Do not specify a value for READBUFFERS unless it becomes necessary.

    Preserve Blanks: If this parameter is set to TRUE, then the PRESERVE BLANKS clause is generated. PRESERVE BLANKS retains leading white space when optional enclosure delimiters are not present. It also leaves the trailing white space intact when fields are specified with a predetermined size.

Database File Name: This parameter enables you to specify the characteristics of the physical files to be loaded. The initial values for these parameters are set from the properties of the flat file used in the mapping.

If this parameter is set to a non-blank value, then the FILE= option is generated. The value specified is enclosed in single quotes in the generated code.

Control File Location and **Control File Name:** The control file name necessary for audit details.

For more information on each SQL*Loader option and clause, see Oracle Database *Database Utilities*.

12. Expand the **Runtime Parameters Reference** to configure your mapping for deployment.

    Audit: Perform audit when the step 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.

13. 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 11-6.

Strategies for Configuring PL/SQL Mappings

This section discusses various strategies for configuring mappings. Use this section to determine which configuration settings are appropriate for a given scenario.

This section includes the following topics:

- "Selecting a Default Operating Mode" on page 11-13
- "Committing Data from a Single Source to Multiple Targets" on page 11-16
- "Avoiding Invalid Designs for PL/SQL Mappings" on page 11-18

Selecting a Default Operating Mode

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 unselected modes. Therefore, at runtime, you can select to run in the default operating mode or any one of the other valid operating modes.

The types of operators in the mapping may limit the operating modes you can select. As a general rule, mappings run in set based mode can include any of the operators except for Match-Merge, Name-Address, and Transformations used as procedures. Although you can include any of the operators in row based and row based (target only) modes, there are important restrictions on how you use SQL based operators such as Aggregators, Joins, and Key Lookups. To use SQL based operators in either of the row based modes, ensure that the operation associated with the operator can be included in the cursor.

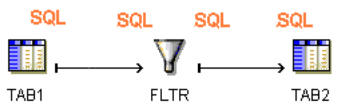
These general rules are explained in the following sections. For more information on how to use operators, see "Avoiding Invalid Designs for PL/SQL Mappings" on page 11-18.

Set based

In set based mode, Warehouse Builder generates a single SQL statement that processes all data and performs all operations. Although processing data as a set improves performance, the auditing information available is limited. Runtime Auditing is limited to reporting to the execution error only. In set based mode, you cannot view details on which rows contain errors.

Table 11–4 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 11–4 Simple Mapping Run in Set Based Mode



To correctly design a mapping for the set based mode, avoid operators that require row by row processing such as Match-Merge and Name-Address operators. If you include an operator in the dataflow that cannot be performed in SQL, Warehouse Builder does not generate set based code and issues an error when you execute the package in set based mode.

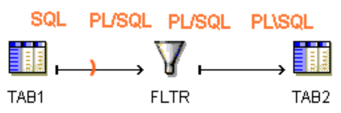
For target operators in a mapping, set their loading types to either INSERT/UPDATE or UPDATE/INSERT. Warehouse Builder does not support UPDATE or DELETE loading in set based mode. For a complete listing of how Warehouse Builder handles operators in set based mappings, see Table 11–3 on page 11-20.

Row based

In row based mode, Warehouse Builder generates statements that process data row by row. The select statement is in a SQL cursor. All subsequent statements are PL/SQL. You can access full runtime auditing information for all operators performed in PL/SQL and only limited information for operations performed in the cursor.

Table 11–5 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 11–5 Simple Mapping Run in Row Based Mode



If the mapping includes any SQL based operators, which cannot be performed in PL/SQL, Warehouse Builder attempts to generate code with those operations in the cursor. To generate valid row based code, design your mapping such that , if you include any of the following SQL based operators, Warehouse Builder can include the operations in the cursor:

- Aggregation
- Deduplicator
- Join
- Key Lookup
- Sequence
- Set
- Sorter

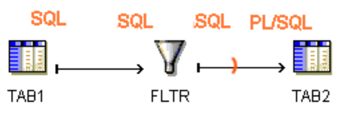
In order for the preceding operators to be included in the cursor, do not directly precede it by an operator that generates PL/SQL code. In other words, you cannot run the mapping in row-based mode if it contains a Transformation implemented as procedure, a Flat File used as a source, a Match-Merge, or Name-Address operator directly followed by any of the seven SQL based operators. For the design to be valid, include a staging table between the PL/SQL generating operator and the SQL based operator. For more information about how to design mappings, see "Avoiding Invalid Designs for PL/SQL Mappings" on page 11-18.

Row based (Target Only)

In row based (Target Only) mode, Warehouse Builder generates a cursor select statement and attempts to include as many operations as possible in the cursor. For each target, Warehouse Builder inserts each row into the target separately. You can access full runtime auditing information for all operators performed in PL/SQL and only limited information for operations performed in the cursor. Use this mode when you expect fast set based operations to extract and transform the data but need extended auditing for the loading the data, which is where errors are likely to occur.

Table 11–6 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 11–6 Simple Mapping Run in Row Based (Target Only) Mode



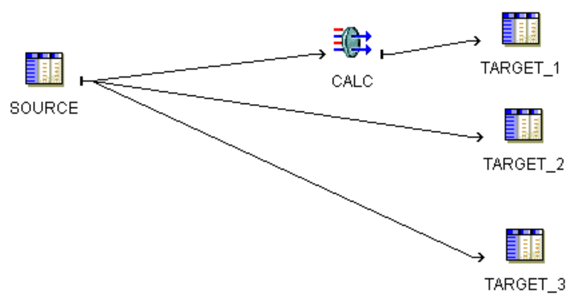
Row based (target only) places the same restrictions on SQL based operators as the row based operating mode. Additionally, for mappings with multiple targets, Warehouse Builder generates code with a cursor for each target.

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 is represented correctly across all targets.

Figure 11–7 shows a mapping that illustrates this case. The target tables all depend upon the source table. For every row added to TARGET_1, one row should be added to the other targets. If this relationship is not maintained when you reload data, the data becomes inaccurate and possibly unusable.

Figure 11–7 Mapping with Multiple Targets Dependent on One Source



If the number of rows from SOURCE is relatively small, maintaining the three targets may not be difficult. Manually maintaining targets dependent on a common source, however, becomes more tedious as you increase the number of rows from the source, or as you design more complex mappings with more targets and transformations.

To use Warehouse Builder to ensure that every row in the source is properly represented in every target, configure the mapping to use a correlated commit strategy.

Commit Strategies

When you assign a commit strategy for a mapping in Warehouse Builder, you determine the commit and rollback behaviors for the multiple targets dependent on a single source. You can choose between the following strategies:

- **Independent Commit:** Warehouse Builder commits and rolls back each target separately and independently of other targets. Use the independent commit when the consequences of targets being loaded unequally are not great or are irrelevant.
- **Correlated Commit**: 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.

The combination of the commit strategy and operating mode determines mapping behavior. Table 11–1 shows the valid combinations you can select.

Table 11–1 Valid Commit Strategies for Operating Modes

Operating Mode	Correlated Commit	Independent Commit
Set based	Valid	Valid

Table 11–1 (Cont.) Valid Commit Strategies for Operating Modes

Operating Mode	Correlated Commit	Independent Commit
Row based	Valid	Valid
Row based (target only)	Not Recommended	Valid

Correlated commit is not recommended 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, in most cases, does not perform the correlated commit.

To understand the affects each operating mode and commit strategy combination has on a mapping, consider the mapping from Figure 11–7 on page 11-16. 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/ Independent 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/Independent 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.

Using the Correlated Commit Strategy

Correlated commit may impact the size of your rollback segments. Space for rollback segments may be a concern when you merge data (insert/update or updated/insert) in set based mode.

Correlated commit operates transparently with PL/SQL bulk processing code.

When you use Correlated Commit to ensure all source rows are represented in multiple targets, consider using the **Set based fail over to row based** operating mode. In this way, you can find a balance between performance and error handling.

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.

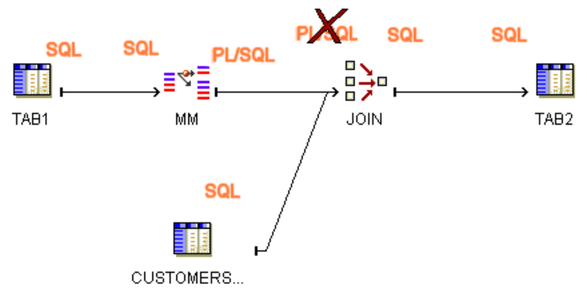
Avoiding Invalid Designs for PL/SQL Mappings

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 implementable 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 11–8 is invalid because the Match-Merge operator MM generates PL/SQL output but the subsequent Join operator accepts SQL input only.

Figure 11–8 Mapping Violates Input Requirement for Join Operator



To achieve the desired results for the mapping, consider joining the source tables before performing the Match-Merge, as shown in Figure 11–9, or loading the results from the Match-Merge to a staging table before performing the join, as shown in Figure 11–10.

Figure 11–9 Valid Mapping Design with Sources Joined Before Match-Merge

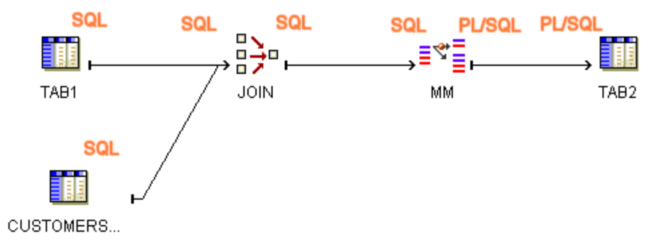


Figure 11–10 Valid Mapping Design with Staging Table

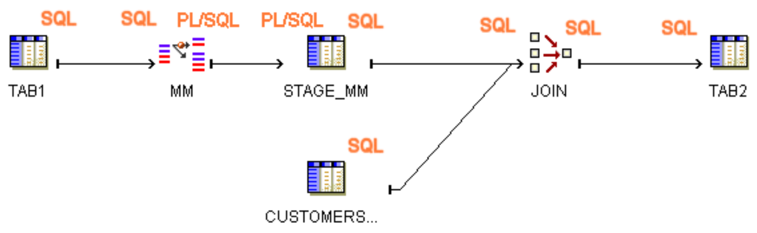


Table 11–2 and Table 11–3 list the implementation types for each Warehouse Builder operator. These tables also indicate whether or not PL/SQL code includes the operation associated with the operator in the cursor. This information is relevant in determining which operating modes are valid for a given mapping design. It also determines what auditing details are available during error handling.

Table 11–2 Source-Target Operators Implementation in PL/SQL Mappings

Operator	Implementation Types	Valid in Set Based Mode	Valid in Row Based Mode	Valid in Row Based (Target Only)
Source Operators: Tables, Cubes, Views, External Tables.	SQL	Yes.	Yes.	Yes. Part of cursor.
Target Operators: Tables, Cubes, Views,	SQL PL/SQL	Yes, except when loading =UPDATE or DELETE.	Yes.	Yes. Not part of cursor.
Flat File as source	For PL/SQL, create External Table.	Yes.	Yes.	Yes. Part of the cursor.
Flat File as target	SQL	Yes, but not targets with loading =UPDATE or DELETE.	Yes.	Yes. Not part of cursor.
Advanced Queue as source	SQL	Yes.	Yes.	Yes, part of cursor.
Advanced Queue as target	SQL	Yes, but not targets with loading =UPDATE or DELETE.	Yes.	Yes. Not part of cursor
Sequence as source	SQL	Yes.	Yes.	Yes, part of cursor.

Table 11–3 Data Flow Operator Implementation in PL/SQL Mappings

Operator Name	Implementation Types	Valid in Set Based Mode	Valid in Row Based Mode	Valid in Row Based (Target Only) Mode
Aggregators	SQL	Yes.	Yes, only if part of the cursor.	Yes, only if part of the cursor.
Constant Operator	PL/SQL SQL	Yes.	Yes.	Yes.
Data Generator	SQL*Loader Only	N/A	N/A	N/A
Deduplicator	SQL	Yes.	Yes, only if part of the cursor.	Yes, only if part of the cursor.
Expression	SQL PL/SQL	Yes.	Yes.	Yes.
Filter	SQL PL/SQL	Yes.	Yes.	Yes.
Joiner	SQL	Yes.	Yes, only if part of the cursor.	Yes, only if part of the cursor.
Key Lookup	SQL	Yes.	Yes, only if part of the cursor.	Yes, only if part of the cursor.
Mapping Input Parameter	SQL PL/SQL	Yes.	Yes.	Yes.
Mapping Output Parameter	SQL PL/SQL	Yes.	Yes.	Yes.
Match-Merge	SQL input PL/SQL output (PL/SQL input from XREF group only)	No.	Yes.	Yes. Not part of cursor.
Name-Address	PL/SQL	No.	Yes.	Yes. Not part of cursor.
Pivot	SQL PL/SQL	Yes.	Yes.	Yes.
Post-Mapping Process	Irrelevant	Yes, independent of dataflow.	Yes.	Yes.
Pre-Mapping Process	Irrelevant	Yes, independent of dataflow.	Yes.	Yes.
Set	SQL	Yes.	Yes, only if part of the cursor.	Yes, only if part of the cursor.
Sorter	SQL	Yes.	Yes, only if part of the cursor.	Yes, as part of the cursor.
Splitter	SQL PL/SQL	Yes.	Yes.	Yes.

Table 11–3 (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
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.

Using Partition Exchange Loading

Data partitioning can improve performance when loading or purging data in a target system. This performance feature is referred to 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:

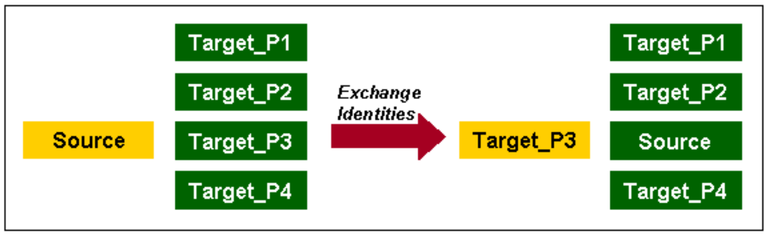
- "About Partition Exchange Loading" on page 11-21
- "Configuring a Mapping for PEL" on page 11-22
- "Configuring Targets in a Mapping" on page 11-25
- "Configuring Targets in a Mapping" on page 11-25
- "Restrictions for Using PEL in Warehouse Builder" on page 11-28

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 11–11 illustrates this example.

Figure 11–11 Overview of Partition Exchange Loading



In Figure 11–11, 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 Database completes the loading process without data movement.

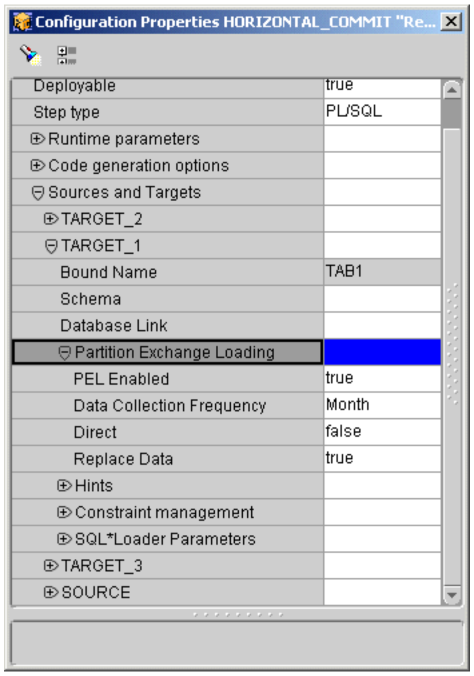
Configuring a Mapping for PEL

To configure a mapping for partition exchange loading, complete the following steps:

1. In the navigation tree, right-click a mapping and select **Configure.**

   Warehouse Builder displays the Configuration Properties window as shown in Figure 11–12.

Figure 11–12 PEL Configuration Properties



Configuration Properties HORIZONTAL_COMMIT "Re...

Deployable	true
Step type	PL/SQL
⊕ Runtime parameters	
⊕ Code generation options	
⊖ Sources and Targets	
⊕ TARGET_2	
⊖ TARGET_1	
Bound Name	TAB1
Schema	
Database Link	
⊖ Partition Exchange Loading	
PEL Enabled	true
Data Collection Frequency	Month
Direct	false
Replace Data	true
⊕ Hints	
⊕ Constraint management	
⊕ SQL*Loader Parameters	
⊕ TARGET_3	
⊕ SOURCE	

2. By default, PEL is disabled for all mappings. Set **PEL Enabled** to True 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. Set **Direct** to True if you want to create a temporary table to stage the collected data before performing the partition exchange. If you set this parameter to False, 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 11-23.

5. If you set **Replace Data** to True, Warehouse Builder replaces the existing data in the target partition with the newly collected data. If set to False **(default)**,

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.

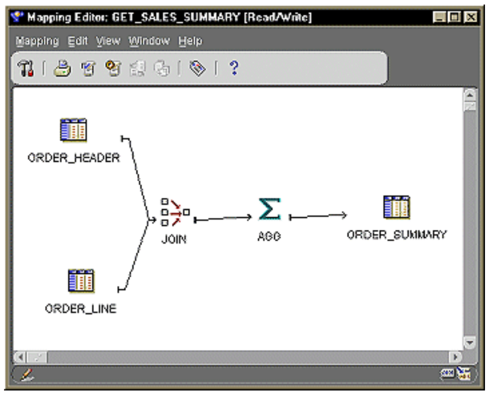
- **Indirect PEL:** By default, Warehouse Builder creates and maintains a temporary table that stages the source data before initiating the partition exchange process. For example, use Indirect PEL when the mapping includes a remote source or a join of multiple sources.
- **Direct PEL:** You design the 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 11–13 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 11–13 Mapping with Multiple Sources


Mapping Editor: GET_SALES_SUMMARY [Read/Write]
Mapping Edit View Window Help
ORDER_HEADER
JOIN
AGG
ORDER_SUMMARY
ORDER_LINE

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 11–13 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.

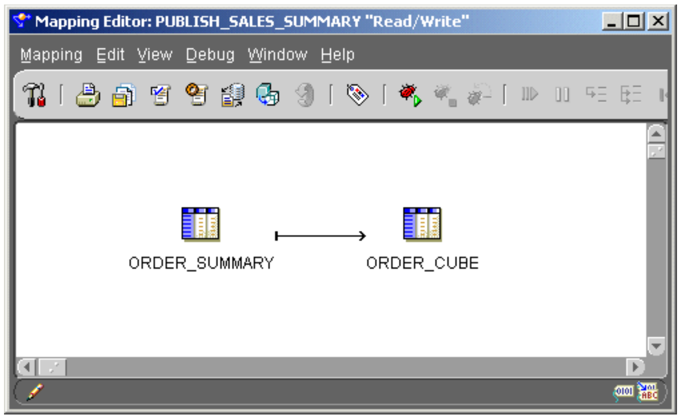
To instantly load data to a target using Direct PEL:

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 11–13 is ORDER_SUMMARY and should be of the same structure as the final target, ORDER_CUBE in Figure 11–14.

2. Create a second mapping that loads data from the staging table to the final target such as shown in Figure 11–14. Configure this mapping to use Direct PEL.

Figure 11–14 Publish_Sales_Summary Mapping


Mapping Editor: PUBLISH_SALES_SUMMARY "Read/Write"
Mapping Edit View Debug Window Help
ORDER_SUMMARY
ORDER_CUBE

3. Use either the Warehouse Builder Process Flow Editor or Oracle Workflow to launch the second mapping after the completion of the first.

Using PEL Effectively

You can use PEL effectively for scalable loading performance if the following conditions are true:

- **Table partitioning and tablespace:** The target table must be 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. You must create all partitions before you can use PEL. See "The new index name displays under Indexes in the Configuration Properties dialog." on page 5-4.

For example, if you select Month as the frequency of new data collection, you need to create all the required partitions for each month of new data. Use the Configuration Properties window to create partitions for a table, dimension, or cube. Figure 11–15 shows the property inspector window for table ORDER_SUMMARY. This figure shows six partitions that have been added for this table.

To use PEL, all partition names must follow a naming convention. For example, for a partition that will hold data for May 2002, the partition name must be in the format Y2002_Q2_M05.

For PEL to recognize a partition, its name must fit one of the following formats.

Ydddd

Ydddd_**Q**d

Ydddd_**Q**d_**M**dd

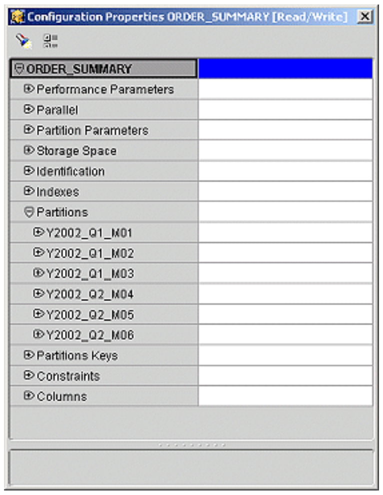
Ydddd_**Q**d_**M**dd_**D**dd

Ydddd_**Q**d_**M**dd_**D**dd_**H**dd

Ydddd_**Q**d_**M**dd_**D**dd_**H**dd_**M**dd

Where d represents a decimal digit. All the letters must be in upper case. Lower case is not recognized.

Figure 11–15 Configuration Properties for Table ORDER_SUMMARY


Configuration Properties ORDER_SUMMARY [Read/Write]

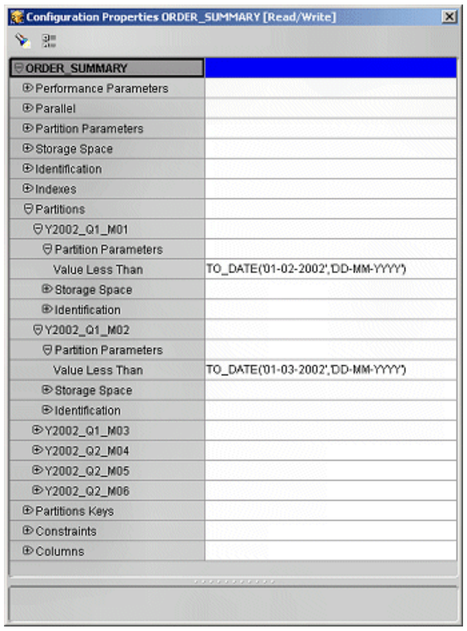
ORDER_SUMMARY	
Performance Parameters	
Parallel	
Partition Parameters	
Storage Space	
Identification	
Indexes	
Partitions	
Y2002_Q1_M01	
Y2002_Q1_M02	
Y2002_Q1_M03	
Y2002_Q2_M04	
Y2002_Q2_M05	
Y2002_Q2_M06	
Partitions Keys	
Constraints	
Columns	

If all partitions are added with correct names, Warehouse Builder automatically computes the Value Less Than property for each partition. Otherwise, the Value Less Than is must be manually configured for each partition in order for Warehouse Builder to generate a DDL statement. The following is an example of a DDL statement generated by Warehouse Builder:

```
. . .
PARTITION A_PARTITION_NAME
      VALUES LESS THAN (TO_DATE('01-06-2002','DD-MM-YYYY')),
. . .
```

Figure 11–16 shows automatically generated configuration values for the Value Less Than parameter.

Figure 11–16 Automatically Generated "Value Less Than" Setting


Configuration Properties ORDER_SUMMARY [Read/Write]

ORDER_SUMMARY	
Performance Parameters	
Parallel	
Partition Parameters	
Storage Space	
Identification	
Indexes	
Partitions	
Y2002_Q1_M01	
Partition Parameters	
Value Less Than	TO_DATE('01-02-2002','DD-MM-YYYY')
Storage Space	
Identification	
Y2002_Q1_M02	
Partition Parameters	
Value Less Than	TO_DATE('01-03-2002','DD-MM-YYYY')
Storage Space	
Identification	
Y2002_Q1_M03	
Y2002_Q2_M04	
Y2002_Q2_M05	
Y2002_Q2_M06	
Partitions Keys	
Constraints	
Columns	

Step 2: Create All Indexes Using the LOCAL Option

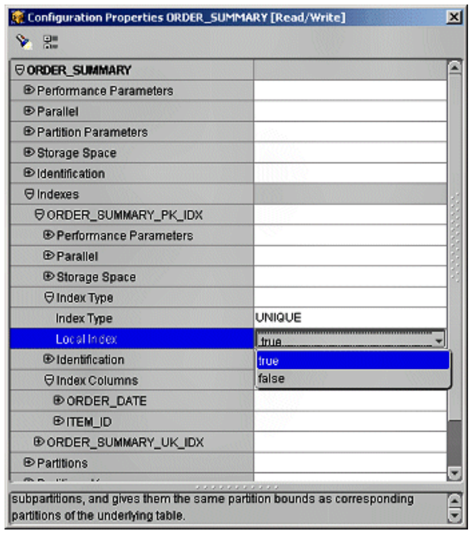
Figure 11–17 shows an index (ORDER_SUMMARY_PK_IDX) added to the ORDER_SUMMARY table. This index has two columns, ORDER_DATE and ITEM_ID. Configure the following:

- Set the Index Type parameter to UNIQUE.
- Set the Local Index parameter to True.

Now Warehouse Builder can generate a DDL statement for a unique local index on table ORDER_SUMMARY.

Using local indexes provides the most important PEL performance benefit. Local indexes require all indexes to be partitioned in the same way as the table. When the temporary table is swapped into the target table using PEL, so are the identities of the index segments.

Figure 11–17 Configure an Index as a Local Index



Configuration Properties ORDER_SUMMARY [Read/Write]

ORDER_SUMMARY	
Performance Parameters	
Parallel	
Partition Parameters	
Storage Space	
Identification	
Indexes	
ORDER_SUMMARY_PK_IDX	
Performance Parameters	
Parallel	
Storage Space	
Index Type	
Index Type	UNIQUE
Local Index	true
Identification	true / false
Index Columns	
ORDER_DATE	
ITEM_ID	
ORDER_SUMMARY_UK_IDX	
Partitions	

subpartitions, and gives them the same partition bounds as corresponding partitions of the underlying table.

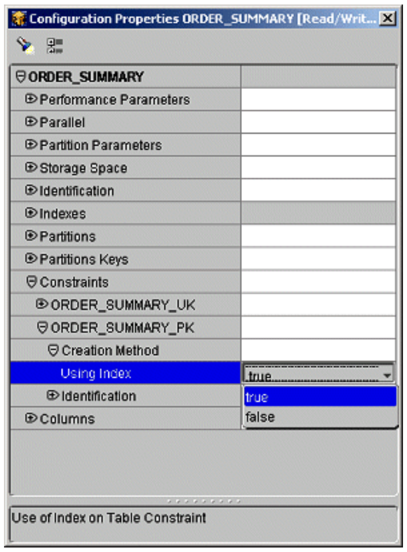
If an index is created as a local index, the Oracle server requires that the partition key column must be the leading column of the index. In the preceding example, the partition key is ORDER_DATE and it is the leading column in the index ORDER_SUMMARY_PK_IDX.

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. Figure 11–18 shows an example where the primary key constraint ORDER_SUMMARY_PK on the ORDER_SUMMARY table is specified with the USING INDEX option.

With the USING INDEX option, a constraint will not trigger automatic index creation when it is added to the table. The server will search existing indexes for an index with same column list as that of the constraint. Thus, each primary or unique key constraint must be backed by a user-defined unique local index. The index required by the constraint ORDER_SUMMARY_PK is ORDER_SUMMARY_PK_IDX which was created in "Step 2: Create All Indexes Using the LOCAL Option" on page 11-27.

Figure 11–18 Specify a Constraint with USING INDEX option


Configuration Properties ORDER_SUMMARY [Read/Writ...

ORDER_SUMMARY
Performance Parameters
Parallel
Partition Parameters
Storage Space
Identification
Indexes
Partitions
Partitions Keys
Constraints
ORDER_SUMMARY_UK
ORDER_SUMMARY_PK
Creation Method
Using Index: true
Identification
Columns

true
false

Use of Index on Table Constraint

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.

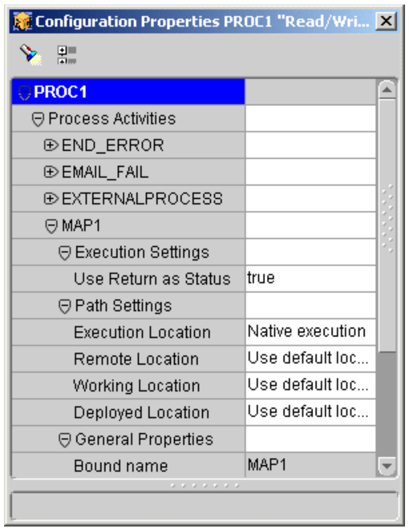
Configuring Process Flows

To configure a process flow:

1. Navigate to a process flow package, right-click a process flow and select **Configure.**

   Warehouse Builder displays the Configuration Properties sheet for the process flow as shown in Figure 11–19.

Figure 11–19 Process Flow Configuration Properties Sheet



PROC1	
Process Activities	
END_ERROR	
EMAIL_FAIL	
EXTERNALPROCESS	
MAP1	
Execution Settings	
Use Return as Status	true
Path Settings	
Execution Location	Native execution
Remote Location	Use default loc...
Working Location	Use default loc...
Deployed Location	Use default loc...
General Properties	
Bound name	MAP1

2. Expand **Execution Settings** and set the property **Use Return as Status.**

   This setting governs the behavior for activities that return NUMBER in their output. These activities include the FTP, External Process, and Transformation activities. When you set **Use Return as Status** to true, 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

3. Expand **Path Settings** and 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.

4. Expand **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.

Part III

Deploying and Executing

This part contains the following chapters:

12 Validating Objects

Validation verifies the definitions of both data objects and ETL objects and identifies any problems or possible errors that could occur during deployment. If objects are invalid, generation and deployment is not possible. You can validate objects and generate scripts for objects at any point in the design process. These functions are also available from within the deployment process; however, you can run them as standalone functions as you define your objects to ensure that the definitions you are providing are complete and valid. In addition, you can generate and view scripts prior to deployment to ensure that there are no problems or issues.

This chapter includes the following topics:

About Validation

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

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

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

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

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

Validating Objects

You can manually select objects for validation at anytime. Select the **Validate** operation from the Object menu or from the right-click menu for an object.

To validate an object or set of objects:

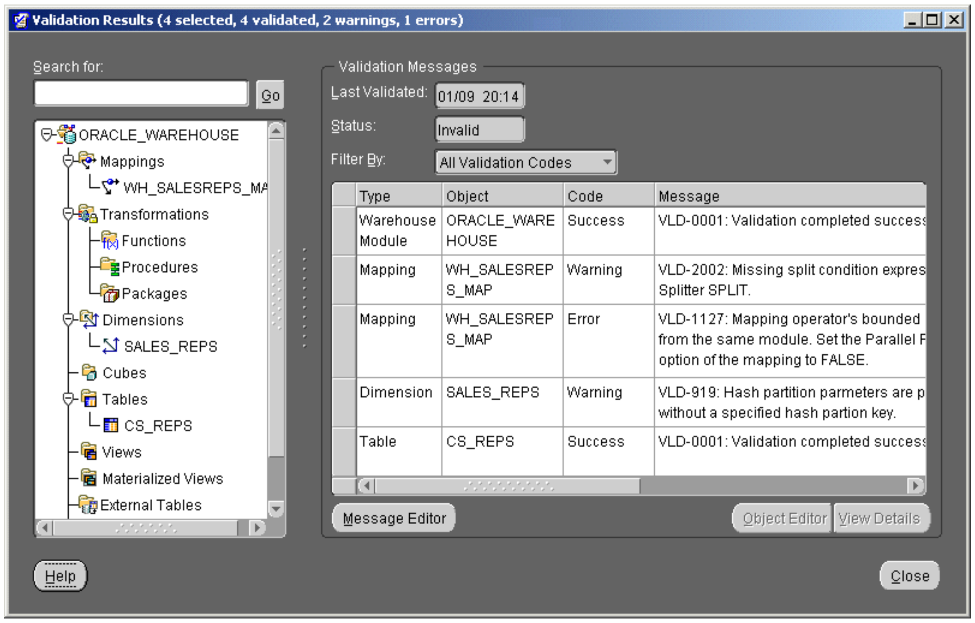
1. Select an object or set of objects from the project navigation tree.

   Use the **Control** key to select multiple objects. You can also select objects that contain objects, such as modules and projects.

2. From the **Object** menu, select **Validate** or right-click the object and select **Validate** from the pop-up menu.

   Warehouse Builder validates the selected object definitions and displays the results in the Validation Results dialog, as shown in Figure 12–1.

Figure 12–1 Validation Results Dialog



Last Validated: 01/09 20:14

Status: Invalid

Filter By: All Validation Codes

Type	Object	Code	Message
Warehouse Module	ORACLE_WAREHOUSE	Success	VLD-0001: Validation completed success
Mapping	WH_SALESREPS_MAP	Warning	VLD-2002: Missing split condition expres Splitter SPLIT.
Mapping	WH_SALESREPS_MAP	Error	VLD-1127: Mapping operator's bounded from the same module. Set the Parallel F option of the mapping to FALSE.
Dimension	SALES_REPS	Warning	VLD-919: Hash partition parmeters are p without a specified hash partion key.
Table	CS_REPS	Success	VLD-0001: Validation completed success

Viewing the Validation Results

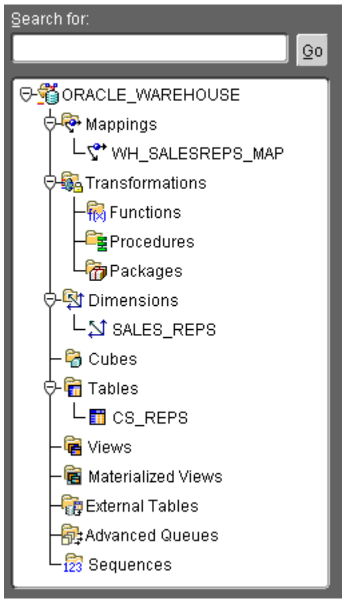
The Validation Results window enables you to view validation results and correct any invalid definitions. When the Validation Results window is opened, a navigation tree displays on the left and the validation messages display on the right. You can use these areas to find and view validation messages for specific objects. A summary of the number of objects selected and validated, and the number of warnings and errors display in the title bar of the window.

Note: A Validation Results window displays when you validate a definition. To view these results at a later time, select **Validation Messages** from the **View** menu.

The Validation Results Navigation Tree

The object or set of objects that were validated display in a navigation tree on the left side, as shown in Figure 12–2. The navigation tree is useful for locating specific objects when validating a large set of objects.

Figure 12–2 Validation Navigation Tree



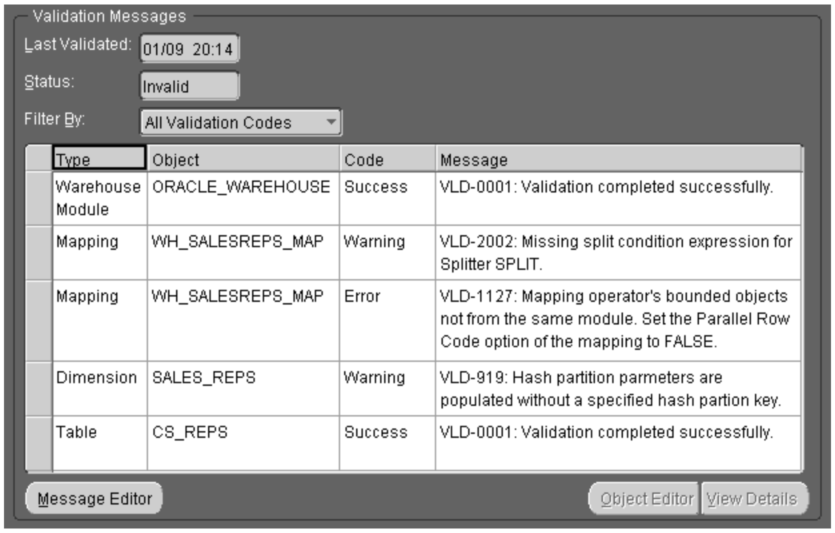
Using this tree, you can select an object or type of object and view the specific validation messages on the right. To search for a specific object, type in the name of the object in the **Search for** field and click **Go.** If the object you are searching for is included in the current Validation Results window, the results display on the right side.

The Validation Messages

The validation messages display on the right side of the Validation Results window. If the object you selected contains other objects, all objects are included in the results. For example, if you select a module for validation, the module definition and the definitions of all the objects within that module are validated. Validation messages display the type of the object, the object name, the validation code, and the validation message, as shown in Figure 12–3.

Note: There can be more than one validation message for a given object. If multiple errors or warnings occur, they display separately.

Figure 12–3 Validation Messages



Last Validated: 01/09 20:14
Status: Invalid

Type	Object	Code	Message
Warehouse Module	ORACLE_WAREHOUSE	Success	VLD-0001: Validation completed successfully.
Mapping	WH_SALESREPS_MAP	Warning	VLD-2002: Missing split condition expression for Splitter SPLIT.
Mapping	WH_SALESREPS_MAP	Error	VLD-1127: Mapping operator's bounded objects not from the same module. Set the Parallel Row Code option of the mapping to FALSE.
Dimension	SALES_REPS	Warning	VLD-919: Hash partition parmeters are populated without a specified hash partion key.
Table	CS_REPS	Success	VLD-0001: Validation completed successfully.

There are three types of message codes:

- **Success:** Indicates that the validation tests for this object were successfully passed. You can continue.
- **Warning:** Indicates that there may be problems during deployment. This is not a critical error. Warnings are shown to make you aware of potential issues.
- **Error:** Indicates that the object definition is invalid. The object definition cannot be generated.

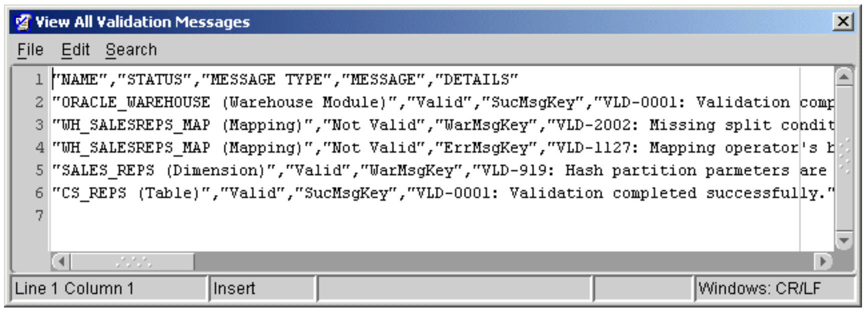
From the Validation Results window, you can use the following buttons:

- Message Editor Button
- Object Editor Button
- View Details Button

Message Editor Button

Click the **Message Editor** button to open all of the validation messages in a text editor, as shown in Figure 12–4. From here, you can type in notes about how you resolved each validation error or warning. You can then save this file and use it at a later time as a record of how you resolved different invalid objects. When you open this editor, all objects included in the current Validation Results window display in the editor.

Figure 12–4 Validation Message Editor



```
"NAME","STATUS","MESSAGE TYPE","MESSAGE","DETAILS"
"ORACLE_WAREHOUSE (Warehouse Module)","Valid","SucMsgKey","VLD-0001: Validation comp
"WH_SALESREPS_MAP (Mapping)","Not Valid","WarMsgKey","VLD-2002: Missing split condit
"WH_SALESREPS_MAP (Mapping)","Not Valid","ErrMsgKey","VLD-1127: Mapping operator's b
"SALES_REPS (Dimension)","Valid","WarMsgKey","VLD-919: Hash partition parmeters are
"CS_REPS (Table)","Valid","SucMsgKey","VLD-0001: Validation completed successfully."
```

Object Editor Button

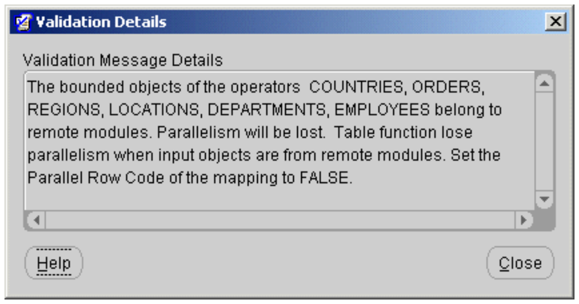
Click the **Object Editor** button to open an object editor for the selected object. For example, if you have a table selected in the Validation Results window and you click **Object Editor,** the Table Editor opens with the selected table. You can edit the object to correct any problems with the object properties or definition.You must have a specific object from the validation messages grid selected in order for this button to be enabled.

Note: When you edit objects using the Object Editor button from the Validation Results window, the objects are not automatically re-validated. You must re-validate any previously invalid objects.

View Details Button

Click the **View Details** button to view more comprehensive details on a message, as shown in Figure 12–5. This is helpful because some of the detail messages contain tips and advice on how to resolve certain issues and problems. You must have a specific object from the validation messages grid selected in order for this button to be enabled.

Figure 12–5 Validation Details Dialog



The bounded objects of the operators COUNTRIES, ORDERS, REGIONS, LOCATIONS, DEPARTMENTS, EMPLOYEES belong to remote modules. Parallelism will be lost. Table function lose parallelism when input objects are from remote modules. Set the Parallel Row Code of the mapping to FALSE.

Editing Invalid Objects

When you validate objects, the Validation Results window displays. From here you can view the invalid objects and access the editors for these objects directly.

To edit invalid definitions:

1. From the Validation Results window, select an invalid object from the tree or from the validation messages grid and click **Object Editor.**

   An editor for the selected object opens.

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

Viewing Generated Scripts

In addition to validating object definitions prior to deployment, you can also generate and view the scripts. During this process, Warehouse Builder validates the object definitions and generates the scripts required to create and populate the objects. You can then view the generation results and the code generated for each script. You can also save these scripts to a file system.

Note: The scripts generated using the method described in this section may not be deployable. This is for viewing and verification purposes only. If you choose, you may also save these scripts to a file system.

Generating Scripts

To generate scripts for an object or set of objects:

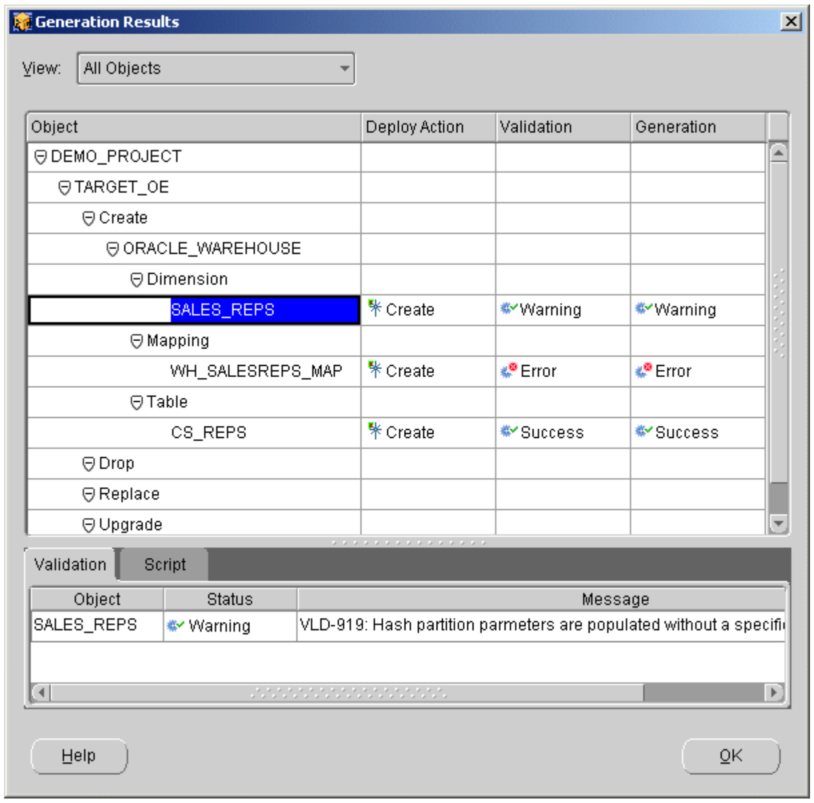
1. From the project navigation tree, select an object or set of objects.

   Select multiple objects using the **Control** key. You can also select modules, however, it may take a long time to generate scripts if it contains a large number of objects.

2. From the **Object** menu, select **Generate.** You can also right-click the object and select **Generate.**

   Warehouse Builder generates the scripts for the selected objects. The Generation Results window, as shown in Figure 12–6, displays when the operation is complete.

Figure 12–6 Generation Results Window



Object	Deploy Action	Validation	Generation
DEMO_PROJECT			
TARGET_OE			
Create			
ORACLE_WAREHOUSE			
Dimension			
SALES_REPS	Create	Warning	Warning
Mapping			
WH_SALESREPS_MAP	Create	Error	Error
Table			
CS_REPS	Create	Success	Success
Drop			
Replace			
Upgrade			

Object	Status	Message
SALES_REPS	Warning	VLD-919: Hash partition parmeters are populated without a specifi

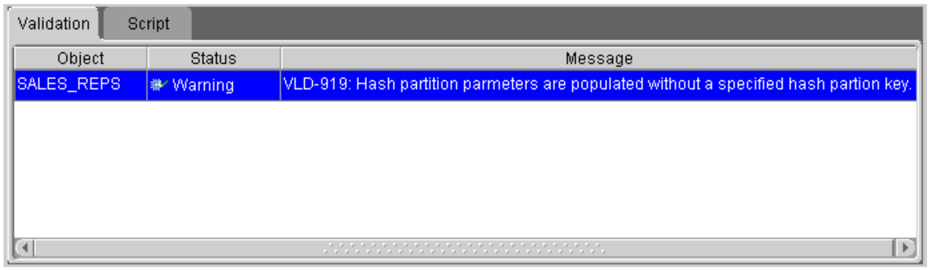
Viewing the Generation Results

The Generation Results window is divided into two main areas. The top area, organized into a navigation tree, enables you to locate objects of interest. There is a drop-down filter that enables you to limit the objects that display. You can also expand or collapse object types.

Note: The Generation Results window displays when you generate a script to create an object. To view these results at a later time select **Generated Scripts** from the **View** menu.

After you have selected an object, specific generation and validation results display in the bottom area. Select the Validation tab to view validation messages that exist for the selected object, as shown in Figure 12–7. This is for viewing purposes only.

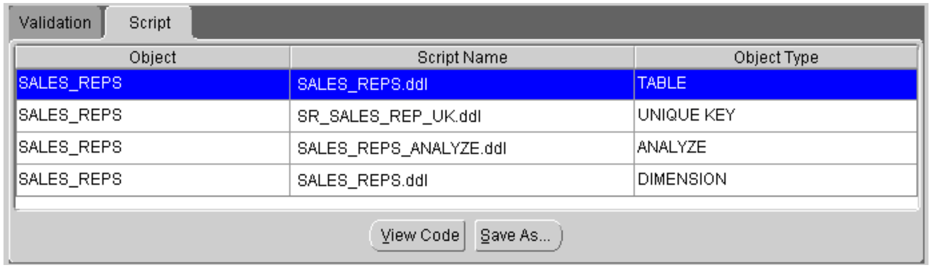
Figure 12–7 Validation Tab



Object	Status	Message
SALES_REPS	Warning	VLD-919: Hash partition parmeters are populated without a specified hash partion key.

Select the Script tab to view a list of the scripts that will be generated in order to create the object you have selected, as shown in Figure 12–8. The list contains the name of the object, the name of the scripts, and the type of object that each script will generate.

Figure 12–8 Script Tab



Object	Script Name	Object Type
SALES_REPS	SALES_REPS.ddl	TABLE
SALES_REPS	SR_SALES_REP_UK.ddl	UNIQUE KEY
SALES_REPS	SALES_REPS_ANALYZE.ddl	ANALYZE
SALES_REPS	SALES_REPS.ddl	DIMENSION

Viewing the Scripts

After you have generated scripts for your target objects, you can open the scripts and view the code. Warehouse Builder generates the following types of scripts:

- **DDL scripts:** Creates or drops database objects.
- **SQL*Loader control files:** Extracts and transports data from file sources.
- **ABAP scripts:** Extracts and loads data from SAP systems.

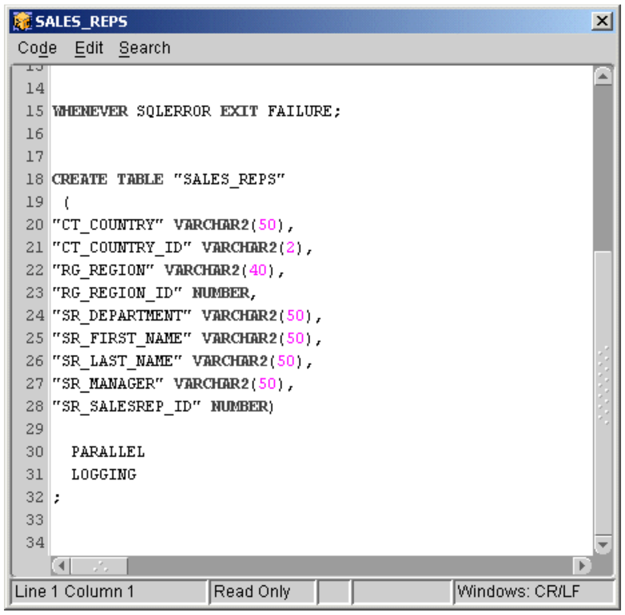
To view the generated scripts:

1. From the Generation Results window, select an object from the top section of the window.
2. Select the Scripts tab from the bottom section of the window.

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

   The selected script displays in a code viewer, as shown in Figure 12–9. This is read-only.

Figure 12–9 Generated Script



```
WHENEVER SQLERROR EXIT FAILURE;


CREATE TABLE "SALES_REPS"
 (
"CT_COUNTRY" VARCHAR2(50),
"CT_COUNTRY_ID" VARCHAR2(2),
"RG_REGION" VARCHAR2(40),
"RG_REGION_ID" NUMBER,
"SR_DEPARTMENT" VARCHAR2(50),
"SR_FIRST_NAME" VARCHAR2(50),
"SR_LAST_NAME" VARCHAR2(50),
"SR_MANAGER" VARCHAR2(50),
"SR_SALESREP_ID" NUMBER)

  PARALLEL
  LOGGING
;
```

Saving the Scripts

When you view the generated scripts, you also have to option of saving them to a file system.

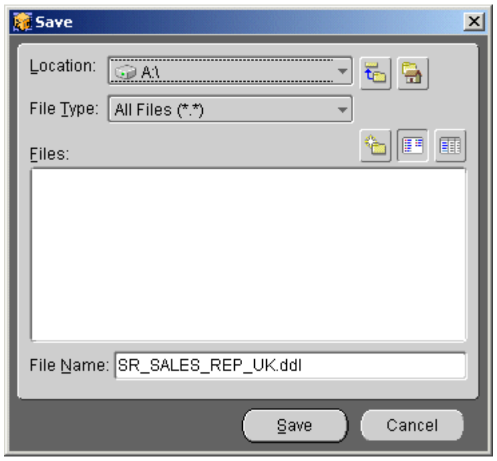
To save generated scripts:

1. From the Generation Results window, select an object from the top section of the window.
2. Select the Scripts tab from the bottom section of the window.

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

   The Save dialog opens, as shown in Figure 12–10, and you can select a location where you want to save the script file.

Figure 12–10 Save Option



Location: A:\
File Type: All Files (*.*)
Files:
File Name: SR_SALES_REP_UK.ddl

13

Deploying Target Systems

After you design and configure the logical definitions of your target system, you can deploy and create the physical instance of your target. You can then execute deployed mapping and process flow scripts to load or update your data. If you are working with a previously deployed system, you can view deployment history and plan an upgrade. All of these processes are all managed by a component in Warehouse Builder called the Runtime Platform Service.

This chapter contains the following topics:

About Deployment

Deployment is the process of creating your target system from the logical design or model. The process includes generating scripts such as DDL that create data objects such as tables, views, and dimensions. The process also includes generating PL/SQL and SQL*Loader scripts that load data into data objects. The Runtime Repository stores detailed information about every deployment. This information determines default deployment actions for future deployments. For example, when you deploy a set of objects that already exists in the target system, upgrade is the default action. You can also access deployment data reports using the Runtime Audit Browser. For more information, see Chapter 14, "Auditing Deployment and Execution".

Deploying and Upgrading Target Systems

When you deploy or upgrade a target system using Warehouse Builder, you can either use the Deployment Manager or you can deploy objects directly from the navigation tree. The Deployment Manager offers a comprehensive deployment console that enables you to view and manage all aspects of deployment including configuration and validation. The Deployment Manager also enables you to view the deployment history of an object to determine how you want to deploy the object. These options are not available if you deploy objects from the navigation tree.

Prior to deployment, you must ensure the following:

- You have assigned locations for all deployable modules.
- You have installed the Runtime Repository and the repository is referenced by a Runtime Repository Connection in the navigation tree.
- You have defined connectors where needed.

You can define runtime repository connections, locations, and connectors at anytime prior to deployment. These objects are necessary for deployment as they define the connection information for data sources and target as well as the connection between these locations. After these are defined, you can proceed to deploying to your target system.

Defining Runtime Repository Connections

Runtime Repository Connections describe a connection to the Runtime Repository that represents the collection of systems and tools that comprise the target system you design. The Runtime Repository also helps to manage deployments to this collection and collects all of the audit data. These connections represent connections to the Runtime Repositories that were installed using the Runtime Assistant. For more information about the Runtime Repository, see the Oracle Warehouse Builder Installation and Configuration Guide.

When you deploy objects, you must select a Runtime Repository Connection. The physical details of the logical locations are registered within a Runtime Repository. The Runtime Repository dispatches the generated scripts to the appropriate physical location during the deployment operation. Audit data created during the deployment operation is stored in the Runtime Repository and can be displayed by using the Runtime Audit Browser. Before you can use deployment, you must install Runtime Repositories, and create Runtime Repository Connections in the navigation tree.

Creating Runtime Repository Connections

To create a new runtime repository connection:

1. Select a project and expand the navigation tree.
2. Select the Runtime Repository Connections node.
3. From the **Object** menu select **Create Runtime Repository Connection** or right-click Runtime Repository Connections and select **Create Runtime Repository Connection.**

   The Runtime Repository Connection Wizard Welcome page displays.
4. Click **Next.**

   The Name Page displays. Use this page to define the following information for the runtime repository connection:

   - **Name:** Provide a name for the runtime repository connection. The name must not exceed 30 characters.
   - **Optional Description:** Provide an optional description for the runtime repository connection. The description must not exceed 400 characters.

   There are no pre-assigned default values assigned to a new runtime repository connection.

5. Click **Next** to continue.

   The Details Page displays. Use this page to define the following information for the runtime repository:

   - **Host Name:** Type the name of the host machine.
   - **Port Number:** Specify the Oracle Listener port number. The default value is 1521.
   - **Service Name:** Type the service name.
   - **Connect As User:** Type a user name that has been granted access to this runtime repository.
   - **Runtime Repository Owner:** Type the name of the runtime repository for which you are creating the connection. This represents the schema name that was entered using the Runtime Assistant.

6. Click **Next** to continue.

   The Finish Page displays. Use this page to verify the definition of the new runtime repository. This page lists the name, host name, port number, service name, user name, and runtime repository name.

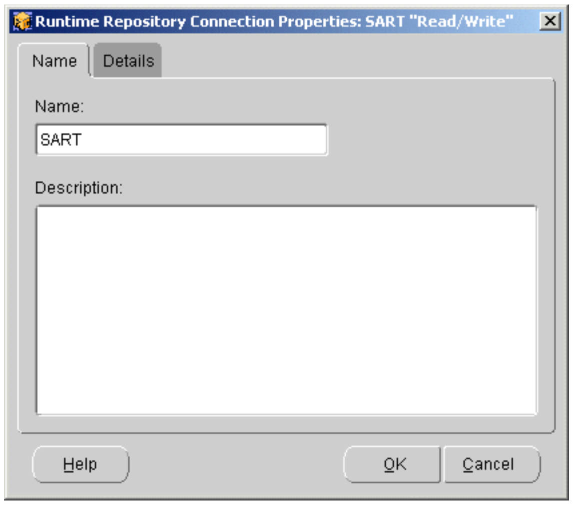
7. Click **Finish** to create the runtime repository as defined.

   The runtime repository connection is created and added under the Runtime Repository Connections node.

Editing Runtime Repository Connections

To edit a runtime repository connection, right-click the runtime repository connection from the navigation tree and select **Properties.** Warehouse Builder displays the properties window as shown in Figure 13–1.

Figure 13–1 Runtime Repository Connection Properties



Name:
SART

Description:

This page displays the properties of the selected runtime repository connection. Use the following two tabs to view and edit the properties. Click **OK** to save changes or **Cancel** to close the window.

Name Tab

- **Name:** Displays the runtime repository connection name. The name must not exceed 30 characters.
- **Description:** Displays an optional description for the runtime repository connection. The description must not exceed 400 characters.

Details Tab

- **Host Name:** Type the name of the host machine.
- **Port Number:** Specify the Oracle Listener port number.
- **Service Name:** Type the service name.
- **Connect As User:** Type a user name that has been granted access to this runtime repository.
- **Runtime Repository Owner:** Type the name of the runtime repository for which you are creating the connection. This represents the schema name that was entered using the Runtime Assistant.

Deploying Objects

When you are ready to deploy objects, you can either use the Deployment Manager or you can select and deploy objects from the navigation tree. Deploying objects from the navigation tree is a very simple, quick process of deployment. This method is beneficial if you want Warehouse Builder to use the default deployment actions and you do not need to view the script prior to deployment. The default deployment action is determined by changes to the object design since the object was last deployed. For example, if you deploy an altered object, the default action will be upgrade.

The Deployment Manager offers a comprehensive deployment console and allows for more flexibility in deployment options. For more information, see "Using the Deployment Manager" on page 13-4.

To deploy objects from the navigation tree:

1. Select a deployable object from the navigation tree.
2. From the Object menu, select **Deploy** or right-click the object and select **Deploy.**

   The Select Runtime Repository Connection dialog displays.
3. Select the Runtime Repository Connection and click **OK.**

   Warehouse Builder generates the scripts for the selected objects and the Pre-Deployment Generation Results window displays.
4. Click **Deploy** to continue, or **Cancel** to cancel.

   Warehouse Builder deploys the object using the default deployment settings. The Runtime Repository you select stores the data about the deployment.

Using the Deployment Manager

The Deployment Manager offers the most flexible way to deploy objects using Warehouse Builder. When you open the Deployment Manager, you can access the design objects that exist in your current project. After the Deployment Manager is open, you can select objects from the tree and set them for deployment. Special icons appear on the tree next to these objects. After you have selected the objects and set their deployment action you can then use the Deploy button to validate the objects and generate the scripts based on their current design. These results display in the generation preview screen. You can confirm the objects you want to deploy as well as catch any errors. Complete the deployment by deploying the scripts to the target locations using the current Runtime Repository Connection.

Opening the Deployment Manager

To open the Deployment Manager:

1. From the Project menu, select **Deployment Manager.**

   A dialog displays asking for you to select a Runtime Repository Connection.

2. Select the Runtime Repository Connection from the drop-down list and click **OK.**

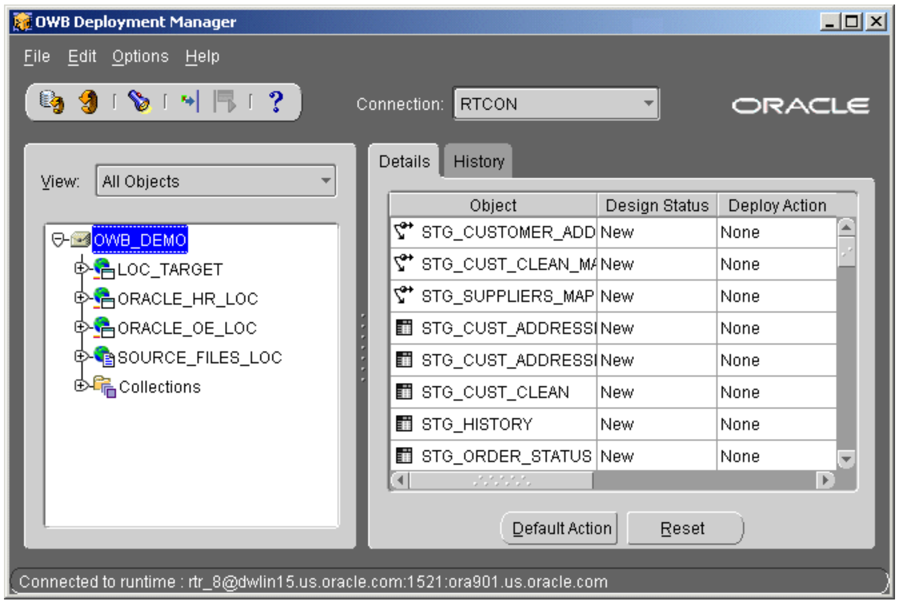
   The Connection Information dialog for the Runtime Repository you have selected displays.

3. Enter the password for the Runtime Repository you have selected.

   The first time you connect to a specific Runtime Repository, you must verify the password. The password is then stored and you can access the repository during the Warehouse Builder session without entering it again.

   The Deployment Manager opens with the current project displayed in the left window, as shown in Figure 13–2.

Figure 13–2 Deployment Manager



Connection: RTCON

View: All Objects

- OWB_DEMO
  - LOC_TARGET
  - ORACLE_HR_LOC
  - ORACLE_OE_LOC
  - SOURCE_FILES_LOC
  - Collections

Object	Design Status	Deploy Action
STG_CUSTOMER_ADD	New	None
STG_CUST_CLEAN_MA	New	None
STG_SUPPLIERS_MAP	New	None
STG_CUST_ADDRESSI	New	None
STG_CUST_ADDRESSI	New	None
STG_CUST_CLEAN	New	None
STG_HISTORY	New	None
STG_ORDER_STATUS	New	None

Connected to runtime : rtr_8@dwlin15.us.oracle.com:1521:ora901.us.oracle.com

About the Deployment Manager

The Deployment Manager has the following components:

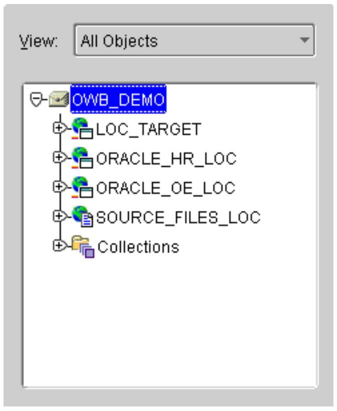
- Deployment Tree
- Details Tab
- History Tab
- Toolbar

Deployment Tree

When you open the Deployment Manager, the deployment tree is on the left side. The tree is initially collapsed into a list of Locations and Collections which can be expanded to display their contents, as shown in Figure 13–3. You can use this tree to register locations, select objects for deployment, and view deployment history. Modules that do not have an associated location do not display in the Deployment Manager.

Note: Only Oracle target warehouse locations expand to show the deployable objects contained within them. All source locations and Flat File locations display the location for registration purposes only.

Figure 13–3 Deployment Tree



When you select an object in the tree, the deployable objects within that object are displayed on the right. You can select multiple objects from the deployment tree by using the Control or Shift keys.

View Selector Select a view from the drop-down list above the deployment tree to limit the objects displayed. Table 13–1 describes the columns displayed in the view selector.

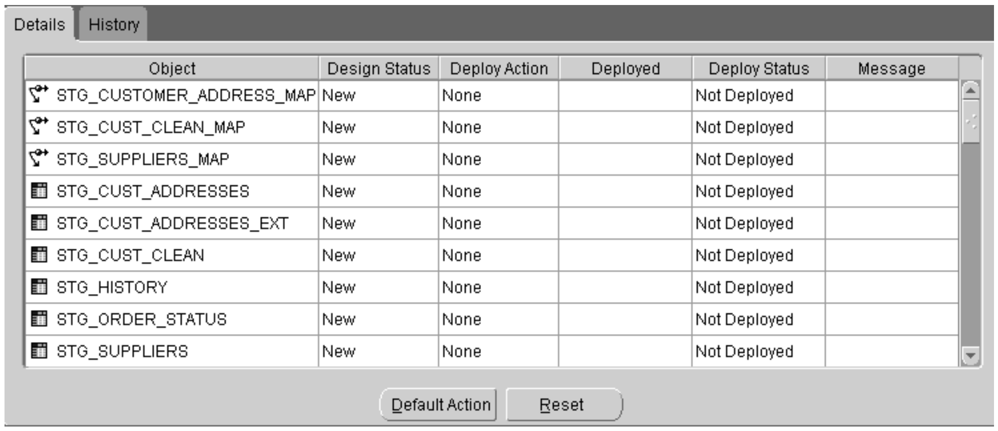
Table 13–1 Views

View Name	Objects Displayed
All Objects	Clears any previously selected filter and displays all objects in the deployment tree.
Changed Objects	Objects that have been changed in the Warehouse Builder Design Repository since the last deployment of the object. The deployment actions for these objects default to upgrade or replace.
Deployed Objects	Objects that have been previously deployed using the runtime repository that is currently selected.
Projected Deployment Objects	All Objects currently selected for deployment.

Details Tab

The Details Tab, located on the right side of the Deployment Manager, displays a summary of information relevant to the object or set of objects selected in the deployment tree. As the status of an object changes, the Details tab reflects those changes. You can also use the buttons on the bottom of the tab to alter the Deploy Action status. The Default Action button changes the Deploy Action, and the Reset button resets the Deploy Action columns back to None if an changes have been made. Figure 13–4 shows the Deployment Window with the Details Tab displayed.

Figure 13–4 Deployment Details Tab



Object	Design Status	Deploy Action	Deployed	Deploy Status	Message
STG_CUSTOMER_ADDRESS_MAP	New	None		Not Deployed	
STG_CUST_CLEAN_MAP	New	None		Not Deployed	
STG_SUPPLIERS_MAP	New	None		Not Deployed	
STG_CUST_ADDRESSES	New	None		Not Deployed	
STG_CUST_ADDRESSES_EXT	New	None		Not Deployed	
STG_CUST_CLEAN	New	None		Not Deployed	
STG_HISTORY	New	None		Not Deployed	
STG_ORDER_STATUS	New	None		Not Deployed	
STG_SUPPLIERS	New	None		Not Deployed	

Table 13–2 describes the columns displayed in the Details tab.

Table 13–2 Detail Tab Columns

Column	Description
Object	Displays the physical name of the object selected.
Design Status	Displays status information about the object stored in the design repository. Available statuses are: New: Object exists in the design repository, but has not yet been deployed. No Change: No change in status since last deployment. Deleted: Object is in the runtime but not in the design repository. Updated: Object was deployed and has been updated in the design repository since the last deployment.
Deploy Action	Action to be performed when the object is deployed. Actions include Create, Upgrade, Drop and Replace.
Deployed	The timestamp of when the object was last deployed.
Deploy Status	The current deployment status of the object in the target.
Message	Lists out any messages.

The Default Action button automatically updates the action for the objects you select. The default action is determined by the data stored in the design and runtime repositories. The details for the objects are shown in the Details tab. The following rules are used to determine the default action:

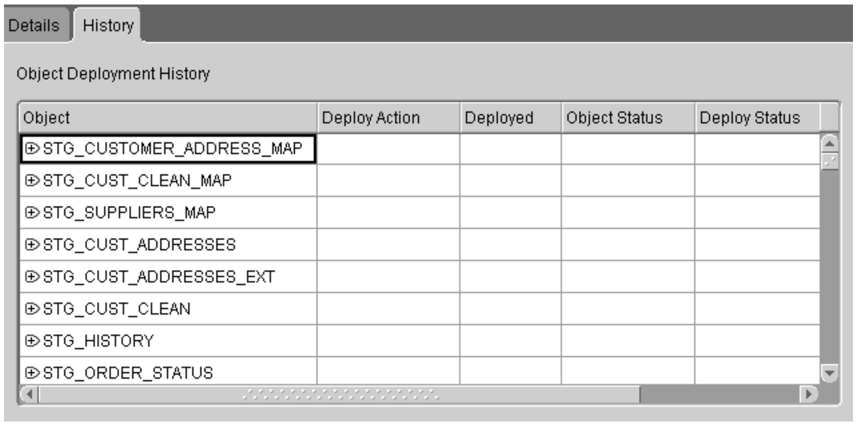
- Design objects not deployed have their action set to Create.
- Previously deployed objects that have been modified since the last deployment will have their action set to Upgrade or Replace depending on how the target system is configured.
- If there is no need to deploy an object, the action is set to None.

Note: Due to the way External Tables are processed, there are no default actions available for these objects. You must manually set the Deploy Action for External Tables.

History Tab

The History tab displays the deployment history for the selected objects, as shown in Table 13–5.

Figure 13–5 Deployment History Tab



Object	Deploy Action	Deployed	Object Status	Deploy Status
STG_CUSTOMER_ADDRESS_MAP				
STG_CUST_CLEAN_MAP				
STG_SUPPLIERS_MAP				
STG_CUST_ADDRESSES				
STG_CUST_ADDRESSES_EXT				
STG_CUST_CLEAN				
STG_HISTORY				
STG_ORDER_STATUS				

Table 13–3 describes the columns displayed in the History tab.

Table 13–3 History Tab Columns

Column	Description
Object	Displays the physical name of the object selected.
Deploy Action	Action performed when the object was deployed. Actions include Create, Upgrade, Drop and Replace.
Deployed	The timestamp of when the object was deployed.
Object Status	Displays whether the object that was deployed is valid or invalid.
Deploy Status	The current deployment status of the object in the target.

Toolbar

The toolbar, as shown in Table 13–6, is located in the upper left side of the Deployment Manager contains shortcuts to a few tasks.

Figure 13–6 Deployment Manager Toolbar

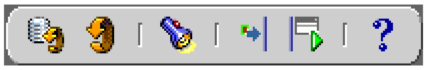


Table 13–4 describes these tools.

Table 13–4 Deployment Toolbar

Icon	Action	Description
	Runtime Repository Synchronize	Synchronizes objects that display in the Deployment Manager with objects that exist in the Runtime Repository. This function is only available from within the Deployment Manager.
	Design Repository Synchronize	Synchronizes objects that display with objects that exist in the Warehouse Builder Design Repository. This is useful when there are multiple users. This provides the same function as the synchronize button in the main console.
	Find	Searches for an object within the Deployment Manager. The Object Find dialog displays and you can type in the name or part of the name of an object you are searching for. Select this icon at anytime.
	Deploy	Deploys selected objects.
	Execute	Executes the object selected in the deployment tree. This is only enabled if you have an executable object selected.
	Help	Displays the Oracle Warehouse Builder User's Guide in the online help navigator. You can select this icon at anytime.

Registering Locations

Before objects can be deployed successfully, all of the location being used must be registered. When you create locations during the design process, you create logical definitions which are limited to only the name, type, and version of the location. This logical information is stored and used to determine which objects can be deployed. All modules must have locations assigned to them in order for them to be used during deployment. This includes the target as well as any sources or files.

When you register locations, you specify the connection information that will be used during deployment to connect to the various data sources and targets. You only have to supply this information the first time you use the location during a deployment. The same connection information is used for future deployments unless you change it.

Database Connections

Database locations can be defined using either Host Name, Port Number and Service Name or by Net Service Name and Service Name. Net Service Name is a name that is defined in your tnsnames.ora file. A Net ServiceName should be defined in the appropriate Oracle home. For example, to deploy to a location identified by Net Service Name the name should be defined in the Oracle home of the Runtime Platform Service.

Database links are generated in the form <service-name>@<connector-name>. If you have global-names set to true, it is important that the service-name you specify for the location matches the global-name of the database you want to link to.

For RAC systems the service-name is typically the cluster-service name. To use RAC it is advisable that you use Net Service Names since this will give you control over some of the RAC features in the tnsnames definition, such as client-side load balancing.

Location Registration

To register locations:

1. Open the Deployment Manager, and select a Location from the deployment tree.

   > **Note:** You can also register locations during the deployment process. During deployment, a Location Registration window will appear for each location that has not been previously registered.

2. Select **File,** and then **Register.** You can also right-click the location and select **Register.**

   The Location Registration window appears. Depending on the type of location, there are different connection information requirements. There are no pre-assigned default values.

3. Type in the location registration details and then click **OK** to return to the Deployment Manager. The following sections contain detailed descriptions of the required registration information for each type of location.:

For Oracle Locations:

- **Schema Name:** Type the user name to logon to the database. Maximum Length: 30 Characters.
- **Password:** Type the password for the specified user name. Maximum Length: 30 Characters.
- **Service Name:** Type the database service name. Maximum Length: 30 Characters.

Select either Net Service Name or Host Name and provide the following connection details:

- **Net Service Name:** Type the tnsname for the database. The tnsname contains the Service Name in the tnsnames.ora file. Maximum Length: 30 Characters.
- **Host Name:** Type the name of the machine. Maximum Length: 30 Characters.
- **Port Number:** Specify the Oracle Listener port number. Maximum Length: 5 Characters.

For Non-Oracle Locations:

- **Schema:** Type the owner of the database objects. Maximum Length: 30 Characters.
- **Connect as User:** Type the user to connect to the database. Maximum Length: 30 Characters.
- **Password:** Type the password for the specified user name. Maximum Length: 30 Characters.
- **Service Name:** Type the database service name for non-Oracle database. Maximum Length: 30 Characters.

Select either Net Service Name or Host Name and provide the following connection details:

- **Net Service Name:** Type the tnsname for the database. The tnsname contains the Service Name in the tnsnames.ora file. Maximum Length: 30 Characters.
- **Host Name:** Type the name of the machine that runs the non-Oracle database. Maximum Length: 30 Characters.
- **Port Number:** Specify the Oracle Listener port number. Maximum Length: 5 Characters.

For File System Locations:

- **User Name:** Type the user name to connect to the operating system. This user name is used for example in case of an FTP process. Maximum Length: 30 Characters.
- **Password:** Type the password for the specified user name. Maximum Length: 30 Characters.
- **Host Name:** Type the name of the machine that contains the flat file or files. Maximum Length: 30 Characters.
- **Root Path:** Specify the path to the directory in which the file or files are located. The path you specify must end in a trailing slash. For example: c:/temp/.

For Oracle Enterprise Manager Locations:

- **User Name:** Type the name of an Oracle Management Server user that has job registration privileges. Typically a super-user in Oracle Enterprise Manager has the right privileges. The default super-user is SYSMAN. Maximum Length: 30 Characters.
- **Password:** Type the password for the specified user name. Maximum Length: 30 Characters.
- **OMS Domain:** Type the name of the node that runs the Oracle Management Server. Maximum Length: 30 Characters.
- **Target Agent Name:** Type the name of the node that will execute the code. Maximum Length: 30 Characters.

For Oracle Workflow Locations:

- **Schema:** Type the user name of the Oracle Workflow server. Maximum Length: 30 Characters.
- **Password:** Type the password for the specified user name. Maximum Length: 30 Characters.
- **Service Name:** Type the database service name for the Workflow server. Maximum Length: 30 Characters.

Select either Net Service Name or Host Name and provide the following connection details:

- **Net Service Name:** Type the tnsname for the database. The tnsname contains the Service Name in the tnsnames.ora file. Maximum Length: 30 Characters.
- **Host Name:** Type the name of the machine that runs the Workflow server. Maximum Length: 30 Characters.
- **Port Number:** Specify the Oracle Listener port number. Maximum Length: 5 Characters.

For SAP Locations:

- **Schema:** Type the name of the user that has access to the SAP data objects. Maximum Length: 30 Characters.
- **Password:** Type the password for the specified user name. Maximum Length: 30 Characters.
- **Service Name:** Type the database service name for the SAP instance. Maximum Length: 30 Characters.

Select either Net Service Name or Host Name and provide the following connection details:

- **Net Service Name:** Type the tnsname for the database. The tnsname contains the Service Name in the tnsnames.ora file. Maximum Length: 30 Characters.
- **Host Name:** Type the name of the machine that runs SAP. Maximum Length: 30 Characters.
- **Port Number:** Specify the Oracle Listener port number. Maximum Length: 5 Characters.

Selecting Objects for Deployment

To select objects for deployment:

1. Open the Deployment Manager.

   The Deployment Manager displays a collapsed view of the project tree in the left pane of the window.
2. Expand the project to display the contents. You can choose from one of the following filters to view a partial list of the objects:
   - **Changed:** Displays all objects that have changes since the last deployment.
   - **Deployed:** Displays all objects that exist in the Runtime Repository.
   - **Projected Deployment:** Displays all objects that have been deployed and that are currently selected for deployment.
3. Select an object or set of objects you want to deploy. The details about the object or objects display in the Details tab on the right pane of the window.
4. To select an object for initial deployment, click the Deploy Action column and choose Create from the drop-down list.

   If this is not you initial deployment you can select another action. Actions include Create, Upgrade, Drop and Replace.

   Alternatively, you can click Default Action. This will change the Deploy Action for each object to a default value. If the object has not been previously deployed, the default action is Create.
5. Continue this process for all objects you want to deploy. Special icons display on the deployment tree next to the objects selected for deployment.

Note: All deploy actions are not available for all objects even though they appear on the drop-down list. For example, you cannot select Upgrade as the deploy action for mappings. If you want to replace old mappings you must use the Replace action.

Viewing Deployment History

The Deployment Manager enables you to view the deployment history of objects within the project. This can be useful when making decisions about upgrades.

To view deployment history:

1. Open the Deployment Manager.
2. Select the History Tab.

3. From here you can select items on the tree to display the deployment history.

Completing the Deployment

After you have selected the objects you want to deploy, there are a few steps to complete the deployment.

To complete deployment:

1. From the File menu, select **Deploy** or click the Deploy icon in the toolbar.

   If this is the first time you are deploying objects to a Location, you will be asked for physical details of the Location.

2. Specify the physical connection details for the locations and press **OK.**

   The Pre Deployment Generation Results page displays.

   You can view the generated script by selecting the script in the lower half of the dialog and clicking **View Code.**

3. Click **Deploy** to confirm the deployment.

   The deployment is completed and the Deployment Results display.

4. Review the Deployment Results and click **OK.**

   The deployment is now finalized. The deployment action and status for the objects you deployed are updated in the Deployment Manager.

Saving Deployment Scripts

In some cases you may find it helpful to save all deployment scripts for a given deployment. You can do that using the Deployment Manager.

To save deployment scripts:

When you are deploying objects, a Deployment Preview page displays prior to deployment completion. From this page, you can save the individual file or the entire deployment specification.

Executing Deployed Objects

You can execute two types of objects after deployment: mappings and process flows. The most direct way to execute mappings and process flows designed and deployed using Warehouse Builder is to use the Deployment Manager. After you deploy process flows or mappings to your target system, they are available for execution. You can select a mapping or process flow from the project tree within the Deployment Manager and then execute it.

Executing Mappings and Process Flows

To select an object for execution:

1. Open the Deployment Manager and expand the project containing the object you want to execute.

   The Deployment Manager displays an expanded view of the project tree in the left pane of the window. All objects are listed under their assigned Locations. Expand the locations node to view objects.

2. Using the deployment tree filter, select **Deployed Objects.**

This limits the objects displayed on the deployment tree to only objects that have been deployed.

3. Select a mapping or process flow from the deployment tree.

   The selected object displays in Detail tab on the right side of the Deployment Manager.

   > **Note:** Only one mapping or process flow can be executed at a time.

4. From the File menu select **Execute,** or click the Execute button on the toolbar.

   The Execution Dialog displays.

5. Specify the Run Name and Parameters for the execution.

6. Click **OK.**

   The execution is performed and the results display.

7. Review the results and click **OK.**

   This completes the execution and you can proceed to execute other mappings or process flows.

Using Other Tools to Execute Process Flows

With Warehouse Builder, you have two main options for executing process flows: you can execute them from within Warehouse Builder using the Deployment Manager as described earlier, or you can execute them from Oracle Workflow. In addition, you can use Warehouse Builder to integrate with Oracle Enterprise Manager to schedule these process flow executions.

For information about Oracle Workflow, see the *Oracle Workflow Guide.* For information about Oracle Enterprise Manager, see the *Oracle Enterprise Manager Administrator's Guide.*

You can also execute process flows from SQL*Plus.

Executing Mappings and Process Flows Using SQL*Plus

In addition executing deployed objects using the Deployment Manager, you can also use SQL*Plus. In order to do this, you can use a script provided withWarehouse Builder called sqlplus_exec_template.sql. This script is located in the following location: `<OWB home>\owb\rtp\sql`

To execute Warehouse Builder generated scripts in SQL*Plus:

1. Open a SQL*Plus session and run the sqlplus_exec_template.sql script from the SQL prompt using the following syntax:

   ```
   @sqlplus_exec_template.sql rt_owner location_name {PLSQL
   |SQL_LOADER | PROCESS} task_name system_params custom_params
   ```

2. Use the following definitions to determine what value to use for each variable in the script syntax:

   - `rt_owner:` The runtime repository owner.
   - `location_name:` For PL/SQL mappings or Process Flows, use the name of the location you used to deploy the mappings. For SQL*Loader mappings, you must set this value to PlatformSchema. This is a case-sensitive variable.

- `{PLSQL | SQL_LOADER | PROCESS}`: Choose one of these to define the object type.
- `task_name`: Name of the mapping or process flow.
- `system_params`: Comma separated, enclosed by double quotes; comma and `\` can be escaped by `\`.
- `custom_params`: If you have mapping input parameters, this is where they need to be defined. Use the same format as `system_params`.

3. In order to use this script to run multiple mappings, you need to run this same script multiple times.

   For example, the following would be used to execute a PL/SQL mapping called *MY_MAPPING* from the location *MY_WAREHOUSE* with the runtime owner being *MY_RUNTIME* and with no system or custom parameters:

   ```
   @sqlplus_exec_template.sql MY_RUNTIME MY_WAREHOUSE PLSQL MY_
   MAPPING "," ","
   ```

   The following would be used to execute a SQL*Loader mapping called *MY_LOAD* with the runtime owner being *MY_RUNTIME* and with no system or custom parameters. Note that you do not define the location_name as the location you used during deployment. You must use the case-sensitive variable `PlatformSchema`.

   ```
   @sqlplus_exec_template.sql MY_RUNTIME PlatformSchema SQL_
   LOADER MY_LOAD "," ","
   ```

Scheduling Mappings and Process Flows

In addition to running mappings and process flows from Warehouse Builder, you can use Warehouse Builder to create jobs in Oracle Enterprise Manager (OEM) that you can schedule to run at specific times or schedules.

Before you can begin scheduling, the following must be complete:

- An Oracle Enterprise Manager (OEM) repository has been created.
- Preferred credentials have been set, both on the node and on the database.
- You must be logged on as an OEM user that has operating system privileges to run batch jobs.
- The OEM Intelligent Agent must be running on the node that has the Runtime Platform Service installed.
- The Oracle Management Server (OMS) must have been started.
- The mappings and process flows you want to schedule have been deployed.

To schedule a mapping or process flow in OEM:

1. Open Warehouse Builder and verify the Runtime Repository Connection details. You must be connecting as a runtime repository user, and logging into the runtime repository owner.
2. Open the Deployment Manager and verify the location, name, and deployment status of the objects you want to schedule.

   The deployment status should be Successful.
3. Start the Enterprise Manager Console and login to the Oracle Management Server as a superuser. Sysman is the default superuser.

4. From the **Job** menu, select **Create Job** or press **Control+J.**

   The Create Job window opens and displays 5 tabs.

5. Specify the name of the Job on the General tab and then select a target type and then a specific target from the available targets.

   Use the Add button to move the selected target to the Selected Targets column.

6. Select the Tasks tab and select Run SQL*Plus Script from the Available Tasks list. Use the Add button to move it over.

7. Select the Parameters tab and provide the parameters.

   You can either import the script `oem_exec_template` that is provided as part of the Oracle Warehouse Builder or you can copy and paste the content of the script.

8. Import a script oem_exec_template that is shipped as part of Oracle Warehouse Builder.

   The script is located in: `<OWB home>\owb\rtp\sql`

9. Browse to the folder and open the script. The script will appear in the Script Text field. The mandatory parameters for the script are as follows:

   - Runtime Repository Owner
   - Target location
   - Target Type: either one of the PL/SQL, SQL_LOADER, or PROCESS.
   - Task name (name of the mapping/process flow)
   - System parameters (comma separated, enclosed by double quotes; comma and \ can be escaped by \)
   - Custom parameters (see system parameters)

   The script has to run under the runtime access user. These can be set as preferred credentials. Make sure you have the right credentials.

10. Submit the Job directly, add it to the job library to be scheduled in the future, or do both.

14 Auditing Deployment and Execution

When you use Warehouse Builder to deploy and execute scripts, data about each deployment and execution is stored in a Runtime Repository. This data can be accessed by viewing reports in the Runtime Audit Browser from the client or from Oracle Portal.

This chapter includes the following topics:

Why Audit Deployments and Executions?

Auditing deployment and execution information can provide valuable insight into how your target is being loaded and what you can do to further optimize mapping and process flow settings and parameters. Also it can give you immediate feedback on deployment information to make you aware of the deployment history of an object.

These reports 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

These reports, available through the Runtime Audit Browser, are built using the Warehouse Builder Public Views and can access all data stored within the Runtime Repository. You can use the pre-built reporting capabilities of the Runtime Audit Browser to view this data, or you can use the Public Views to create your own custom reports. For more information about Public Views, see Appendix D, "Warehouse Builder Public Views".

About the Runtime Audit Browser

Warehouse Builder Runtime Audit Browser is a browser-based reporting tool that you can use to view detailed information about past deployments and executions. Reports are generated from data stored in the Runtime Repositories.

Warehouse Builder provides two versions of this reporting tool:

- Client Browser Version
- Oracle Portal Version

The reports and functionality provided in both of these tools are the same. The only difference is The Client Browser Version is installed with the Warehouse Builder client and can be invoked from the same menu as the client. The Oracle Portal Version does not require a client installation, however, does require installation of the Runtime Audit Browser on an application server running Oracle Portal.

For information about setting up the Runtime Audit Browser and choosing a version, see Oracle Warehouse Builder Installation and Configuration Guide.

Starting the Client Browser Version

To start the client browser version:

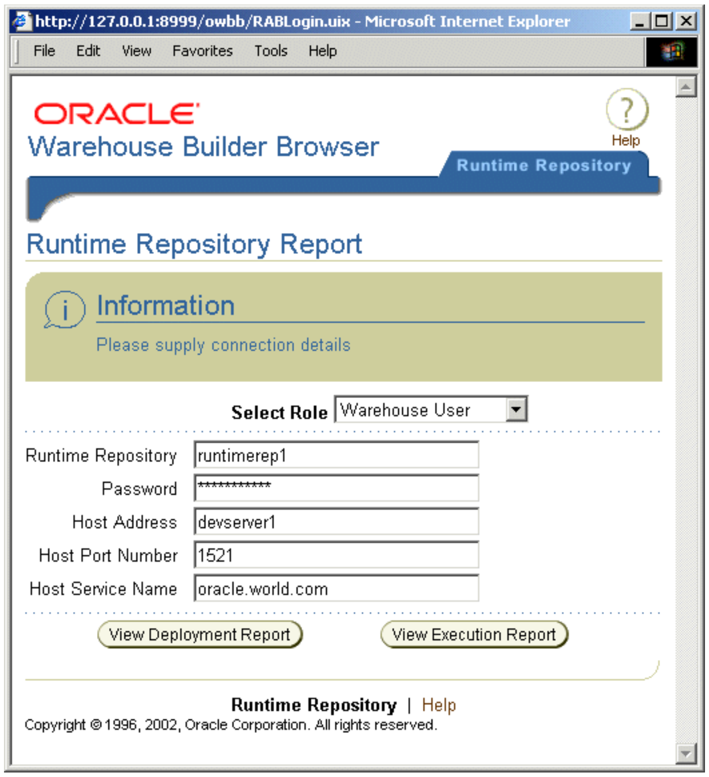
1. From the **Start** menu, select **Programs,** then **Oracle9i Developer Suite,** then **Warehouse Builder,** and then **Start OWB OC4J Instance.**

   The OC4J instance is initialized.

2. From the **Start** menu, select **Programs,** then **Oracle9i Developer Suite,** then **Warehouse Builder,** and then **OWB Runtime Audit Browser.**

   The Warehouse Builder Browser opens and displays the connection information page for a Runtime Repository as shown in Figure 14–1. If you receive an error message, the Warehouse Builder Browser window may have been configured incorrectly. Refer to the Oracle Warehouse Builder Installation and Configuration Guide for instructions on configuring for the Warehouse Builder Runtime Audit Browser.

Figure 14–1 Client Browser Runtime Connection


http://127.0.0.1:8999/owbb/RABLogin.uix - Microsoft Internet Explorer
File Edit View Favorites Tools Help
ORACLE
Warehouse Builder Browser
Help
Runtime Repository
View Deployment Report
View Execution Report

Runtime Repository Report

Information

Please supply connection details

Select Role: Warehouse User

Runtime Repository	runtimerep1
Password	***********
Host Address	devserver1
Host Port Number	1521
Host Service Name	oracle.world.com

Runtime Repository | Help

Copyright © 1996, 2002, Oracle Corporation. All rights reserved.

3. Select a role from the drop-down menu.

   For more information, see "Selecting a Role" on page 14-4.

4. Specify the Runtime Repository connection information and click either **View Deployment Report** or **View Execution Report.**

   The connection information is verified and the Runtime Repository report displays. For information about using the Runtime Audit Browser after connecting to the repository, see "Viewing Runtime Repository Reports" on page 14-5.

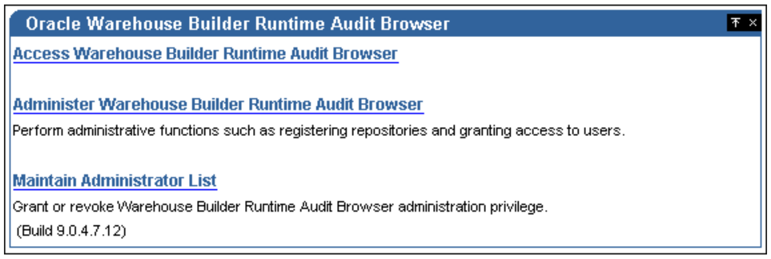
Starting the Oracle Portal Version

To start the Oracle9iAS Portal version:

1. Open a browser and connect to Oracle9*i*AS Portal and login as a user with access to Runtime Audit Browser portlets.

2. Navigate to the page that contains the Runtime Audit Browser Portlet as shown in Figure 14–2.

Figure 14–2 Runtime Audit Browser Portlet


Oracle Warehouse Builder Runtime Audit Browser

Access Warehouse Builder Runtime Audit Browser

Administer Warehouse Builder Runtime Audit Browser

Perform administrative functions such as registering repositories and granting access to users.

Maintain Administrator List

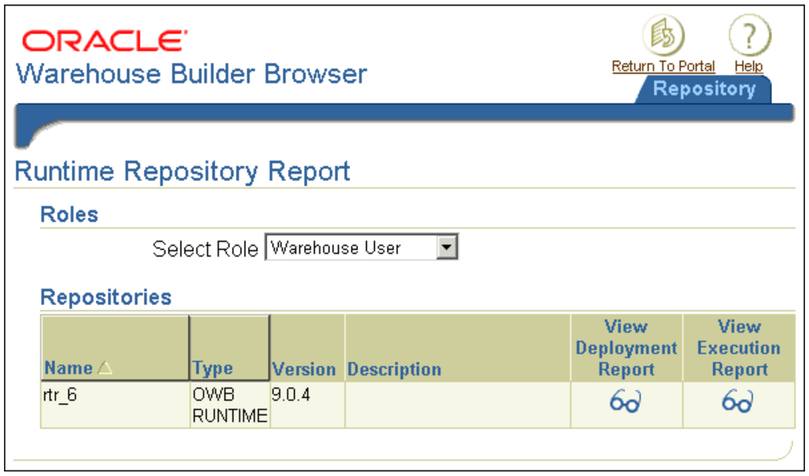
Grant or revoke Warehouse Builder Runtime Audit Browser administration privilege.

(Build 9.0.4.7.12)

3. Select the Access Warehouse Builder Runtime Audit Browser link.

   The Runtime Repository Report page, as shown in Figure 14–3, displays a list of repositories that you can access.

Figure 14–3 Runtime Repository Report Page



Runtime Repository Report

Roles

Select Role: Warehouse User

Repositories

Name	Type	Version	Description	View Deployment Report	View Execution Report
rtr_6	OWB RUNTIME	9.0.4		6ò	6ò

4. Select a role from the drop-down menu.

   For more information, see "Selecting a Role" on page 14-4.

5. Locate the Runtime Repository you want to access and click one of the icons under either the View Deployment Report or View Execution Report columns.

   The report you select displays in the next screen.For information about using the Runtime Audit Browser after connecting to the repository, see "Viewing Runtime Repository Reports" on page 14-5.

Selecting a Role

When you logon to the Runtime Audit Browser you must select a role. The information and functions available from within the Runtime Audit Browser differ depending on the role you select.

You can choose from one of the following three roles:

- QA User
- Warehouse User
- Warehouse Engineer

All roles have access to viewing all of the Runtime Audit Browser reports, however, the data and available functions within certain reports differ. Table 14–1 describes the roles and their main differences.

Table 14–1 Role Descriptions

Role	Description
QA User Role	If the you select this role, the Purge button is enabled, and you can view source values in error diagnostic data from the Run Error Diagnostic report.
Warehouse User or Warehouse Engineer Roles	If you select one of these roles, the Purge button is disabled, and source values are shown as (not available) from the Run Error Diagnostic report.

Viewing Runtime Repository Reports

Use the Runtime Audit Browser to view the deployment and execution auditing information stored in the Runtime Repository. After you have connected to a Runtime Repository from either the client or Oracle Portal version, select a deployment or execution report from the available repositories.

Viewing Deployment Reports

The deployment reports contain audit data about specific deployments. When you first access a Runtime Repository from the portlet, you are shown a broad report of the entire Runtime Repository. You can then drill down and focus on reports that contain the specific information you want to view.

Use the following tabbed sections to view details:

- Deployment Schedule
- Object Summary
- Location

Deployment Schedule

The Deployment Schedule tabbed section enables you to search for deployment data based on the time of the deployment. Use the Filter Options section to filter for deployments that occurred during a specific data range. You can then drill into a specific deployment and view object information. You can also use the icon under the Focus column to view only the deployment details for a row.

Object Summary

The Object Summary tabbed section enables you to view deployment data based on the type of object. When you first select this tab for the entire Runtime Repository a comprehensive list of all objects display. Use the Filter Options section to filter by object type and deployment status.

The following information displays:

- Name
- Type
- Location
- Latest Deployment
- Object Status

You can then drill down on a specific object or location to view more details.

Location

The Location tabbed section enables you to view deployment data based on the location. When you first select this tab for the entire Runtime Repository a comprehensive list of all registered locations display.

The following information displays:

- Name
- Type
- Type Version

- Latest Deployment

You can then drill down on a specific location to view more details.

Viewing Execution Reports

The execution reports contain audit data about specific executions. When you first access a Runtime Repository from the portlet, you are shown a broad report of the entire Runtime Repository. You can then drill down and focus on reports that contain the specific information you want to view.

Use the following tabbed sections to view details:

- Execution Schedule
- Execution Summary
- Execution
- Trace

Execution Schedule

The Execution Schedule tabbed section enables you to search for execution data based on the time of the execution. Use the Filter Options section to filter for executions that occurred during a specific data range. You can also filter for a specific mapping or process flow.

You can then drill down on a specific execution and view information. You can also use the icon under the Focus column to view only the e details for a row.

Execution Summary

The Execution Summary tabbed section enables you to view execution data based on a mapping or process flow. When you first select this tab for the entire Runtime Repository a comprehensive list of all objects display. Use the Filter Options section to filter by object and execution status.

The following information displays:

- Name
- Type
- Latest Execution
- Execution Status
- Execution Report Icon

Execution

When you click the Execution Report icon, the Execution tabbed section of the Execution Report displays. An Execution Report is provided for each execution of a mapping or process flow.

Use the Execution tabbed section of this report to view the following information:

- Execution Details
- Execution Parameters
- Step Details
- Error Messages

- Audit Details

Trace

The Trace tabbed section of the Execution Report contains detailed trace information about a specific execution of a mapping or process flow.

The following information is provided:

- Rowkey
- Severity
- Source/Target
- Table Name
- Action
- View Diagnostic Report

Purging Audit Data

From time to time you may want to purge data from the Runtime Repository. You can do this by using the Purge button on each of the report pages. This function is useful because it limits the amount of data held in audit tables and makes it easier to find relevant data and maintain performance. To use the Purge buttons you must be logged on as a QA User. The Purge button is disabled if you are logged on as a Warehouse User or Warehouse Engineer.

Note: If you are running mappings with the Audit Level configuration parameter set to Complete, this will generate a large amount of diagnostic trace data. This will cause the tablespace to become full. In this case, use the Purge Error and Trace button.

Runtime Audit Browser Administration

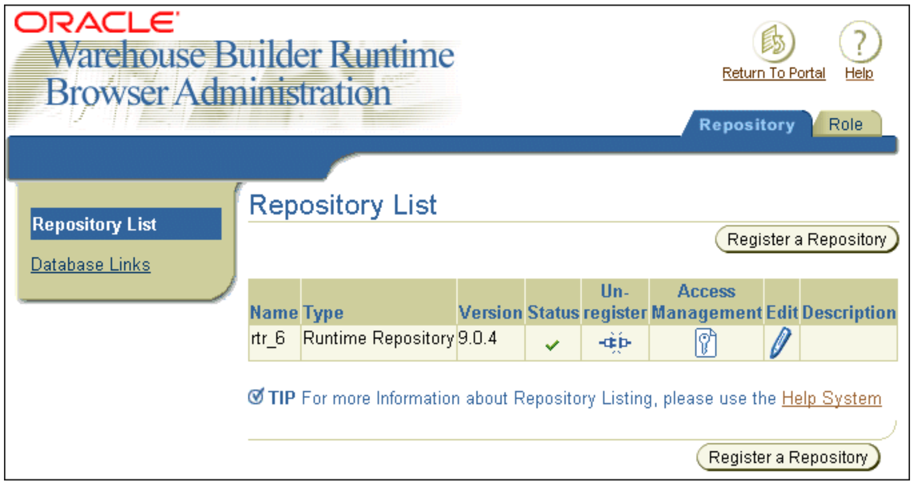
Runtime Audit Browser administration functions are only available using the Oracle Portal version. Access the following functions from the **Administer Warehouse Builder Runtime Audit Browser** and **Maintain Administrator List** links from the Runtime Audit Browser Portlet:

- Managing Available Repositories: Enables you to view connection information for available repositories as well as register new repositories.
- Administering Roles: Enables you to define which user roles and access privileges.
- Maintaining the Administrator List: Enables you to define which users are granted administrator level privileges.

Managing Available Repositories

Use the Repository List page, shown in Figure 14–4, to view the list of all registered Runtime Repositories. All repositories must be registered before you can access the audit reports.

Figure 14–4 Repository List Page



Repository List

Name	Type	Version	Status	Un-register	Access Management	Edit	Description
rtr_6	Runtime Repository	9.0.4	✓				

TIP For more Information about Repository Listing, please use the Help System

To unregister a repository:

- Locate the name of the repository that you want to un-register and click the icon under the Un-register column.

  The repository is immediately un-registered and the page refreshes to display the updated list.

To manage access to a repository:

- Locate the repository and click the icon under the Access Management column.

  The Access Management displays on the next page. Using this you can add or revoke access to a repository by user or group. Additional help and instructions are provided on that page.

To edit repository information:

- Locate the repository and click the icon under the Edit column.

  The Edit Repository page displays next. Additional help and instructions are provided on that page.

To register a repository:

- Click **Register a Repository.**

  The Register a Repository page displays on the next page. Be prepared to provide connection details for the repository. Additional help and instructions are provided on that page.

Note: Before you can register a repository, you must have already defined the database link.

To add or view database links:

- Click the Database Links link on the left column.

  The Database Link page displays next. Additional help and instructions are provided on that page.

Registering a Repository

Use the Register a Repository page to register Runtime Repositories. You can only view audit data from registered Runtime Repositories.

To register a repository:

1. Type the name of the Runtime Repository.
2. Specify the database link, or click the flashlight or torch icon to select from a list of previously created database links.
3. Type an optional description.
4. Click **Apply** to register the repository.

   The Repository List page displays next with an updated list of registered repositories.

Editing Repository Definitions

Use the Edit Repository page to edit the repository registration information. You can only edit one Repository at a time.

You can edit the following fields:

- Name
- Database Link
- Description

Click **Apply** to save edits. The Repository List page displays next.

Managing Database Links

Use the Database Links page to view, create, edit, and drop database links.

To create a database link:

- Click **Create Database Link** located under the list of database links.

  The Create Database page displays next. Be prepared to provide detailed connection information. Additional help and instructions are available on that page.

To drop a database link:

- Locate the name of the database link that you want to drop and click the icon under the Drop column.

  The database link is immediately dropped and the page refreshes to display the updated list.

To edit a database link:

- Locate the database link and click the icon under the Edit column.

  The Edit Database Link page displays next. Additional help and instructions are provided on that page.

Creating Database Links

Use the Create Database Links page to specify connection information for new database links you are creating.

To create a database link specify the following information:

- Database Link Name
- Warehouse Builder Repository User Name
- Warehouse Builder Repository User Password
- Host Address
- Host Service Name
- Host Protocol
- Host Port Number

Click **Apply** to submit the database link information. If the information is valid, the Database Links page displays next an updated list database links.

Editing Database Links

Use the Edit Database Link page to view details about a specific database link and make changes.

You can edit the following fields:

- Warehouse Builder Repository User Name
- Warehouse Builder Repository User Password
- Remote Database Information

Click **Apply** to save edits. The Database Links page displays next.

Viewing the Database Link Error Status Page

Use the Database Link Error Status page to view details about the error. This is for viewing purposes only.

The following information is provided:

- **Name:** Name of the database link.
- **Create On:** Timestamp for when the database link was created.
- **Status:** Error
- **Description:** Reason why the database link is not valid.

Administering Roles

Use the Role Administration page to define access by role type. Role types determine what how the audit information is displayed in the reports. For more information about roles, see "Selecting a Role" on page 14-4.

There are three role types available:

- QA User
- Warehouse User
- Warehouse Engineer

To manage access to role types:

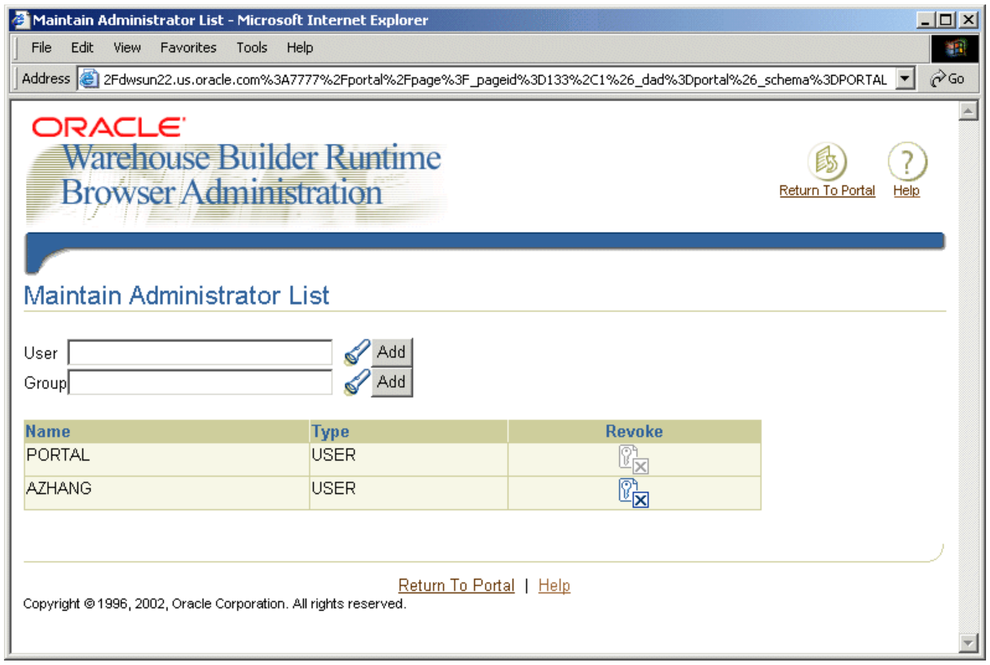
- Choose a role type and select the icon under the Access Management column.

  The Access Management page displays next. From there you can define which user and groups you want to grant or revoke access.

Maintaining the Administrator List

Use the Maintain Administrator List page, shown in Figure 14–5, to add or revoke administrator privileges to users and groups.

Figure 14–5 Maintain Administrator List


Maintain Administrator List - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Go
ORACLE
Warehouse Builder Runtime Browser Administration
Return To Portal
Help
User
Group
Add
Add

Address: 2Fdwsun22.us.oracle.com%3A7777%2Fportal%2Fpage%3F_pageid%3D133%2C1%26_dad%3Dportal%26_schema%3DPORTAL

Maintain Administrator List

Name	Type	Revoke
PORTAL	USER	☒
AZHANG	USER	☒

Return To Portal | Help

Copyright © 1996, 2002, Oracle Corporation. All rights reserved.

Note: You must have administrator privileges to make changes to the list of administrators.

To add a user or group:

- Type in the name of a user or group in the open fields and click **Add.** You can also search for a user or group by selecting the flashlight or torch icon.

  If the user or group name you provided is valid, it is added to the list of administrators.

To revoke privileges from a user or group:

- Locate the name of the user or group and click the icon under the Revoke column.

  The user or group is removed from the list of administrators.

Available Runtime Audit Reports

The following tables list the audit reports available from the Runtime Audit Browser:

- Table 14–2, " Warehouse/Repository Reports" on page 14-12
- Table 14–3, " Process Reports" on page 14-12
- Table 14–4, " Process Run Reports" on page 14-12
- Table 14–5, " Mapping Reports" on page 14-13
- Table 14–6, " Mapping Execution Reports" on page 14-13

- Table 14–7, " Execution Error Reports" on page 14-13
- Table 14–8, " Data Object Reports" on page 14-13
- Table 14–9, " Location Reports" on page 14-13
- Table 14–10, " Deployment Error Reports" on page 14-14
- Table 14–11, " Management Reports" on page 14-14

Warehouse/Repository Reports

Table 14–2 Warehouse/Repository Reports

Report	Description
Deployment Schedule	Displays a summary of all the deployment actions that have occurred for the selected repository. This is displayed as a Calendar view based initially on the current date. This report contains hyperlinks to the Deployed Objects.
Execution Schedule	Displays all execution processes, in date/time order, and a summary of their contained tasks. This is displayed as a Calendar view based initially on the current date.This report contains hyperlinks to Process, Map, Process Run and Map Run.
Execution Summary	Displays summary details of all processes in alphabetical or run-date/time order. It includes the Process Run and Mapping Run details. This report contains hyperlinks to Process, Map, Process Run and Map Run.
Object Summary	Displays a summary of all the objects that have been deployed in the selected repository. It displays details of the last deployment of each object. This report contains hyperlinks to the Processes, Maps, and Data Objects that have been deployed.
Location	Displays a list of all locations into which objects have been deployed. This report contains hyperlinks to the various locations.

Process Reports

Table 14–3 Process Reports

Report	Description
Process Execution	Displays summary details of the Process Runs. This report contains hyperlinks to the Process Run report.
Process Deployment	Displays details of the deployment history. This report contains hyperlinks to the sub-processes and maps that it contains and any deployment errors that occurred.

Process Run Reports

Table 14–4 Process Run Reports

Report	Description
Process Run Execution	Displays execution details, return status and runtime parameters, and a summary of its contained Mapping runs. This report contains hyperlinks to the Process, Map Run and Run Errors reports.

Mapping Reports

Table 14–5 Mapping Reports

Report	Description
Map Execution	Displays summary details of its Map Runs. This report contains hyperlinks to the Map Runs report.
Map Deployment	Displays details of its deployment history. This report contains hyperlinks to any Deployment Errors that occurred.

Mapping Execution Reports

Table 14–6 Mapping Execution Reports

Report	Description
Map Run Execution	Displays execution details, return status and runtime parameters. This report contains hyperlinks to the Run Errors.
Map Run Trace	Displays a diagnostic trace of source and target data values used. This can be useful for diagnostic purposes. It is only available if the Map Run was executed with the appropriate Audit Level parameter setting.

Execution Error Reports

Table 14–7 Execution Error Reports

Report	Description
Error Diagnostics	Displays details of Run Error messages encountered for a given mapping or process flow execution.

Data Object Reports

Table 14–8 Data Object Reports

Report	Description
Data Object Deployment	Displays details of the deployment history for a specific object. This report contains hyperlinks to any Deployment Errors that occurred.

Location Reports

Table 14–9 Location Reports

Report	Description
Deployment Schedule	Displays a summary of all the deployment actions that have occurred for the selected location. This is displayed as a calendar view initially based on the current date. This report contains hyperlinks to the deployed objects.
Object Summary	Displays a summary of all the objects that have been deployed in the selected location. It displays details of the last deployment of each object. This report contains hyperlinks to the processes, maps, and data objects that have been deployed.

Deployment Error Reports

Table 14–10 Deployment Error Reports

Report	Description
Error Details	For a given Deployment Error this shows details of all related error, warning and information messages that occurred in the deployment.

Management Reports

Table 14–11 Management Reports

Report	Description
Service Node Report	Displays the current status of cluster nodes for the runtime repository. This report shows whether the nodes are in use or not.

Part IV

Managing Metadata

This part contains the following chapters:

15 Importing and Exporting with the Metadata Loader (MDL)

The Metadata Loader (MDL) enables you to populate a new repository as well as transfer, update, or restore a backup of existing repository metadata. You can also take snapshots of your metadata and use them for backup, compare, and restore purposes.

This section contains the following topics:

Overview of Import and Export Using Metadata Loader

Warehouse Builder provides several features that enable you to copy and move metadata for the purposes of backup, history management and version management.

You can import and export metadata for any type of object on the navigation tree using the Metadata Loader (MDL) utility. Access MDL through the Warehouse Builder client or through the OMB Plus scripting interface. Use the import and export functionality to backup metadata or to migrate metadata when upgrading Warehouse Builder.

You can also then move exported files into a third-party version control tool such as Oracle Repository, ClearCase, or SourceSafe. If you enter version numbers in your project properties, it is easier to track your export and import versions in this setting.

You can also perform metadata change management by taking snapshots of your metadata using the OMB Plus scripting interface. Snapshots enable you to capture definitions of metadata objects using Warehouse Builder scripts. Use snapshots for metadata backup and version management. For more information about metadata change management, see Chapter 16, "Metadata Change Management".

Importing and Exporting Metadata Using the Metadata Loader

The MDL enables you to copy or move metadata objects between repositories, even if those repositories reside on platforms with different operating systems.

The MDL consists of two utilities: metadata export and metadata import. The export utility extracts metadata objects from a repository and writes the information into a text file. The import utility reads the metadata information from an exported text file and inserts the metadata objects into a repository. MDL uses its own format, and the MDL import utility only reads files of MDL format (files created by MDL Export).

You can operate the MDL from the Warehouse Builder console or by using a command-line interface. For instructions on using MDL through the command-line interface, refer to "Using the Metadata Loader Command Line Utility" on page 15-20. If you use the console menu, a graphical interface guides you through the export or import processes.

Use the metadata loader to perform any of the following tasks:

- **Backup metadata:** The MDL is an important part of your disaster recovery strategy. You can export a file with your existing repository metadata as a backup and use that exported file to restore a repository if necessary.
- **Seed a new repository:** You can export data from an existing repository and use it as the basis for a new repository.
- **Migrate a repository:** You can export metadata to a file and then re-import it. This is commonly done when upgrading to a newer version of Warehouse Builder. For details on upgrading from a previous version of Warehouse Builder, see the Oracle Warehouse Builder Installation and Configuration Guide.
- **Copy metadata:** A multiple user development environment can result in multiple copies of the same metadata. The MDL makes it easy to load single set metadata into more than one repository.

This section contains the following topics:

- Required Access Privileges for MDL on page 15-2
- About Metadata Loader Results on page 15-3
- About the Metadata Loader Log File on page 15-3

Required Access Privileges for MDL

The Warehouse Builder repository allows multiple clients to access the same repository schema concurrently. Warehouse Builder uses locks to allow only one client to access to change repository objects. While an object is locked, other clients can only view it as it existed after the last transaction instigated by any user is committed.

Tip: To ensure that you are exporting the most up-to-date metadata, you need to be the sole client accessing the repository.

If you click **OK** when prompted, the MDL commits changes made to the repository after a successful metadata import (any import with no error messages, including imports with only information or warning messages). The MDL also executes a rollback after an unsuccessful import. This means that Warehouse Builder attempts to acquire one lock for each primary object (an object in the first level on the navigation tree) in the repository that matches an object in the MDL file. These objects include, but are not limited to, projects, modules, and tables—individual columns are not locked. Therefore, you must be able to hold the locks for these objects while you import metadata. If other users hold locks for objects to which you are importing, the MDL will fail.

Tip: To ensure a successful metadata import, you need to be the sole client accessing the repository.

If the MDL import affects too many objects in the repository, the MDL automatically switches to single user mode. This means that no other users can log on to the repository until after the MDL import completes. Single-user mode allows the MDL to avoid the performance degradation that results from using a large number of locks. In single-user mode, the MDL is less likely to deplete the repository enqueue resources. If other users are logged into this repository when MDL attempts to switch to single-user mode, MDL cannot switch to single-user mode and subsequently fails.

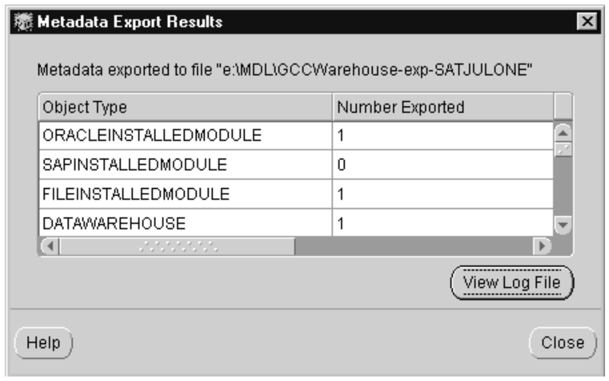
You also need to have MDL_IMPORT security privileges in order to import metadata. For more information on security, see "Managing Security with PL/SQL" on page 19-5.

About Metadata Loader Results

Each time you use the export or import utilities, the MDL reports the results of an action and writes diagnostic and statistical information to a log file.

The MDL reports the results after any import or export task with a dialog. If you want detailed information, you can view a detailed log by clicking **View Log File** from the Metadata Export Results dialog as shown in Figure 15–1.

Figure 15–1 Metadata Export Results



Metadata exported to file "e:\MDL\GCCWarehouse-exp-SATJULONE"

Object Type	Number Exported
ORACLEINSTALLEDMODULE	1
SAPINSTALLEDMODULE	0
FILEINSTALLEDMODULE	1
DATAWAREHOUSE	1

The results dialog lists the metadata objects found in either the file or the repository, and the number of each object that was exported or imported. You can use the results dialog to ensure that all of the objects were exported or imported. The MDL identifies the objects that were exported or imported and compares it with the eligible objects list. A zero in the Number Exported or Number Imported column for any object indicates that the MDL found no object of that type in the repository. However, if a zero appears for any object that exists in the repository or imported MDL file, then MDL encountered a problem when importing or exporting that object.

About the Metadata Loader Log File

Whenever you export or import repository metadata, the MDL writes diagnostic and statistical information to a log file. By default, the log file is located in the directory and path specified in the Message Log tab located in the Preferences dialog. You can specify an alternative location for the log file when invoking MDL.

Example 15–1 displays the contents of a typical import log file.

Example 15–1   Log File Showing Import Results

```
Import started at 04/25/2001 4:59:46 PM
****************************************************************************
*  Import for OWB Release: 3.0.0.0.0   Version: 3.0.0.3.0
*  User: user30_3i   Connect String: epaglina-pc:1521:ora8i
*  Data File: d:\owb3000\sco_dim_time_phy_m_tgt.mdl
*  Log File: d:\owb3000\imp_dim_time_phy_m_tgt.log
*  Trace: B
*  Trace File: d:\owb3000\imp_dim_time_phy_m_tgt.trc
*  Physical Names: Y   Mode: CREATE
*  Ignore Universal Identifier: Y   Commit At End: Y
****************************************************************************
Informational at line 15: MDL-1207 PROJECT with physical name <PRJ_Dimension> not imported
because it already exists.
Informational at line 21: MDL-1207 DATAWAREHOUSE with physical name <WH> not imported because
it already exists.
Informational: MDL-1134 COMMIT issued at end of import data file.

Counts for OWB Import Utility
-----------------------------

Total Projects Processed by Import = 1

-------------------------------

Project = PRJ_Dimension

Entity in Project                   Added     Replaced    Skipped
-------------------------------     -----     --------    -------
DATAWAREHOUSE:                        0           0          1
DIMENSION:                            1           0          0
LEVEL:                                3           0          0
HIERARCHY:                            2           0          0
LEVELRELATIONSHIP:                    4           0          0
COLUMN:                              18           0          0
UNIQUEKEY:                            8           0          0
PRIMARYKEY:                           2           0          0
CONFIGPARAM:                         60           0         60
CHILDCONFIG:                         14           0          0

Import ended at 04/25/2001 4:59:52 PM
```

The log file enables you to monitor and troubleshoot export and import activities in detail. The log file contains the following types of status messages:

- **Informational:** Provides information about the import or export, such as missing metadata objects, whether or not objects were imported, and any reasons why they were not imported or exported.
- **Warning:** Cautions you about the import or export of an object but does not indicate a failed or aborted load. A warning notifies you of the possibility of unexpected load results for the load.
- **Error:** Indicates that the MDL export or import was aborted and did not complete successfully. The error message offers a brief reason for the failure.

This log also displays the total number of objects that have been added, replaced, and skipped. A zero in any column for any object indicates that the MDL found no object of that type in the repository or in the imported MDL file.

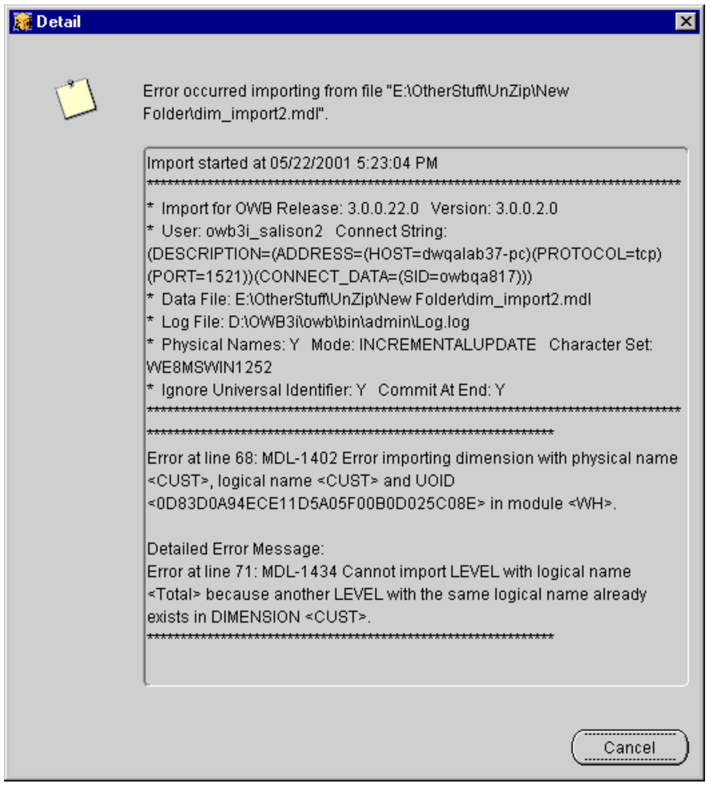
Detailed Error Logs

If you are running an MDL Import and encounter an error, an error message displays.

Click **Detail** to display a detailed error log that lists the repository object and the object line in which the error occurred. Detailed messages are useful whenever you import metadata into repositories with existing metadata because they alert you to problems such as improperly defined metadata objects and object duplication.

Figure 15–2 is an example of a detailed error message.

Figure 15–2 Detailed Error Message



Error occurred importing from file "E:\OtherStuff\UnZip\New Folder\dim_import2.mdl".

```
Import started at 05/22/2001 5:23:04 PM
********************************************************************************
* Import for OWB Release: 3.0.0.22.0  Version: 3.0.0.2.0
* User: owb3i_salison2  Connect String:
(DESCRIPTION=(ADDRESS=(HOST=dwqalab37-pc)(PROTOCOL=tcp)
(PORT=1521))(CONNECT_DATA=(SID=owbqa817)))
* Data File: E:\OtherStuff\UnZip\New Folder\dim_import2.mdl
* Log File: D:\OWB3i\owb\bin\admin\Log.log
* Physical Names: Y  Mode: INCREMENTALUPDATE  Character Set:
WE8MSWIN1252
* Ignore Universal Identifier: Y  Commit At End: Y
********************************************************************************
*************************************************************

Error at line 68: MDL-1402 Error importing dimension with physical name
<CUST>, logical name <CUST> and UOID
<0D83D0A94ECE11D5A05F00B0D025C08E> in module <WH>.

Detailed Error Message:
Error at line 71: MDL-1434 Cannot import LEVEL with logical name
<Total> because another LEVEL with the same logical name already
exists in DIMENSION <CUST>.
*************************************************************
```

In this example, the repository object is the CUST dimension and the imported object is the TOTAL level. The dialog explains that the TOTAL level cannot be imported into the CUST dimension because a level named TOTAL already exists.

Exporting Metadata

The Metadata Loader can export all repository objects. The MDL also exports information belonging to metadata objects such as table columns and their constraints, data loading configuration parameters, and named attribute sets. You can use the MDL to export an entire project or a subset of objects within a project.

When you export repository metadata, the Metadata Loader writes the extracted metadata to a delimited text file. The MDL stores this file outside the repository by assigning a default path and file name to the exported MDL file.

This section contains the following topics:

- Before Exporting Metadata on page 15-6
- About the Metadata Export Utility on page 15-6
- Exporting Metadata using Warehouse Builder Client on page 15-6
- Metadata Export File Format on page 15-8

- Archiving a Project on page 15-8

Before Exporting Metadata

Before you attempt to export metadata, ensure you have the following:

- **Required access privileges.** To ensure that you are exporting the most up-to-date metadata, verify that you are the sole client accessing the repository in read/write mode. For more details, see "Required Access Privileges for MDL" on page 15-2.
- **Sufficient disk storage.** If you lack sufficient disk space on the machine to which you export the metadata, the export fails. Your destination machine must be able to contain the entire metadata file. The export utility cannot save portions of the metadata file.

About the Metadata Export Utility

You can export metadata from a Warehouse Builder repository using one of the following:

- **Metadata Loader command line utility.** You can use the command line utility to perform tasks additional tasks not available from the client interface. For instructions on exporting from the command line, see "Using the Metadata Loader Command Line Utility" on page 15-20.
- **Warehouse Builder client interface.** For instructions on using the client interface, see "Exporting Metadata" on page 15-5.

Using the command line or the client interface, you can export an entire project, collection, or module, or any subset of objects. If you export a subset of objects, the MDL exports definitions for each object you have selected and the parent objects to which the subset belongs. This enables the MDL to maintain the tree relationships for those objects during metadata import.

For example, if you export a single dimension, the export file contains definitions for:

- The dimension (and its hierarchies and levels)
- The dimension node to which the dimension belongs
- The module to which the dimension node belongs
- The project to which the module belongs

If you are exporting a subset of objects, make sure you export all referenced objects and import them as well. The Metadata Import Utility enables you to import repository objects even if the references for those objects cannot be satisfied.

For example, if you export a cube, the foreign key references will be exported, but the dimensions to which they refer will not. If the metadata in the dimension tables changed, then the foreign key references imported to the new repository would become incorrect.

Exporting Metadata using Warehouse Builder Client

Use the metadata export utility to export objects from a Warehouse Builder repository into an MDL file.

To export metadata from a repository using the Warehouse Builder client interface:

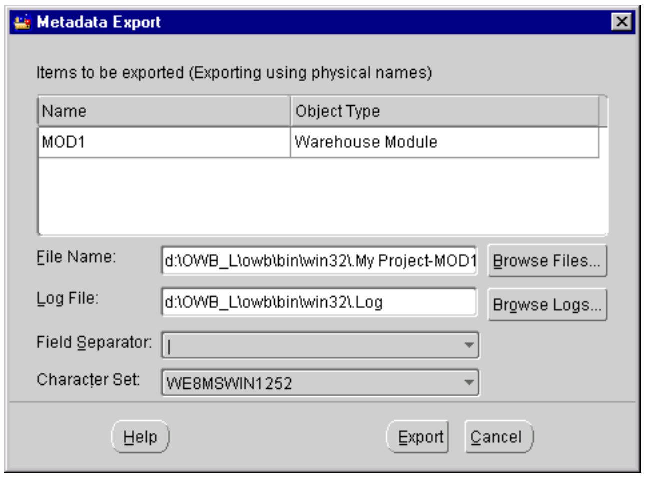
1. From the Warehouse Builder Console, select the object or objects you want to export.

You can export individual objects such as tables or groups of objects. When you export projects nodes, or modules, you also export the objects they contain. When you export collections, you also export the objects they reference.

2. From the **Project** menu, select **Metadata Export** and then **File.**

   The Metadata Export dialog displays the names and types of the objects you are exporting. The Metadata Export dialog also displays default settings for the export file as shown in Figure 15–3.

Figure 15–3 Metadata Export Dialog



Items to be exported (Exporting using physical names)

Name	Object Type
MOD1	Warehouse Module

File Name: d:\OWB_L\owb\bin\win32\My Project-MOD1

Log File: d:\OWB_L\owb\bin\win32\Log

Field Separator: |

Character Set: WE8MSWIN1252

3. You can accept or change the following default settings:

   File Name: Type the name of the export file to create or click **Browse** to locate a directory or file. The filename you assign must end with **.mdl.**

   Log File: Warehouse Builder records information about the export in a log file. You can change this location by entering a new path and filename. Type the path and file name in the field or click **Browse** to locate a directory or filename.

   Field Separator: Table fields in the export file are separated with a pipe (|) by default. If your file already has the pipe (|) symbol as part of its data, you can change the default field separator to a caret (^) by selecting it from the list.

   Character Set: Select the character set to use in the export file. The default character set is defined by the Warehouse Builder client system. Use the list to change the output character set.

4. Click **Export.**

   If you made changes to the repository metadata prior to running the export utility, the Metadata Export Confirmation dialog displays. Click **Commit** to save changes or **Rollback** to revert to the previously saved version. You must have read/write access to the repository in which you are exporting metadata to commit changes.

   The Metadata Export Progress dialog displays the progress. When the export completes, the Metadata Export Results dialog displays.

   Click **View Log File** for a detailed view of the export process.

Metadata Export File Format

The Metadata Loader formats the .mdl export file using keywords and position as shown in Example 15–2.

Example 15–2 Sample Records from an Export File

```
#Project data <PhysicalName> <LogicalName> <UniversalID> <Version Label>
PROJECT|WarehouseName|Warehouse Name|A86184D5336911D58E9000B0D02A59E4|null
#Dimension <PhysicalName> <LogicalName> <UniversalID> <Prefix> <UsageType> <Imported>
<Generated>
DIMENSION|Channels|Channels Dimension Data Mart|7E727655029911D58DC900C04F48E9ED|ch|null|N|N
```

In this example, each record in the file begins with a keyword followed by one or more variable-length fields. Table fields are separated by a pipe (|) by default.

Archiving a Project

Archiving a project enables you to copy metadata stored within a Warehouse Builder repository to an external location for securing that data at a fixed point in time. Warehouse Builder provides an Archive Wizard to assist you in this process. The Archive and Restore utilities initially write to a file system. You can then move files from this file system into a third-party version control tool such as Oracle Repository, ClearCase, or SourceSafe.

Note: The Archive and Restore utilities will be desupported in the next release of Oracle Warehouse Builder .

You must set up your Archive/Restore settings on the Preferences page before you can archive or restore your project. If you attempt to archive or restore without setting these preferences, you get an error.

Project Version Labels

Before you archive your project, you can update the project version label with the Project Properties dialog. There are two places in Warehouse Builder where you can set up the version label used in the archive/restore:

- The first is in the New Project Wizard. The New Project Wizard contains a step that enables you to define version properties. The version label that you set here is the version label that is used when that project is archived.
- After you have created a project, you can edit the version label by opening the Properties dialog for the project. Click the Version Properties tab to modify the project version label.

Differences Between Archive and Export

Archive and Restore are different from Import and Export. Table 15–1 describes the differences between Archive and Export.

Table 15–1 Differences Between Archive and Export

Feature	Archive	Export
Character Set	UTF8	User Configured
Field Separator	Pipe Character (|)	User Configured

Table 15–1 (Cont.) Differences Between Archive and Export

Feature	Archive	Export
Read-only Detection	Detects and prompts you to re-try	Detects and then fails
Dump Format	MDL	MDL
Log File Name	Generated	Generated and User configured
File Location	Configured by preferences in the following structure: `$ARCHIVE_HOME/ project_name/Label/ Archive_Name`	User-defined

Archiving a Project

Before you archive your project, you can update the project version label with the Project Properties dialog (see "Project Version Labels" on page 15-8).

To archive a project:

1. Select **Archive** from the Project menu.

   You can also select **Archive** from the right-click menu when a project is selected.

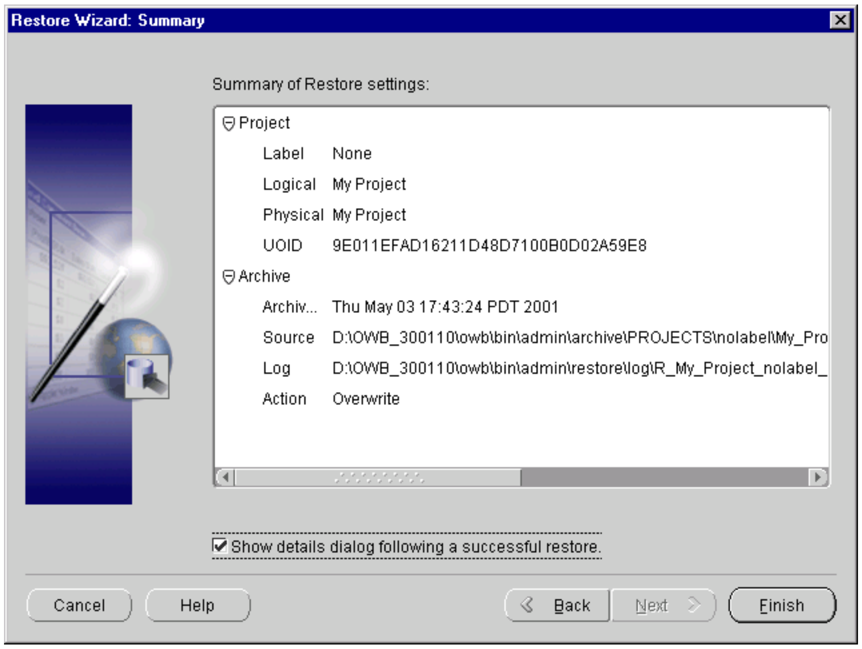
   The Archive Wizard Welcome page displays, as shown in Figure 15–4.

2. Click **Next.**

   The Summary page displays a summary of the archive settings prior to running the archive process. If you want to see the details of your archive after the archive process is complete, check the **Show details dialog following a successful archive** box.

   No changes can be made from the wizard. If you notice an error in the Archive Wizard Summary page, click **Cancel** and make the appropriate changes to your Archive/Restore Preferences before continuing with the archive.

Figure 15–4 Archive Wizard Summary Page



Summary of Restore settings:

Project

Label	None
Logical	My Project
Physical	My Project
UOID	9E011EFAD16211D48D7100B0D02A59E8

Archive

Archiv...	Thu May 03 17:43:24 PDT 2001
Source	D:\OWB_300110\owb\bin\admin\archive\PROJECTS\nolabel\My_Pro
Log	D:\OWB_300110\owb\bin\admin\restore\log\R_My_Project_nolabel_
Action	Overwrite

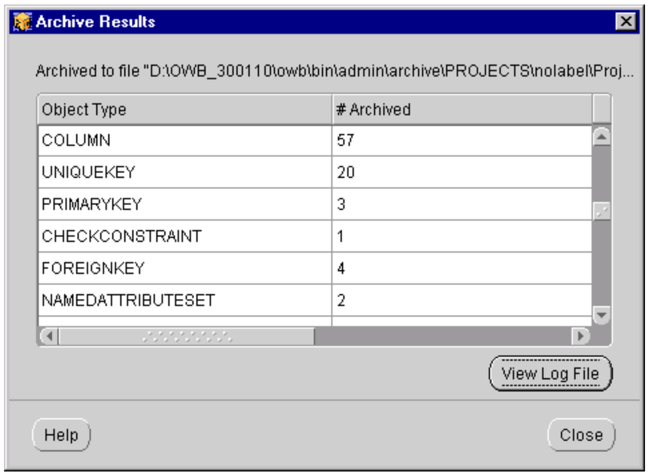
☑ Show details dialog following a successful restore.

3. Click **Finish.**

This begins the archive process. A progress window appears. When the progress bar reaches 100%, the archive process is complete.

If you checked the **Show details dialog following a successful archive box,** the Archive Results dialog displays. This dialog displays the name of each object type and how many of each were archived, as shown in Table 15–5.

Figure 15–5 Archive Results

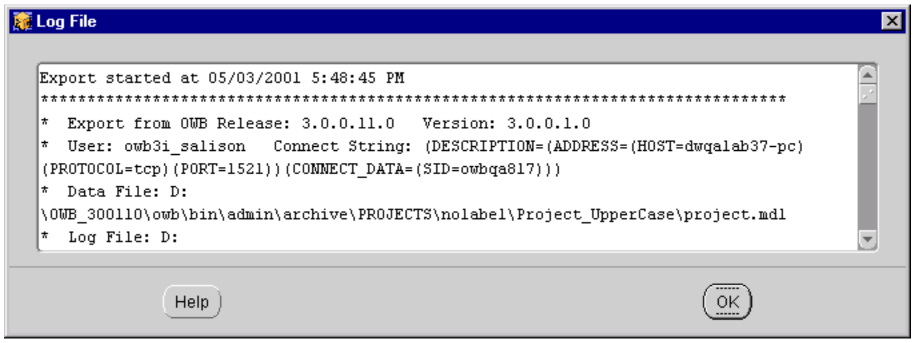


Archived to file "D:\OWB_300110\owb\bin\admin\archive\PROJECTS\nolabel\Proj...

Object Type	# Archived
COLUMN	57
UNIQUEKEY	20
PRIMARYKEY	3
CHECKCONSTRAINT	1
FOREIGNKEY	4
NAMEDATTRIBUTESET	2

For a more detailed look at the archive process, click **View Log File.** This displays the entire log file, as shown in Table 15–6.

Figure 15–6 Archive Log File



```
Export started at 05/03/2001 5:48:45 PM
*******************************************************************************
*  Export from OWB Release: 3.0.0.11.0   Version: 3.0.0.1.0
*  User: owb3i_salison   Connect String: (DESCRIPTION=(ADDRESS=(HOST=dwqalab37-pc)
(PROTOCOL=tcp)(PORT=1521))(CONNECT_DATA=(SID=owbqa817)))
*  Data File: D:
\OWB_300110\owb\bin\admin\archive\PROJECTS\nolabel\Project_UpperCase\project.mdl
*  Log File: D:
```

Importing Metadata

The import utility reads the metadata information from an exported text file and inserts the metadata objects into a repository. The metadata import utility only reads files created by the metadata export utility.

The MDL imports information belonging to exported metadata objects such as table columns and their constraints, data loading configuration parameters, and named attribute sets. You can use the MDL to import objects into a project or a collection.

If you import an .mdl file containing metadata for gateway Modules, such as DB2 or Informix, from an older version of Warehouse Builder, the file may not import the metadata into the corresponding source module folders in a project. The imported files are stored under the Others node in the navigation tree. You need to manually copy the metadata for the gateway modules into the correct source module folders.

The following sections describe how to use the Metadata Import Utility:

- Before Importing Metadata on page 15-11
- About the Metadata Import Utility on page 15-12
- Validation Rules Governing Import on page 15-12
- Importing Metadata using Warehouse Builder Client on page 15-12
- Restoring a Project on page 15-17

Before Importing Metadata

Before you attempt to import metadata, ensure you have the following:

- **Required security privileges:** You need the MDL_IMPORT privilege before you begin an import. For more information on security, see "Managing Security with PL/SQL" on page 19-5.
- **Required access privileges:** Only a user with read/write access can use the metadata loader import utility. Because the import utility is altering the repository, the metadata objects must be locked prior to importing. For more details, see "Required Access Privileges for MDL" on page 15-2.
- **A backup of your current repository:** Consider taking a backup of your existing repository (either in the form of an export or a metadata snapshot) before attempting a large or complex import. For more information on exporting metadata, see "Exporting Metadata" on page 15-5. For more information on metadata snapshots, see Chapter 16, "Metadata Change Management".

- **Multiple Language Support base language compatibility:** The base language is the default language used in the repository and is set using the Repository Assistant during installation. This setting cannot be altered after installing the repository. Loading differing base language metadata objects into a repository results in error. For more information on setting the base language in a repository, see the Oracle Warehouse Builder Installation and Configuration Guide.

About the Metadata Import Utility

You can import metadata into a Warehouse Builder repository using one of the following:

- **Metadata Loader Command Line Utility:** You can use the command line utility to perform tasks additional tasks not available from the client interface. For example, you can override default values for configuration parameters for loading data. For instructions on importing from the command line, see "Using the Metadata Loader Command Line Utility" on page 15-20.
- **Warehouse Builder Client Interface:** For instructions on using the client interface, see "Importing Metadata" on page 15-11.

Validation Rules Governing Import

When you import a set of definitions from exported metadata, the import utility can update existing definitions in a Warehouse Builder project. However, certain metadata definitions require attention to ensure that they are updated. The following are examples of some of the errors you can see:

- **Mapping Definitions.** The Metadata Import Utility does not bind imported mapping operators to their physical objects. If a mapping definition appearing in the source MDL file replaces a mapping definition in the target repository, then the new repository mapping definition is unbound. You may need to reconcile new mapping operators with the physical objects they represent. The MDL generates a warning message to the log file stating that the mapping operators are not bound.
- **Foreign Key Definitions.** It is possible that a source MDL file can contain foreign key references to unique or primary keys that are not in the target repository. If the referenced unique or primary keys for any foreign key appearing in the MDL file does not exist in the target repository, the MDL will generate a warning message in the log file. This message will state that the repository does not contain a referenced key for the foreign key.

Importing Metadata using Warehouse Builder Client

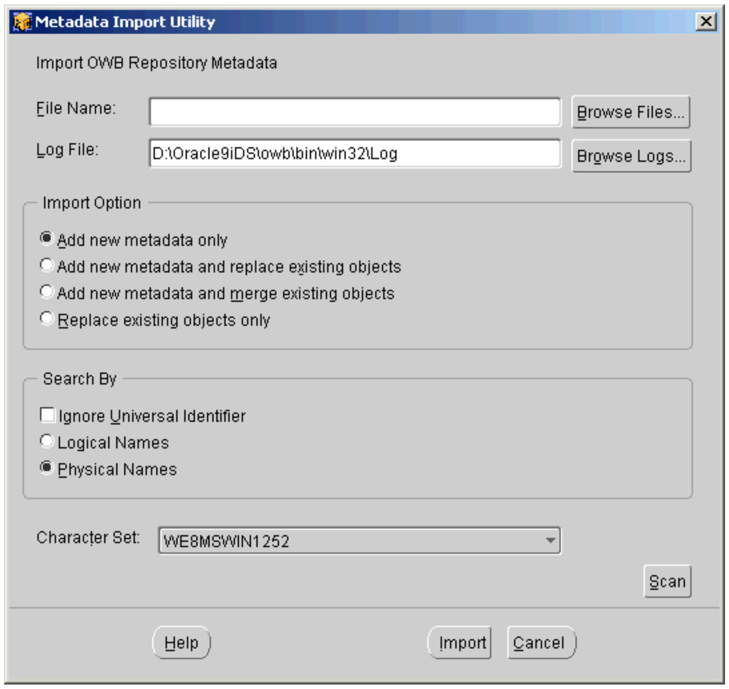
Use the metadata import utility to import objects from an MDL file into a Warehouse Builder repository.

To import objects from an export file using the Warehouse Builder client:

1. Select the project to which you will import metadata.
2. From the Warehouse Builder Console, select **Project** and select **MetaData Import.**

   Warehouse Builder displays the Metadata Utility Import dialog as shown in Figure 15–7.

Figure 15–7 Metadata Import Utility



Metadata Import Utility

Import OWB Repository Metadata

File Name: | Browse Files...

Log File: D:\Oracle9iDS\owb\bin\win32\Log | Browse Logs...

Import Option
- (●) Add new metadata only
- ( ) Add new metadata and replace existing objects
- ( ) Add new metadata and merge existing objects
- ( ) Replace existing objects only

Search By
- [ ] Ignore Universal Identifier
- ( ) Logical Names
- (●) Physical Names

Character Set: WE8MSWIN1252

3. Specify the names and locations for the import file and its log:

   File Name: Type the name of the MDL file or click **Browse** to find the MDL file you want to import.

   Log File: Warehouse Builder records information about the import in a log file. You can change this location by entering a new path and filename. Type the path and file name in the field or click **Browse** to locate a directory or filename.

4. Select an **Import Option.** For more information on import options, see "Import Modes" on page 15-15. You can select from the following import options:

   Add new metadata only: Adds new objects to a repository.

   Add new metadata and replace existing objects: Adds new objects to a repository and replaces existing objects.

   Add new metadata and merge existing objects: Adds new objects and merges columns into existing objects in your repository.

   Replace existing objects only: Replaces existing objects in your repository.

5. In **Match By,** specify the matching criteria the utility uses to compare the metadata in the import file against the metadata existing repository. For more information, see "Metadata Matching Criteria" on page 15-16.

   Ignore Universal Identifier: The import utility does not use Universal Identifiers to search for objects you are importing.

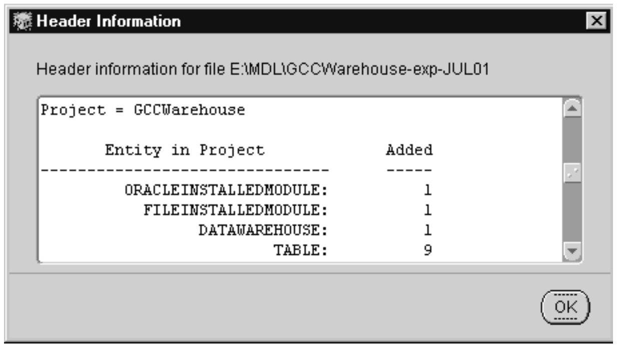
   Name: Searches your repository using the physical names of the objects you are importing to make sure the objects do not already exist.

   Character Set: Select the type of character set used to create the import file. The default character set is defined by the Warehouse Builder client machine. Use the drop-down list to change the output character set.

You can add new languages and character sets using the Repository Assistant. For more information, refer to the Oracle Warehouse Builder Installation and Configuration Guide.

6. Click **Scan** to display the exported metadata header information as shown in Figure 15–8. The Header Information dialog displays a summary of the total number of object types contained in the metadata file you selected.

Figure 15–8 Header Information



Header Information

Header information for file E:\MDL\GCCWarehouse-exp-JUL01

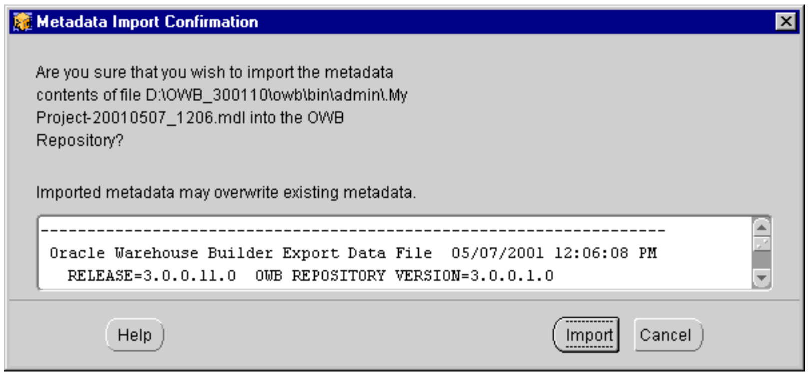
```
Project = GCCWarehouse

       Entity in Project            Added
-------------------------------     -----
         ORACLEINSTALLEDMODULE:         1
           FILEINSTALLEDMODULE:         1
                 DATAWAREHOUSE:         1
                         TABLE:         9
```

7. Click **Import.**

   If you have made any changes before starting the import, the Metadata Import Confirmation dialog displays. Click **Commit** to save any changes or **Rollback** to ignore changes and revert to the previously saved version.

   The Metadata Import Confirmation dialog displays as shown in Figure 15–9 if the exported metadata data information has not been reviewed.

Figure 15–9 Metadata Import Confirmation



Metadata Import Confirmation

Are you sure that you wish to import the metadata contents of file D:\OWB_300110\owb\bin\admin\My Project-20010507_1206.mdl into the OWB Repository?

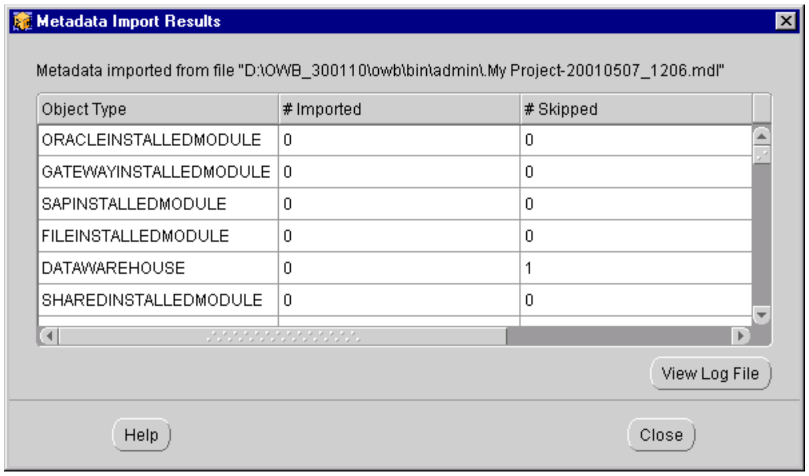
Imported metadata may overwrite existing metadata.

```
---------------------------------------------------------------------
 Oracle Warehouse Builder Export Data File  05/07/2001 12:06:08 PM
   RELEASE=3.0.0.11.0  OWB REPOSITORY VERSION=3.0.0.1.0
```

8. Click **Import** to continue.

   The Metadata Import Progress panel displays, as shown in Figure 15–10.

Figure 15–10 Metadata Import Results



Metadata imported from file "D:\OWB_300110\owb\bin\admin\My Project-20010507_1206.mdl"

Object Type	# Imported	# Skipped
ORACLEINSTALLEDMODULE	0	0
GATEWAYINSTALLEDMODULE	0	0
SAPINSTALLEDMODULE	0	0
FILEINSTALLEDMODULE	0	0
DATAWAREHOUSE	0	1
SHAREDINSTALLEDMODULE	0	0

This dialog displays the object types and the number of each type that were imported or skipped. For a detailed view of the import process, click **View Log File.**

Import Modes

The graphical user interface for the metadata import utility operates in one of the following modes:

- **Add new metadata only (Create Mode):** Adds new objects to a repository. If objects from the MDL file already exist in the repository, they remain unchanged.
- **Add new metadata and replace existing objects (Update Mode):** Adds new objects to a repository and overwrites existing objects with those in the MDL file.
- **Add new metadata and merge existing objects (Merge Mode):** Adds new objects and overwrites existing objects in your repository only if they differ with those in the MDL file.
- **Replace existing objects only (Replace Mode):** Replaces only existing objects in your repository. When importing metadata objects, the MDL will overwrite any existing metadata unless you use this mode.

When you import using the Update or the Replace modes, the import completely replaces the existing object's children so that the final object is exactly the same as the source object. Any existing children of a repository object that are not replaced or added are deleted. This occurs regardless of whether a child object occurs in a mapping or is a foreign, primary, or unique key column in a table or view.

For example, in the MDL export file, the CUST table contains three columns with the physical names: Last_Name, First_Name, and Middle_Init. In the repository, the same table already exists, and contains four columns with the physical names: Last_Name, First_Name, Status, and license_ID. During a replace operation, the columns Last_ Name and First_Name are replaced, column Middle_Init are added, and column Status and license_ID are deleted. The final result is that the CUST table in the Warehouse Builder repository contains the same metadata from the CUST table in the export file.

Tip: Using the replace mode can result in lost data constraints, metadata physical property settings, data loading properties, and mapping definitions. If you choose to use replace mode, ensure that you can restore your repository from backup to its state prior to importing in replace mode.

Metadata Matching Criteria

When you use the metadata import utility, it first searches the repository for metadata objects that exist in the repository and compares them to those in the file you are importing. How the comparison is made is determined by the loading mode and by the search method you choose. The following methods are available:

- **Physical Name:** Physical names are exported to the export file. The physical name determines whether an object needs to be created, replaced, or merged during an import operation. Use this method when object names in the target directory change, and you want to create new UOIDs for those objects.
- **Universal Object Identifier:** The metadata export utility assigns a unique system-generated identifier to each exported row object called the Universal Object Identifiers or UOIDs. The purpose of the UOID for a row object is to uniquely identify it in an object table. The MDL import utility uses these UOIDs to determine whether a row object needs to be created, replaced, or merged during an import operation. Use this method if you want to maintain UOIDs across different repositories even when object names in the target repository have changed.

By default, the import utility searches by UOIDs. However, the import utility ignores the UOIDs for mappings in the MDL file that already exist in the target repository.

Note: MDL imports that run in merge mode must use UOIDs for the search criteria in order to merge into existing mappings. Also, if the mapping in the MDL file does not have a Universal Identifier, the mapping cannot be merged into a mapping that matches by name. For more information, see "Import Modes" on page 15-15.

Each search method can be combined with an import mode in several different combinations. Each combination can offer different results in the import process. The mode that you select determines how the metadata import utility will search for metadata objects in the repository prior to importing.

For example, if the search is by the logical name of a repository object in the export file, the Metadata Import Utility searches the repository for the object's logical name. If an object with the corresponding logical name is not found, the resulting actions are based on the import mode you select.

Table 15–2 describes what happens in the available import modes for repository objects that do not match the MDL file names.

Table 15–2 Import Mode without Matching Names

Import Mode	Result
Create Mode	A new object is created.
Replace Mode	A warning message is written to the log file that the object cannot be found to replace and the object is skipped.

Table 15–2 (Cont.) Import Mode without Matching Names

Import Mode	Result
Update Mode	A new object is created.
Merge Mode	A new object is created.

Table 15–3 describes what happens in the available import modes for repository objects that match the MDL file names.

Table 15–3 Import Mode with Matching Names

Import Mode	Result
Create Mode	A message is written to the log file that the object already exists and the object is skipped.
Replace Mode	The object is replaced.
Update Mode	The object is replaced.
Merge Mode	The object is merged.

The MDL reads and processes the imported metadata and writes status and diagnostic information in the log file. When the import is complete, the Metadata Import Results dialog displays.

Restoring a Project

Restoring a project enables you to re-create metadata within a Warehouse Builder repository from an external location. Warehouse Builder provides a Restore Wizard to assist you in this process.

Note: The Archive and Restore utilities will be desupported in the next release of Oracle Warehouse Builder .

You must set up your Archive/Restore settings on the Preferences page before you can archive or restore your project. If you attempt to archive or restore without setting these preferences, you get an error.

Differences Between Restore and Import

Archive and Restore are different from Import and Export. Table 15–4 describes the differences between Restore and Import.

Table 15–4 Differences Between Restore and Import

Feature	Restore	Import
Character Set	UTF8	User Configured
Complete Project Replacement	Yes	Does not delete the project; may replace it, depending on MDL mode
Dump Format	MDL	MDL
UniversalID Preservation	Always	User Configured
Name Preservation	Always	User Configured
Log File Name	Generated	Generated and User configured

Table 15–4 (Cont.) Differences Between Restore and Import

Feature	Restore	Import
Mode	Replace	Create/Update/Replace/ Merge

Restoring a Project

Follow these instructions to restore a project.

To restore a project:

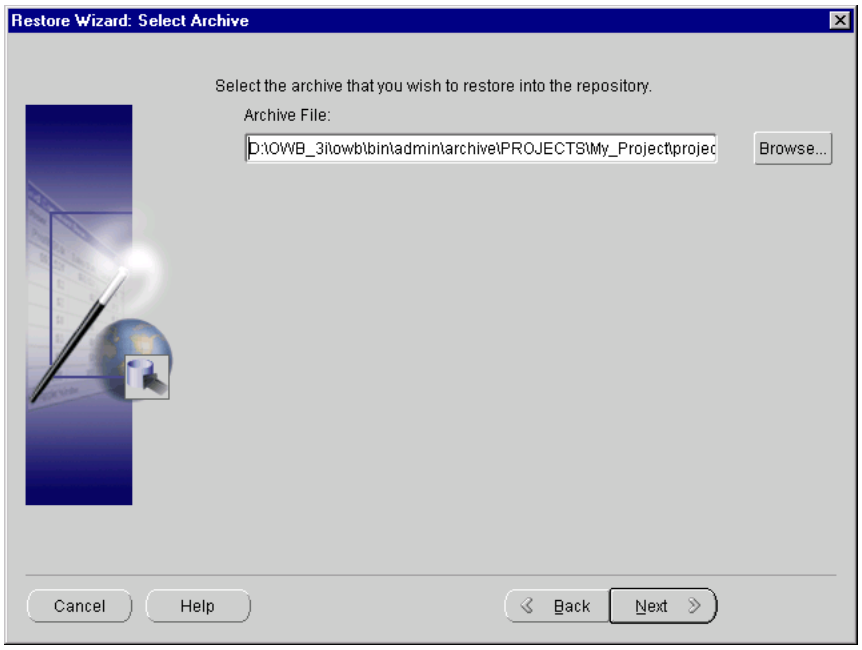
1. Select **Restore** from the Project menu.

   The Restore Wizard Welcome page displays.

2. Click **Next.**

   The Select Archive page displays, as shown in Figure 15–11. Browse to or type the Archive File you want to restore.

Figure 15–11 Select Archive Page



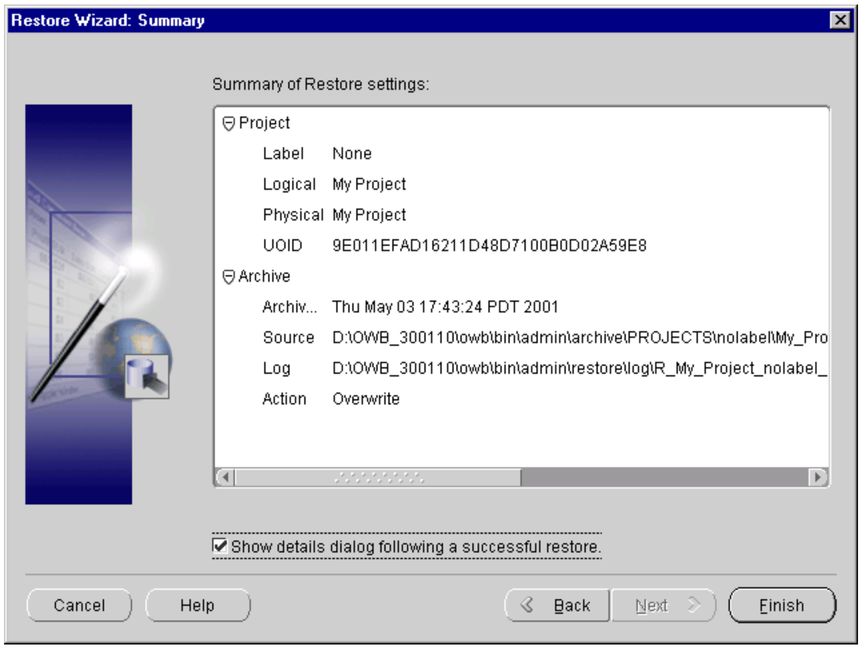
Select the archive that you wish to restore into the repository.

Archive File: D:\OWB_3i\owb\bin\admin\archive\PROJECTS\My_Project\projec

3. Click **Next.**

   The Summary page displays a summary of the restore settings prior to running the restore process, as shown in Figure 15–12. If you want to see the details of your restore after the restore process is complete, check the **Show details dialog following a successful restore** box.

Figure 15–12 Restore Wizard Summary Page



Summary of Restore settings:

```
Project
    Label     None
    Logical   My Project
    Physical  My Project
    UOID      9E011EFAD16211D48D7100B0D02A59E8
Archive
    Archiv... Thu May 03 17:43:24 PDT 2001
    Source    D:\OWB_300110\owb\bin\admin\archive\PROJECTS\nolabel\My_Pro
    Log       D:\OWB_300110\owb\bin\admin\restore\log\R_My_Project_nolabel_
    Action    Overwrite
```

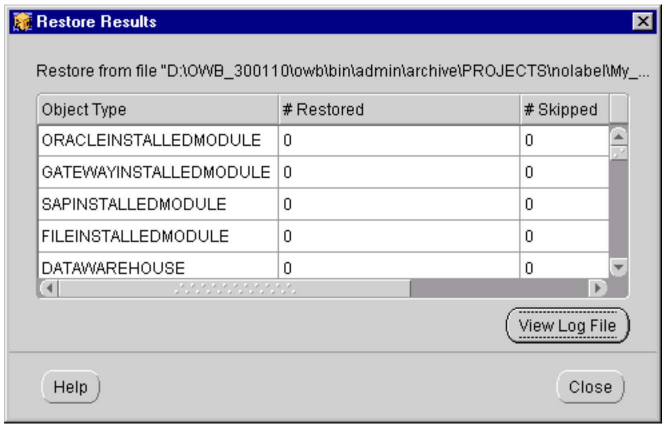
☑ Show details dialog following a successful restore.

4. Click **Finish.**

   This begins the restore process. A progress window appears. When the progress bar reaches 100%, the restore process is complete.

   If you checked the **Show details dialog following a successful restore** box, the Restore Results dialog displays, as shown in Figure 15–13. This dialog displays the name of each object type, how many of each were restored, and how many of each were skipped.

Figure 15–13 Restore Results

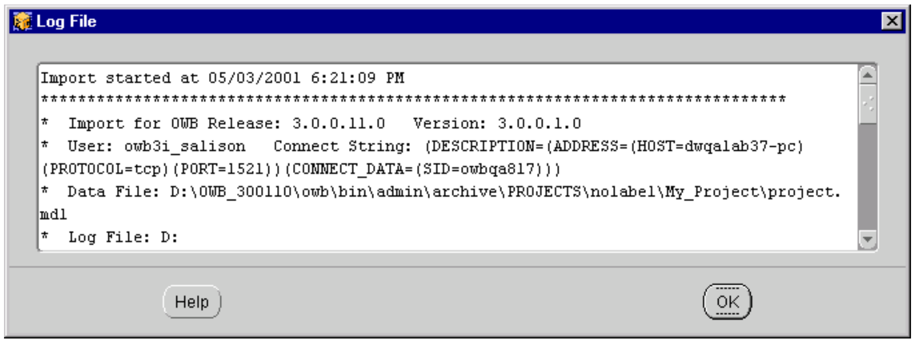


Restore from file "D:\OWB_300110\owb\bin\admin\archive\PROJECTS\nolabel\My_...

Object Type	# Restored	# Skipped
ORACLEINSTALLEDMODULE	0	0
GATEWAYINSTALLEDMODULE	0	0
SAPINSTALLEDMODULE	0	0
FILEINSTALLEDMODULE	0	0
DATAWAREHOUSE	0	0

For a more detailed look at the archive process, click **View Log File.** This displays the entire log file, as shown in Figure 15–14.

Figure 15–14 Restore Log File



```
Import started at 05/03/2001 6:21:09 PM
**********************************************************************************
*  Import for OWB Release: 3.0.0.11.0   Version: 3.0.0.1.0
*  User: owb3i_salison   Connect String: (DESCRIPTION=(ADDRESS=(HOST=dwqalab37-pc)
(PROTOCOL=tcp)(PORT=1521))(CONNECT_DATA=(SID=owbqa817)))
*  Data File: D:\OWB_300110\owb\bin\admin\archive\PROJECTS\nolabel\My_Project\project.
mdl
*  Log File: D:
```

Using the Metadata Loader Command Line Utility

You can operate the MDL from the command line instead of the user interface.

This section contains the following topics:

- Creating MDL Parameter Files at the Command Line on page 15-20
- Exporting Metadata Using the Command Line Utility on page 15-20
- Importing Metadata Using the Command Line Utility on page 15-23

For related information on using the Metadata Loader's command line utility for upgrade purposes, refer to the Oracle Warehouse Builder Installation and Configuration Guide.

Creating MDL Parameter Files at the Command Line

If you use the command-line interface, you can customize how to move your metadata at a more detailed level than you can when using the GUI. For example, when exporting metadata, the command line enables you the flexibility to disable the export of configuration values, vary the separator character within an export file, and maintain parameter files for selected export operations.

For importing metadata, the command line offers the flexibility of creating specific import actions for each object. These operations automate the consolidation or synchronization of metadata in multiple repositories that have a similar project structure.

The scripts for executing both the export and import utilities reside in the `$OWBHOME\owb\bin\win32` directory. Both the export and import utilities are driven by a set of parameters. You can specify the MDL parameters by:

- Providing MDL parameters as a response to command-line prompts.
- Creating an MDL parameters file.
- Providing MDL parameters as a response to command-line prompts, and create a MDL parameters file.

Exporting Metadata Using the Command Line Utility

By default, the MDL also exports values for configuration parameters for loading data, but you can override this setting at the command line. You can choose how you want to export the files within a project. For example, if you are exporting three source modules and two target modules, you can choose to export them separately or together.

To export a project at the command line:

1. Create the MDL parameters file.
2. Execute the Metadata Export Utility.

   The following command invokes the Metadata Export Utility and specifies the preceding parameters file:

   ```
   w:\owb\bin\win32>exp parfile=e:\MDL\EXP_Directives
   Processing ... Export successful.
   ```

   The objects are exported to the file and can be imported into a repository using the metadata import utility.

Keywords for the Export Utility

An MDL parameters file is a text file that contains a set of parameters for the export utility. The format for an export parameter is:

Keyword=Value

You can also form an export parameters by replacing the value with the wildcard character (*), which matches any string, or with a list of named objects:

Keyword=*

Keyword=(value-1, value-2, ..., -k)

For example, you can specify a set of tables to be exported as:

```
TABLES=(Customers, Products, Days)
```

Example 15–3 shows a typical parameters file for importing a module.

Example 15–3 Parameters File format

```
USERID=GCCWH/GCCWH@dwdoc11-pc:1521:ora816
PROJECT=GCCWarehouse
FILE=e:\MDL\GCCWarehouse-exp-JUL01
FIELDSEPARATOR=|
LOG=e:\MDL\GCCWarehouse-exp-JUL01-LOG
CONFIGPARAM=N
```

Table 15–5 summarizes the keywords used to form export parameters. You can use the comment indicator (#) to document the scripts. Put the indication in the first column of a record and follow it with text.

Table 15–5 Keywords for Export Utility Parameters

Utility Prompt	Keyword	Description
Username/password @host:port:sid	USERID	Username, password and connection as a string.
	USERNAME	The user name for accessing Warehouse Builder repository.
	PASSWORD	The password that corresponds to the USERNAME.
	HOST	Machine name for Warehouse Builder repository.
	PORT	Port for Warehouse Builder repository database listener.

Table 15–5 (Cont.) Keywords for Export Utility Parameters

Utility Prompt	Keyword	Description
	SID	SID for Warehouse Builder repository database.
Project Name	PROJECT	Project name. Wildcard format supported for Project, but if used, no other object type keywords can follow. In order to export shared transformations, use `PROJECT=Global Shared`.
Export File	FILE	File name for the exported data.
Field Separator	FIELDSEPARATOR	Field separators: |, ^ or ~.
Log File	LOG	File name for the status and statistics of the export.
Parameter File	PARFILE	Parameter file containing keywords.
	CONFIGPARAM	Export configuration values (Y/N). Default is Y.
	TRACE	Debug messages. Options: S - write messages to screen Y - write messages to a file B - write messages to screen and a file
	TRACEFILE	Trace file name.
	PHYSICALNAMES	Use physical names (Y/N) for lookup of objects to be exported. Default is N.
	CHARACTERSET	The character set to be used for the export data file.
	MODULES	If a wildcard or multi-value format is used for MODULE, no other object type keywords can follow. If a simple format is used, this keyword can appear multiple times, directly followed by keywords for any of its owned object types which can be selected using any format (simple, wildcard, multiple).
	TABLES	
	VIEWS	
	FILES	
	SEQUENCES	
	MATERIALIZED VIEWS	
	DIMENSIONS	
	FACTS	
	TRANSFORM CATEGORIES	For wildcard or multi-value format, no FUNCTIONS keyword can follow. If simple format then this keyword can appear multiple times, directly followed by a FUNCTIONS keyword, which can use any format (simple, wildcard, multiple).
	FUNCTIONS	
	MAPPINGS	
	COLLECTIONS	

Table 15–5 (Cont.) Keywords for Export Utility Parameters

Utility Prompt	Keyword	Description
	LOCATIONS	
	CONNECTORS	
	RUNTIMEREPOSITORYCONNECTIONS	
	STANDALONEFUNCTIONS	
	STANDALONEPROCEDURES	
	ADVANCEDQUEUES	
	EXTERNALTABLES	
	PROCESSES	
	SNAPSHOTS	
	QUERYOBJECTS	
	REPORTS	
	REPORTGROUPS	
	IOBUSINESSAREAS	
	HELP	Use HELP=Y for a complete list.
	#	Comment line used in a parameter file.

Importing Metadata Using the Command Line Utility

To import selected modules:

1. Create an MDL parameter file.
2. Execute the Metadata import utility.

   The following command invokes the Import Utility and specifies the preceding MDL parameter file:

   ```
   w:\owb\bin\win32>imp parfile=e:\MDL\IMP_Directives.txt
   Processing ...
   Import successful.
   ```

Keywords for the Import Utility

Like the MDL export, you can direct the MDL import to import objects from a file by answering prompts or by creating a file with a set of parameters. Example 15–4 shows a typical parameters file for importing a module.

Example 15–4 Parameters File format

```
USERID=GCCWH/GCCWH@dwdoc11-pc:1521:ora816
FILE=e:\MDL\gccstar-exp
LOG=e:\MDL\gccstar-imp-LOG
MODE=CREATE
CONFIGPARAM=N
```

Table 15–6 summaries the keywords used to form import parameters.

Table 15–6 Keywords for Import Utility Parameters

Utility Prompt	Keyword	Description
Username/passw @host:port:sid	USERID	Username, password and connection as a string.
	USERNAME	The user name for accessing Warehouse Builder repository.
	PASSWORD	The user password that matches USERNAME.
	HOST	Machine name for Warehouse Builder repository.
	PORT	Port for Warehouse Builder repository.
	SID	SID for Warehouse Builder repository.
Import File	FILE	File name for the data to be imported.
Import Mode	MODE	CREATE, REPLACE, UPDATE, or INCREMENTALUPDATE.
Log File	LOG	File name for the status and statistics of the export.
Parameter File	PARFILE	Parameter file containing keywords.
	CONFIGPARAM	Import configuration values (Y/N). Default is Y.
	TRACE	Debug messages. Options: S - write messages to screen Y - write messages to a file B - write messages to screen and a file
	TRACEFILE	Trace file name.
	PHYSICALNAMES	Use physical names (Y/N) to lookup objects to be imported. Default is Y.
	CHARACTERSET	The character set to use for the export data file.
	HELP	Use HELP=Y for a complete lis.t
	#	Comment line used in a parameter file.
	IGNOREUniversalID	Ignore (Y/N) the universal id as the search criteria. Default is N.
	PRESERVEDESCRIPTION	Preserve the description (Y/N) of already existing objects if the MDL data file does not have a description for the object. Default is N.
	SINGLEUSER	Request a single user lock (Y/N) for running the import. Default is N.

If a MODE parameter is not included, then the default is CREATE.

In addition to running an MDL parameter file from the import utility, you can also specify an action plan within the file that will allow you to specifically define what you want to do with each object in the imported file. First you need to specify if you want the object imported, skipped, or deleted. If you choose to import the object, you can set the import mode to CREATE, UPDATE, REPLACE, or INCREMENTAL UPDATE.

Example 15–5 shows an example of an MDL parameter file that contains an action plan.

Example 15–5 MDL Action Plan

```
USERID=user_sample/user_sample@test-pc:1521:ora8i
```

```
#
FILE=e:\test\data\sample_file.mdl
LOG=e:\test\log \imp_sample_file.log
#
MODE=ACTIONPLAN
PHYSICALNAMES=Y
IGNOREUOID=Y
#
# User-Specified Action Plan
#
ACTION=NONE
PROJECT=MY PROJECT
MODULES=(DATAWAREHOUSE)
#
ACTION=CREATE
TABLES=(TABLE_3)
FACTS=(FACT1, FACT2, FACT3)
SEQUENCES=(SEQ_A, SEQ_B, SEQ_C)
#
ACTION=REPLACE
TABLES=(TABLE_1, TABLE_2)
DIMENSIONS=(DIM1, DIM2, DIM3)
#
ACTION=DELETE
 TABLES=(TABLE_A, TABLE_B)
#
# Switching to a different module
ACTION=REPLACE
MODULES=(FLAT_FILE)
FILES=(FILE_1, FILE_2)
#
ACTION=CREATE
FILES=(FILE_3)
#
ACTION=DELETE
FILES=(FILE_X)
```

Splitter for Exporting and Importing Warehouse Builder Mappings

The Split utility provides a workaround for the memory limitations of the MDL import utility when you are importing a large number of mappings. This utility generates export and import scripts for migrating mappings in pieces as opposed to migrating them all at the same time. The generated scripts have matching MDL parameter files that utilize the `CREATE` mode. These files can be edited.

If the MDL import fails because of large data, the split utility can be used to re-export and import the mapping data in smaller pieces. All other object types must be exported and imported using the standard MDL utilities. Only mappings can be split into smaller pieces. To export all entities other than mappings, a parameter file containing the following can be used:

- `VIEWS=*`
- `TABLES=*`
- `SEQUENCES=*`
- `MATERIALIZEDVIEWS=*`
- `CUBES=*`

- `FILES=*`
- `DIMENSIONS=*`
- `VIRTUALTABLES=*`
- `TEMPORARYTABLES=*`
- `TRANSFORMCATEGORIES=*`

The split utility splits the mappings within a module in a Warehouse Builder project. The size of the pieces is determined by a parameter located in a file provided with this application.

The `expsplit` batch script accepts the following arguments:

- A parameter file, specified in the form `c:\temp\owb_apps.txt`. The parameter file has special keywords outlined in the following section that identify the number of mappings, data file names, and extensions.
- A parameter target file prefix, specified in the form `c:\temp\owb_apps` the piece number (numbered from 1). A `.txt` suffix is added to the generated parameter file, a `.bat` suffix is added for the export batch file, and an `_imp.bat` suffix is added to the import batch file.

The following is an example of how to start the split utility:

```
expsplit exampleparams.txt c:\temp\ora_apps
```

The following example uses a parameter file `exampleparams.txt`. This file contains the following parameters:

- `userid=apps/apps@130.35.12.73:1521:orcl0`
- `PHYSICALNAMES=Y`
- `LOG=c:\temp\owb_data_apps`
- `LOGEXT=log`
- `FILE=c:\temp\owb_data_apps`
- `FILEEXT=dat`
- `FIELDSEPARATOR=^`
- `PROJECT=EDWPRJ`
- `MODULES=EDW_COMMON_MODULE`
- `TYPE=MAPPINGS`
- `COUNT=70`

This file is similar to the export parameter file for Warehouse Builder Metadata Loader, with the changes listed in Table 15–7.

Table 15–7 Split Utility Export Parameter Keyword Descriptions

Keyword in Parameter File	Description
FILE	Prefix of data file, the chunk number, and file extension (FILEEXT) define the data file name where you exported the data.
FILEEXT	The data file extension.
PHYSICALNAMES	Used for name matching.

Table 15–7 (Cont.) Split Utility Export Parameter Keyword Descriptions

Keyword in Parameter File	Description
LOG	Prefix of the log file, the chunk number, and file extension (FILEEXT) define the log file name.
LOGEXT	The log file extension.
PROJECT	A single project name must be specified.
MODULES	A single module name must be specified.
TYPE	Must be MAPPINGS.
COUNT	The number of mappings to be written to each export chunk.

If the mappings for a Warehouse Builder project are split, the generated parameter files are named as follows:

```
owb_apps1.txt
owb_apps2.txt
```

A batch file is generated: `c:\temp\owb_apps.bat` (given the parameter target file prefix) to export the data from the repository. An import batch file is created to import, using create mode, into the same repository. These files can be edited if different target databases are required.

To migrate data using the split utility:

1. Using command line or Warehouse Builder, perform MDL export of all objects other then mappings.

   To export all objects other than mappings in a command line, use a parameter file with the following keywords:

   ```
   VIEWS=*

   TABLES=*

   SEQUENCES=*

   MATERIALIZEDVIEWS=*

   FACTS=*

   FILES=*

   DIMENSIONS=*

   VIRTUALTABLES=*

   TEMPORARYTABLES=*

   TRANSFORMCATEGORIES=*
   ```

   If you are using Warehouse Builder to perform export, use multi-select to select and export objects other than mappings.

2. Import the new export file into target repository.

3. Split the mappings and export them using split utility

   ```
   expsplit exampleparams.txt c:\temp\ora_apps
   ```

   The utility connects to the source repository, splits mappings, and creates multiple parameter files according to `exampleparams.txt`. These parameter files are used during the export. The utility also creates an export batch file and an import batch file.

Table 15–8 lists the files that are created.

Table 15–8 Files Created by the Split Utility

Description	File Name
Batch file to perform export	c:\temp\ora_apps.bat
Batch file to perform import	c:\temp\ora_apps_imp.bat
Multiple parameter files to be used by export and import batch files	c:\temp\ora_apps1.txt c:\temp\ora_apps2.txt c:\temp\ora_apps3.txt

4. Run the export batch file to export mappings into the location specified in the parameter file (variable FILE specified in step 3).
5. Modify generated parameter files `c:\temp\ora_apps1.txt`, `c:\temp\ora_apps2.txt`. Edit the connection information to point to the target repository.
6. Run import batch file `c:\temp\ora_apps_imp.bat` to complete the import.

16 Metadata Change Management

Oracle Warehouse Builder enables metadata change management and version management through snapshots. You can use metadata snapshots for backup, version management, compare, and restore purposes.

This section contains the following topics:

This section describes the metadata change management feature using the Warehouse Builder graphical user interface. For instructions on the commands and arguments to use when creating and managing snapshots using OMB Plus, refer to the Oracle Warehouse Builder Scripting Reference .

About Metadata Snapshots

Unlike an MDL export, which results in an .mdl file stored separately in your file system, a snapshot is stored as an object in the database. You can create snapshots for all first class objects (objects that you can access from the navigation tree). For a classification of Warehouse Builder objects, refer to "Object Ownership Tree" on page E-3.

A snapshot is related to the object it describes in the repository. It contains all the information about its objects and the relationships for the object. Because collections contain shortcuts to actual objects, a snapshot of a collection can be a snapshot of all the actual objects to which the shortcuts point.

If you take a snapshot of a parent object that includes child objects, and a child object changes, a snapshot comparison can show the parent object as having changed. For instance, if you create a Cascade snapshot of a project, all the modules in that project are its children. If any of the modules change, a comparison of the snapshot to the repository will show the project to have changed.

While an object can only have one current definition in the repository, it can have multiple snapshots that describe it at various points in time.

Note: When you take a snapshot of a collection, it is not a snapshot of the shortcuts in the collection but a snapshot of the actual objects. If you restore objects from this snapshot, you will restore the collection and the actual objects. Thus, when you restore a collection, you must first deselect the objects to be restored if you want to alter only the collection and not the actual objects.

Snapshots and Deleting Objects

When you delete an object from the Warehouse Builder navigation tree, a Delete Confirmation dialog displays. If you check the box indicating that you want to put this object in the recycle bin and click **OK**, Warehouse Builder displays a Snapshot Action dialog indicating that it is taking a snapshot of the object you are deleting.

Snapshot Types

Warehouse Builder enables you to take Full and Signature snapshots as well as Cascade and No Cascade snapshots.

Full and Signature Snapshots

This category determines the use of the snapshot:

- **Full snapshot:** Contains a complete set of metadata for the selected component at the time of capture. Full snapshots take longer to create and require more storage space than Signature snapshots, but provide complete metadata that you can use to restore your repository to previous history points. You can also export Full snapshots like any other repository object.
- **Signature snapshot:** Capture the object's signature. A signature contains enough information about the selected metadata component to detect changes when compared with another snapshot and compute the delta. Signature snapshots provide quick snapshot creation, but can only be used for comparison and information purposes. A comparison report of two snapshots identifies new, missing, and changed objects. Signature snapshots cannot be exported and imported.

If space becomes a concern, you can convert Full snapshots that you no longer need to keep for restore purposes to Signature snapshots, which consume less space. Conversion preserves the snapshot history and results in significant space savings in your repository. Currently, Warehouse Builder only enables you to perform conversions using scripting.

Note: Converting a Full snapshot to a Signature snapshot means that you can no longer use that snapshot for restore purposes.

Cascade and No Cascade Snapshots

Cascading determines the objects contained in the snapshot.

- **No Cascade snapshot:** Includes only the definition of the object itself, even if the object is a container object (such as a project or module) that contains child objects.

For instance, if you take a No Cascade snapshot of a Warehouse Module that contains tables, your snapshot will include the definition of Module A only.

- **Cascade snapshot:** Includes the expanded content of the object. If you take a Cascade snapshot of Warehouse Module A, which contains two tables, your snapshot will include the definition of Module A and the definition of both tables.

Snapshot Combinations

The two categories are *not* mutually exclusive. Table 16–1 describes the outcomes for a snapshot of a parent object taken at various combinations.

Table 16–1 Snapshot Type Combinations

Snapshot Type	Results of a Snapshot of Module A, which contains two tables: Table1 and Table2
Signature, Cascade Snapshot	This type of snapshot includes information about Module A, Table1, and Table2. This information cannot be used to restore those three objects; it can only be used for comparison purposes.
Full, Cascade Snapshot	This type of snapshot includes the definition of Module A, Table1, and Table2. This snapshot can be used to restore any of these three objects, or it can be used for comparison purposes only.
Signature, No Cascade Snapshot	This type of snapshot includes information about Module A only. This information cannot be used to restore Module A; it can only be used for comparison purposes. If Table1 or Table2 has changed, the comparison does not show that Module A has changed.
Full, No Cascade Snapshot	This type of snapshot includes the definition of Module A only. This snapshot can be used to restore or compare Module A, but not its children objects. If Table1 or Table2 has changed but if Module A has otherwise remained the same, the comparison does not show that Module A has changed.

Creating Snapshots

Metadata change management enables you to take snapshots that capture the content of your metadata repository, or specific objects in your repository, at a given point in time. You can use a snapshot to detect and report changes in your metadata.

Note: When you take a snapshot of a collection, it is not a snapshot of the shortcuts in the collection but a snapshot of the actual objects. If you restore objects from this snapshot, you will restore the collection and the actual objects. Thus, when you restore a collection, you must first deselect the objects to be restored if you want to alter only the collection and not the actual objects.

To create a snapshot in Warehouse Builder:

1. From the Warehouse Builder navigation tree, multi-select the components you want to include in the snapshot. For example, you can select tables, mappings, dimensions from an Oracle warehouse module.

2. Right-click the multi-selected components and select **Create Snapshot** from the pop-up menu.

   The Create Snapshot Welcome page displays.

3. Click **Next.**

   The Create Snapshot Name page displays.

4. Provide the following information:

   Name: Type a name up to 30 characters for the new snapshot.

   Type: Indicate the type for the snapshot, Full or Signature. For more information on snapshot types, see "Snapshot Types" on page 16-2.

   Description: Type a description up to 2000 characters for the new snapshot.

5. Click **Next.**

   The Create Snapshot Components page displays.

   This page contains three columns:

   Object: Lists the Warehouse Builder components that you want to include in your snapshot.

   Type: Indicates the type of the Warehouse Builder component. For example, warehouse module, table, dimension, mapping, or view.

   Cascade: This option enables you to select how an object is included in the snapshot. When you choose to cascade an object, you choose to include all child components of that object in the snapshot. For example, when you cascade an Oracle target module, you choose to include all the tables, dimensions, cubes, and other objects contained in that module. This option only applies to folder-level objects, such as modules, projects, or collections.

6. Select the Cascade options for each component in the snapshot and click **Next.**

   The Create Snapshot Dependency page displays.

   This page enables you to select dependees for the objects in your snapshot. For example, a table with a foreign key is dependent on the table containing the referenced key. You can select 1 as the depth of dependency if you want to include the table containing the unique key in the snapshot. The maximum depth you can indicate is 100.

7. Optionally, click **Preview** to view the list of dependees for the depth you indicate.

   The Dependency Preview dialog enables you to preview the dependees that would be included in your snapshot. The list of dependees is based on the depth level you indicate on the Dependency page. For example, if you choose to include a mapping with a table and indicate a depth of 1, this dialog shows you the table contained in that mapping. If this table contains a foreign key and you indicate a depth of 2, this dialog lists the associated table containing the unique key.

8. Click **Next.**

   The Create Snapshot Finish page displays.

9. Review the choices you made in this wizard to create a new snapshot. Click **Back** to make changes. Or click **Finish** to create the snapshot.

   The snapshots you create are displayed in the Metadata Change Management window.

Updating Snapshots

After you create a snapshot, you can update it by adding more components to it. This section shows you how to update an existing snapshot.

To update an existing snapshot:

1. From the Warehouse Builder navigation tree, select the components you want to add to the snapshot. For example, you can select tables, mappings, dimensions from an Oracle warehouse module.
2. Right-click the selected components and select **Add to Snapshot** from the pop-up menu.

   The Add to Snapshot Welcome page displays.
3. Click **Next.**

   The Add to Snapshot page displays.
4. From the drop-down field, select an existing snapshot that you want to update and click **Next.**

   The Add to Snapshot Components page displays. This page contains three columns:

   Object: Lists the Warehouse Builder components that you want to add to your snapshot.

   Type: Indicates the type of the Warehouse Builder object. For example, warehouse module, table, dimension, mapping, or view.

   Cascade: This option enables you to select how an object is included in the snapshot. When you choose to cascade an object, you choose to include all child components of that object in the snapshot. For example, when you cascade a module, you choose to include all the tables, dimensions, cubes, and other objects contained in that module. This option only applies to folder-level objects, such as modules or projects.
5. Select the cascade options for the components you want to add and click **Next.**

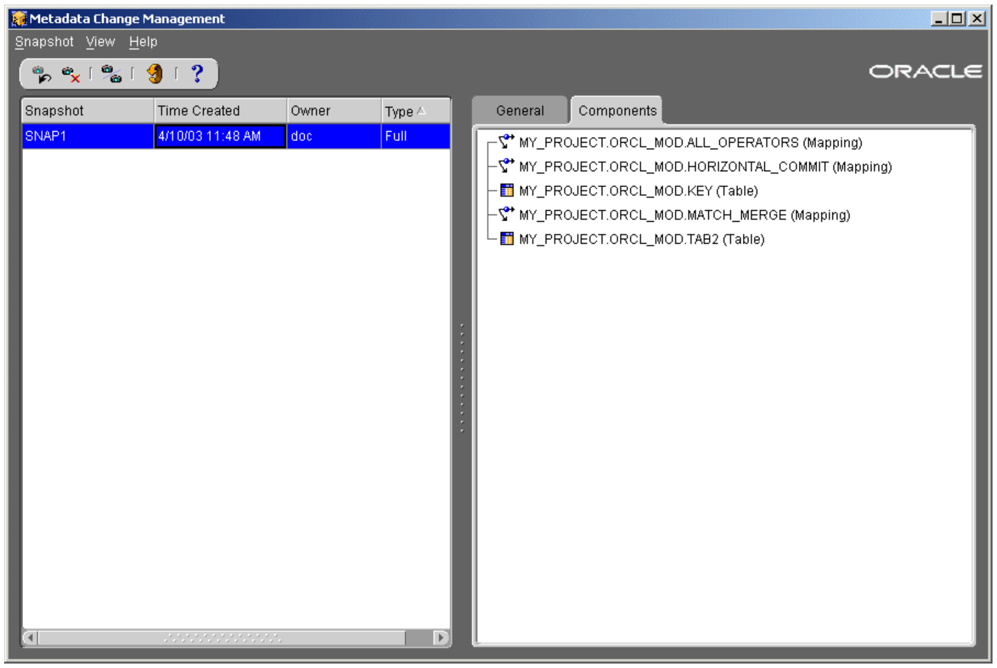
   The Add to Snapshot Finish page displays.
6. Review the choices you specified for the snapshot. To make changes, click **Back.** Or click **Finish** to update the snapshot.

   You can view the updated snapshot in the Metadata Change Management window.

Metadata Change Management Window

Warehouse Builder enables you to manage your snapshots from the Metadata Change Management Window. To open this window, select **Project** then **Change Manager** from the Warehouse Builder console. Figure 16–1 shows the Metadata Change Management window.

Figure 16–1 Metadata Change Management Window



Snapshot	Time Created	Owner	Type
SNAP1	4/10/03 11:48 AM	doc	Full

- MY_PROJECT.ORCL_MOD.ALL_OPERATORS (Mapping)
- MY_PROJECT.ORCL_MOD.HORIZONTAL_COMMIT (Mapping)
- MY_PROJECT.ORCL_MOD.KEY (Table)
- MY_PROJECT.ORCL_MOD.MATCH_MERGE (Mapping)
- MY_PROJECT.ORCL_MOD.TAB2 (Table)

This window contains two columns. The column on the left side displays the snapshots available in your repository, the date and time they were created, their owners, and their type. You can sort the list of snapshots by any of these columns by clicking on the column header.

The right side column contains two tabs: General and Components. The General tab displays name, type, and description of the selected snapshot. The Components tab displays all the objects contained in the selected snapshot in a navigation tree format. The root of this tree depends on the components contained in that snapshot. For example, if the highest level object in the snapshot is a warehouse module, then that is the root. If the snapshot contains an entire project, then the project is the root. If a snapshot only contains three tables, then those tables become the roots of the tree.

You can perform the following activities from the Metadata Change Management window:

- "Compare Snapshots" on page 16-6
- "Restore Snapshots" on page 16-8
- "Restore Snapshots" on page 16-8
- "Delete Snapshots" on page 16-10

Compare Snapshots

Warehouse Builder enables you to compare two snapshots or a snapshot with a current repository object by using Universal Object Identifiers (UOIDs) as the basis for all comparisons.

To compare two snapshots:

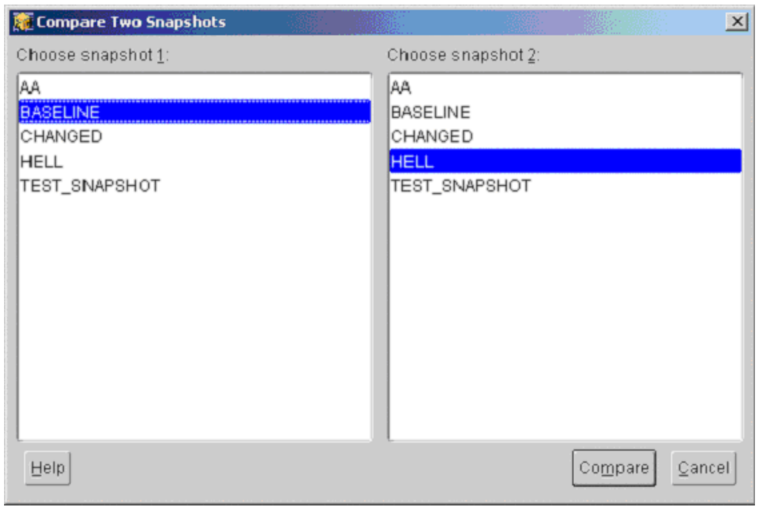
1. Use one of following methods:

   From the Metadata Change Management window, select **Snapshot** then **Compare.**

Or from the list of snapshots displayed in the Metadata Change Management window, multi-select the two snapshots you want to compare, right-click the snapshots, and select **Compare.**

The Compare Two Snapshot dialog displays as shown in Figure 16–2. If you already selected the two snapshots you want to compare, they will be pre-selected in the columns of this dialog.

Figure 16–2 Compare Two Snapshots Dialog



2. Select the snapshots you want to compare from the Choose Snapshot 1 and Choose Snapshot 2 columns and then click **Compare.**

   Warehouse Builder displays the comparison results in the Snapshot Comparison dialog. For more information, see "Snapshot Comparison Dialog" on page 16-7.

To compare a current repository object with a snapshot component:

1. Select the object you want to compare from the Warehouse Builder navigation tree.
2. Right-click the object and select **Compare With Snapshot.**

   The Choose Snapshot dialog displays. This dialog lists all the snapshots containing versions of that object.

3. Select the snapshot to which you want to compare the current repository object and click **OK.**

   The Snapshot Comparison dialog displays the comparison results. For more information, see "Snapshot Comparison Dialog" on page 16-7. If the repository object has not changed, then Warehouse Builder displays a message stating that there is no change.

Snapshot Comparison Dialog

When you compare two snapshots or when you compare a snapshot component with a repository object, the results display in the Snapshot Comparison dialog.

Note: When two snapshots are identical, Warehouse Builder displays a message that the two snapshots are identical. In this case, you will not see the Snapshot Comparison dialog.

The View filter option at the top enables you to view only the changed objects. You can also select All Objects to display all objects in the snapshot, whether they are changed or not.

The Snapshot Comparison dialog contains two columns. The left side column displays the comparison results in a navigation tree format. The tree icons indicate the status of the snapshot components as shown in Table 16–2.

Table 16–2 Snapshot Comparison Icons

Icon	Status
✓✓	Both Snapshot 1 and Snapshot 2 contain the component. There is no change.
✓▢	The component only exists in Snapshot 1.
▢✓	The component only exists in Snapshot 2.
✓✗	The object exists in both snapshots but it has changed.

The right side column contains three tabs: General, Properties, and Links. These tabs display the following information:

General: Provides an overview of the comparison results. Select an object name from the navigation tree in the left column to view a summary of its physical name, business name, and type. This column also indicates whether there are any changes in the object properties, associated child components, or links.

Properties: Shows whether the properties of an object have changed or whether the object exists in both snapshots or only one of them.

Links: Shows whether there are changes to any links or associations for the object. The icons under the Difference field indicate the status of the object as shown in Table 16–2.

To save the comparison results as an XML file, click **Save As.** Or click **Close** to close the dialog.

Considerations

When you compare snapshots, Warehouse Builder uses the Unique Object Identifier (UOID) for each object to perform the match. If you delete an object and re-create it with same metadata, the object now has a different UOID, although it looks similar to the deleted object. When you compare this object to its previous version, the results will show all the metadata as changed because you re-created the object again. Not only the changed metadata but the unchanged metadata also displays in the comparison results.

Because Warehouse Builder uses UOID to match nodes, if objects are deleted and re-created during MDL import or Intelligence Object derivation, the results will be different. This is true even if you trigger IGNORE UOID = TRUE during MDL Import.

Restore Snapshots

When you restore a snapshot, you replace the current definition of the object in the repository with the snapshot image of that object. When restoring from full, cascade snapshots of projects, collections, or folders, you can select which child objects you want to restore. You can restore the complete snapshot, or only a part of it by selecting specific child objects.

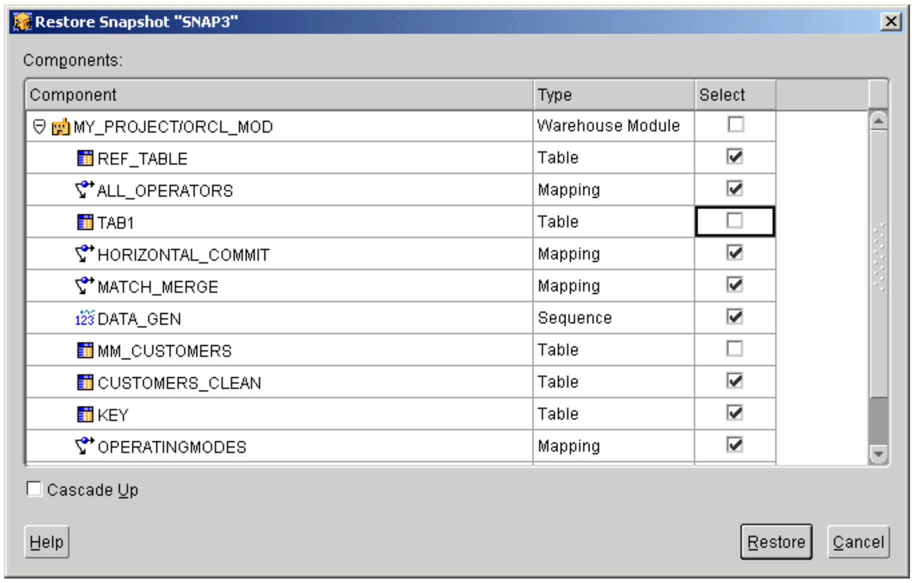
To restore a snapshot:

1. Open the Metadata Change Management window and select the snapshot you want to restore.
2. Right-click the snapshot and select **Restore** from the pop-up menu. Or from the Snapshot menu, select **Restore.**

   If you have uncommitted work, you will be asked to commit first before restoring a snapshot.

   Then the Restore Snapshot dialog displays as shown in Figure 16–3.

Figure 16–3 Restore Snapshot Dialog



Component	Type	Select
MY_PROJECT/ORCL_MOD	Warehouse Module	☐
REF_TABLE	Table	✓
ALL_OPERATORS	Mapping	✓
TAB1	Table	☐
HORIZONTAL_COMMIT	Mapping	✓
MATCH_MERGE	Mapping	✓
DATA_GEN	Sequence	✓
MM_CUSTOMERS	Table	☐
CUSTOMERS_CLEAN	Table	✓
KEY	Table	✓
OPERATINGMODES	Mapping	✓

☐ Cascade Up

   This dialog enables you to select the components to restore from a snapshot. By default, Warehouse Builder restores all components if you click **Restore.** You can also restore only a part of the snapshot by selecting specific child objects.

When you restore folders, all its child objects are also restored. For example, if you choose to restore a warehouse module, all the objects contained in that module are also automatically restored. To restore only selected objects within this warehouse module, you must first uncheck the warehouse module or any other folder and then selectively restore the objects within that folder.

The Restore dialog contains the following columns and options:

Component: Lists the Warehouse Builder components included in the snapshot.

Type: Indicates the type of the Warehouse Builder object. For example, warehouse module, table, dimension, mapping, or view.

Select: This is the column you check to indicate whether you want to restore the component.

Cascade Up: This option controls the restore when the parent of a component no longer exists in the repository. For example, you are restoring a table T1 originally located under MY_WH module under project MY_PROJECT. By checking Cascade Up, the system creates a dummy MY_PROJECT and MY_WH containing T1. If you do not check Cascade Up, the system will not be able to locate the parent folders for this table and the restore will fail.

Click **Restore** to restore the selected snapshot objects to the repository. Warehouse Builder warns you that all currently open editors will be closed after a successful restore. Click **Yes** to continue and complete the restore.

Delete Snapshots

You can delete snapshots or components within a snapshot from the Metadata Change Management window.

To delete a snapshot:

1. Open the Metadata Change Management window and select the snapshot you want to delete.
2. Right-click the snapshot and select **Delete** from the pop-up menu. Or from the Snapshot menu, select **Delete.**

   Warehouse Builder displays a delete confirmation dialog.
3. Click **Yes** to delete the selected snapshot from the repository.

To delete components in a snapshot:

1. Open the Metadata Change Management window and select the snapshot from which you want to delete components.
2. From the right-side column of the Metadata Change Management window, select the Components tab.

   Warehouse Builder displays the components of the snapshot on the right-side column.
3. Right-click the component you want to delete and select **Delete** from the pop-up menu. Or from the Snapshot menu, select **Delete.**

   Warehouse Builder displays a delete confirmation dialog.
4. Click **Yes** to delete the selected component from the snapshot.

Using Metadata Snapshots

This section explains snapshot usage concepts using OMB Plus. For instructions on the commands and arguments for creating and managing snapshots using OMB Plus, refer to the Oracle Warehouse Builder Scripting Reference .

History Management

History management includes creating, deleting, restoring, and altering snapshots.

Creating Snapshots

Use the following syntax to create a snapshot:

```
OMBCREATE SNAPSHOT 'SNAPSHOTNAME' SET PROPERTIES
(DESCRIPTION,SNAPSHOTTYPE) VALUES (Your description value,
'FULL')
```

```
ADD FIRST_CLASS_OBJECT_TYPE FULLY_QUALIFIED_FIRST_CLASS_OBJECT_
NAME …
```

Example:

```
OMBCREATE SNAPSHOT S1 SET PROPERTIES (DESCRIPTION, SNAPSHOTTYPE)
VALUES ('Sample Snapshot', Y ,Y)

ADD TABLE 'MY_PROJECT/WH/EMP'

ADD TABLE 'MY_PROJECT/WH/DEPT'

ADD DIMENSION 'MY_PROJECT/WH/DIM1'

ADD MAPPING 'AnotherProject/WH1/MAP1'
```

Deleting Snapshots

Use the following syntax to delete a snapshot:

```
OMBDROP SNAPSHOT 'SNAPSHOTNAME'
```

Example:

```
OMBDROP SNAPSHOT 'SS1'
```

Altering Snapshots

You can alter an existing snapshot directly, thus "altering history" as you need. You can also use the ALTER command to delete a non-root snapshot (for more information about root snapshots, see "Cascade and No Cascade Snapshots" on page 16-2).

Use the following syntax to alter a snapshot:

```
OMBALTER SNAPSHOT 'SNAPSHOTNAME'

( SET PROPERTIES (SNAPSHOTTYPE) VALUES ('FULL'/'SIGNATURE')

)?

( ADD FIRST_CLASS_OBJECT_TYPE RELATIVE_FIRST_CLASS_OBJECT_NAME

 | DELETE FIRST_CLASS_OBJECT_TYPE RELATIVE_FIRST_CLASS_OBJECT_
NAME

 | MODIFY FIRST_CLASS_OBJECT_TYPE RELATIVE_FIRST_CLASS_OBJECT_
NAME

)+
```

Example:

```
OMBALTER SNAPSHOT S1

ADD TABLE 'MY_PROJECT/MOD1/T1'

ADD TABLE 'MY_PROJECT/MOD1/T2'

DELETE DIMENSION 'MY_PROJECT/WH/DIM1'
```

This command adds two tables and removes the existing dimension from the snapshot definition.

Restoring Snapshots

Restoring a snapshot means replacing the current definition of the object in the repository with the snapshot image of that object. When restoring from full, cascade snapshots of projects, collections, or folders, you can select which child objects you want to restore. You can restore the complete snapshot, or only a part of it by selecting specific child objects.

Restoring an Entire Snapshot Use the following syntax to restore an entire snapshot:

```
OMBRESTORE SNAPSHOT 'SNAPSHOTNAME'
```

Example:

```
OMBRESTORE SNAPSHOT 'SS1'
```

This command restores all the objects in the snapshot. If any instances fail to restore, the whole operation fails with an error message. Failure to restore means that the aggregate parent of all the objects within the snapshot does not exist in the current repository.

Restoring Selected Objects from a Snapshot Use the following syntax to restore select objects from a snapshot:

```
OMBRESTORE SNAPSHOT 'SNAPSHOTNAME' FOR FIRST_CLASS_OBJECT_TYPE
'RELATIVE_FIRST_CLASS_OBJECT_NAME'
```

Example:

```
OMBRESTORE SNAPSHOT 'SS1' FOR TABLE 'MY_PROJECT/WH/EMP' MAPPING
'AnotherProject/WH1/MAP1'
```

This command replaces the current repository metadata definition of table `EMP` and mapping `MAP1` with the table and mapping defined in snapshot `SS1`.

Orphaned Snapshots Deleting an object that has snapshots does not delete the snapshots of that object. Deleting an object can therefore result in a snapshot being orphaned, or remaining in the repository without an object to which it belongs. If you try to restore an orphaned snapshot, Warehouse Builder produces an error message.

Use the following syntax to restore the missing parents of an orphaned snapshot:

```
OMBRESTORE SNAPSHOT 'SNAPSHOT_NAME' CASCADE UP
```

Snapshots follow Warehouse Builder's transaction mechanism, so you will not have orphaned snapshots resulting from a rollback. If you create a new object and a snapshot of it without committing, and then rollback your changes, the object and snapshot are both deleted.

Exporting and Importing Snapshots

You can export and import full snapshots only, using the OMB Plus scripting interface and the Warehouse Builder user interface. Export and import full snapshots to move them between repositories or to use them for future upgrades. Exporting a full snapshot converts it into a Metadata Loader `.mdl` file. Importing a full snapshot converts an `.mdl` file into a snapshot and stores the snapshot in the tablespace allotted for snapshot storage in your repository.

Change Management

Change management includes listing, retrieving information about, and comparing snapshots.

Listing Snapshots

You can list all available snapshots for a given repository, or you can list any snapshots associated with a specific object in the repository.

Listing All Snapshots for a Repository Use the following syntax to list all snapshots for a repository:

```
OMBLIST SNAPSHOTS regexp
```

Example:

```
OMBLIST SNAPSHOTS
```

This command returns a list of all snapshots for the repository in question.

Listing Snapshots Associated With an Object Use the following syntax to list existing snapshots for a specific object:

```
OMBLIST SNAPSHOTS FOR FIRST_CLASS_OBJECT_TYPE 'RELATIVE_FIRST_
CLASS_OBJECT_NAME'
```

Example:

```
OMBLIST SNAPSHOTS FOR TABLE 'EMP'
```

This command lists the names of all snapshots that are associated with the current table `EMP`.

Retrieving Information About Snapshots

Use the following syntax to retrieve information about a snapshot:

```
OMBRETRIEVE SNAPSHOT 'SNAPSHOTNAME' GET

( PROPERTIES (TIMESTAMP|DESCRIPTION|SNAPSHOTTYPE)

| CONTENTS (ALL | MAPPINGS | TABLES |…)

 ( START WITH 'FIRST_CLASS_OBJECT_NAME')? (CASCADE | NO
CASCADE)? ) )
```

Examples:

- `OMBRETRIEVE SNAPSHOT S1 GET PROPERTIES (DESCRIPTION)`

  This command returns the description for snapshot S1:

```
Sample Snapshot
```

- `OMBRETRIEVE SNAPSHOT S1 GET CONTENTS ALL`

  This command returns:

```
TABLE 'MY_PROJECT/WH/EMP'

TABLE 'MY_PROJECT/WH/DEPT'

DIMENSION 'MY_PROJECT/WH/DIM1'

MAPPING 'AnotherProject/WH1/MAP1'
```

- `OMBRETRIEVE SNAPSHOT S1 GET CONTENTS TABLES`

  This command returns:

```
TABLE 'MY_PROJECT/WH/EMP'

TABLE 'MY_PROJECT/WH/DEPT'
```

Comparing Snapshots

Using the OMB Plus utility, you can compare two snapshots to each other, or a snapshot to a current repository object. Warehouse Builder uses Universal Object Identifiers (UOIDs) as the basis of all comparisons.

When comparing two snapshots or a snapshot with an object, identify one as the source and one as the target. For instance, if you are comparing a snapshot to a current module object with the intention of restoring certain properties to the object, the snapshot is your source and the object is your target.

You can select from the following comparison report filters, to report on:

- **Objects that are found in the source only:** This method only identifies objects that are found in the source but not in the target.
- **Objects that are found in the target only:** This method only identifies objects that are found in the target object but not in the source object.
- **Changed objects found in both source and target:** This method identifies objects that are found in both the source and target, but whose definitions differ in the two.
- **Unchanged objects found in both source and target:** This method identifies objects that are identical in both source and target.
- **All objects:** This method identifies all the objects in the target, plus objects found in the source only.

You can generate a comparison report when you compare two snapshots to each other, or a snapshot to a current repository object. A comparison report is generated in XML format, and identifies new, missing, and changed objects. You can generate snapshot comparison reports from the `mcmview.bat` command line utility, or from OMB Plus.

Comparing Snapshots in OMB Plus To generate snapshot comparison reports from OMB Plus, use the prepackaged XML schema defined for the generated XML. For more information, consult the Oracle Warehouse Builder Scripting Reference .

Use the following syntax to compare two snapshots in OMB Plus:

```
OMBCOMPARE SNAPSHOT 'SRC_SNAPSHOTNAME' WITH

( SNAPSHOT 'TGT_SNAPSHOTNAME' | CURRENT )

FOR FIRST_CLASS_OBJECT_TYPE 'RELATIVE_FIRST_CLASS_OBJECT_NAME'

SHOW (ALL | CREATED | DELETED | MODIFIED | UNMODIFIED) OUTPUT TO
'XMLFILENAME'
```

Examples:

- `OMBCOMPARE SNAPSHOT 'SS1' WITH SNAPSHOT 'SS2' FOR TABLE 'EMP' SHOW ONLY DELETED OUTPUT TO 'c:\diffresults\emp_diff.xml'`
- `OMBCOMPARE SNAPSHOT 'SS1' WITH CURRENT FOR TABLE 'MY_PROJECT/WH/EMP' SHOW ONLY DELETED OUTPUT TO 'c:\diffresults\emp_diff.xml'`

Comparing Snapshots in the Command Line Utility MCMVIEW.BAT A command line utility, `mcmview.bat`, is packaged for generating snapshot comparison reports as well. The command line utility prepares the XML for reading and printing the report in HTML format. The utility is located in the `owb/bin/win32` directory and should be executed passing the OWB compare XML filename as a parameter.

For example, the command `mcmview c:\temp\tables_021111.xml` results in a file named `tables_021111.xml.html` in the `win32` directory, and the suffix `.html` is added to the XML file name.

Snapshot Usage Suggestions

Used in conjunction with other Warehouse Builder features, such as the MDL Metadata Import utility and impact analysis, snapshots facilitate metadata management. Snapshots are also designed to meet the unique needs of different Warehouse Builder users.

Using Snapshots in Conjunction with Other Warehouse Builder Functionality

Use snapshots in conjunction with MDL metadata import to determine the impact an import will have on the current metadata repository. Based on a repository snapshot comparison, select actions to perform metadata import at a finer grain of detail. You can use impact analysis prior to metadata import to determine the dependent objects that will be impacted by the changes you plan to accept during import. You can also use snapshots during deployment to identify the objects that need to be redeployed.

Snapshot Views for Data Warehouse Designers and Managers

Metadata snapshots vary in granularity to account for the unique needs of different Warehouse Builder users. Warehouse designers can use snapshots to capture their individual components, such as their mappings, entities, and process flows, at various points in time, whereas data warehouse managers can use snapshots to detect or preserve the history of an entire warehouse module or project. Snapshot views are accessible through OMB Plus. For more information, see the Oracle Warehouse Builder Scripting Reference .

The Object to Snapshot view for designers enables data warehouse designers to take snapshots of their work.

- For instance, when working on a specific mapping or entity, a developer can take a full snapshot of that entity. Later, that developer can restore the mapping to its previous state using the full snapshot.
- Or, a developer can take a signature of a specific component and later use that snapshot to identify the changes that have occurred for that component.

The Snapshot to Object view for managers enables data warehouse managers to survey metadata history.

- Managers can use this view to oversee, delete, and compare the snapshots created in the repository.
- This view enables the administrator to perform large-scale actions, such as deploying a warehouse or restoring a warehouse to a previous point in history.
- The data warehouse manager can also delete old, unnecessary snapshots in bulk using this view.
- Managers can view all the snapshots associated with a specific object in one window and can manage them through one consolidated view in the repository.
- Finally, this view also enables the data warehouse manager to upgrade snapshot storage to match a new version of Warehouse Builder software.

17 Metadata Browsing and Reporting

Use Warehouse Builder Design Browser to browse metadata stored in your repository and view reports on that metadata. You can access metadata reports from the Warehouse Builder client, which opens through a browser created and maintained by your Internet Application Server Administrator.

This chapter contains the following information:

- About Warehouse Builder Design Browser on page 17-1
- Using Warehouse Builder Design Browser on page 17-3
- Viewing Warehouse Builder Reports on page 17-16

About Warehouse Builder Design Browser

You can use Warehouse Builder Design Browser to view and generate reports on all repository metadata objects and the relationships between those objects.

You have two options with the Warehouse Builder Design Browser. You can use the client version, which is installed together with the Warehouse Builder Client, or you can use the Portal version—the version of Warehouse Builder Design Browser that is integrated with Oracle Database Application Server. The version available to you depends on what steps your Warehouse Builder Administrator performed during installation. For more information on the two versions of the Warehouse Builder Design Browser, refer to the Oracle Warehouse Builder Installation and Configuration Guide.

The Warehouse Builder Design Browser consists of portlets that you can add to customize your portal page. You can add more than one of each type of portlet. For example, you can add two Reports portlets with each running a report on a different repository.

The portlets used for viewing and generating reports on metadata include the following:

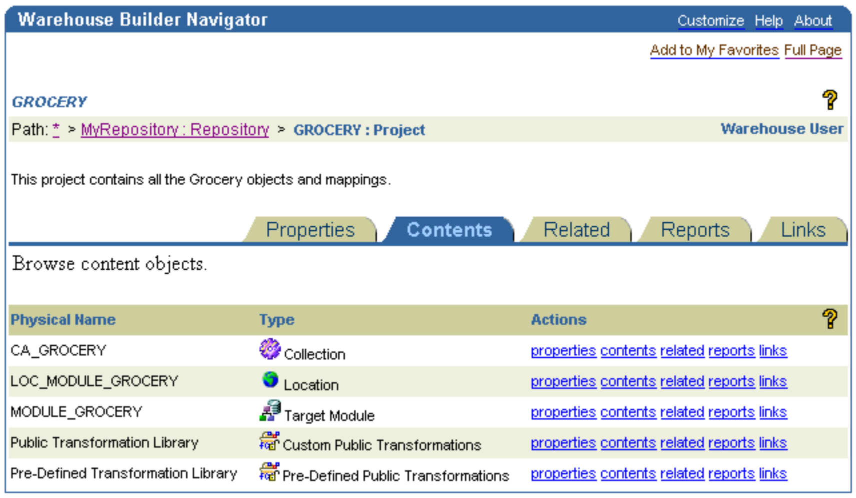
- Navigator Portlet on page 17-1.
- Favorites Portlet on page 17-2.
- Reports Portlet on page 17-3.

Navigator Portlet

The Navigator Portlet enables you to browse repositories accessible to the current user, as shown in Figure 17–1. The Navigator Portlet has the same functionality as the main

Navigator page, except it displays within the area of the portlet. You can click the Full Page link to display the Navigator in full page mode.

Figure 17–1 Navigator Portlet



Warehouse Builder Navigator

GROCERY

Path: * > MyRepository : Repository > GROCERY : Project

This project contains all the Grocery objects and mappings.

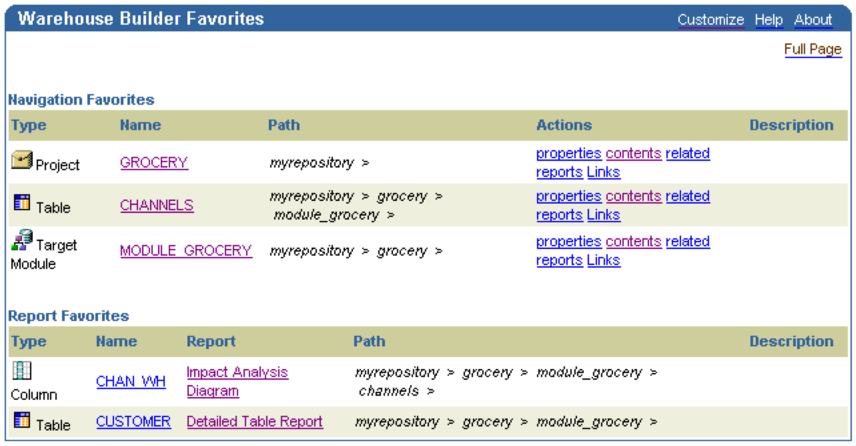
Browse content objects.

Physical Name	Type	Actions
CA_GROCERY	Collection	properties contents related reports links
LOC_MODULE_GROCERY	Location	properties contents related reports links
MODULE_GROCERY	Target Module	properties contents related reports links
Public Transformation Library	Custom Public Transformations	properties contents related reports links
Pre-Defined Transformation Library	Pre-Defined Public Transformations	properties contents related reports links

Favorites Portlet

The Favorites Portlet provides access to your favorite Navigator pages and Reports from your Oracle Portal home page, as shown in Figure 17–2. This portlet does not contain any default favorites. You can select Navigator pages and Reports as favorites using the Add to My Favorites links on the Warehouse Builder Navigator. You can also click the Full Page link to display Favorites in full page mode.

Figure 17–2 Favorites Portlet



Warehouse Builder Favorites

Navigation Favorites

Type	Name	Path	Actions	Description
Project	GROCERY	myrepository >	properties contents related reports Links	
Table	CHANNELS	myrepository > grocery > module_grocery >	properties contents related reports Links	
Target Module	MODULE_GROCERY	myrepository > grocery >	properties contents related reports Links	

Report Favorites

Type	Name	Report	Path	Description
Column	CHAN_WH	Impact Analysis Diagram	myrepository > grocery > module_grocery > channels >	
Table	CUSTOMER	Detailed Table Report	myrepository > grocery > module_grocery >	

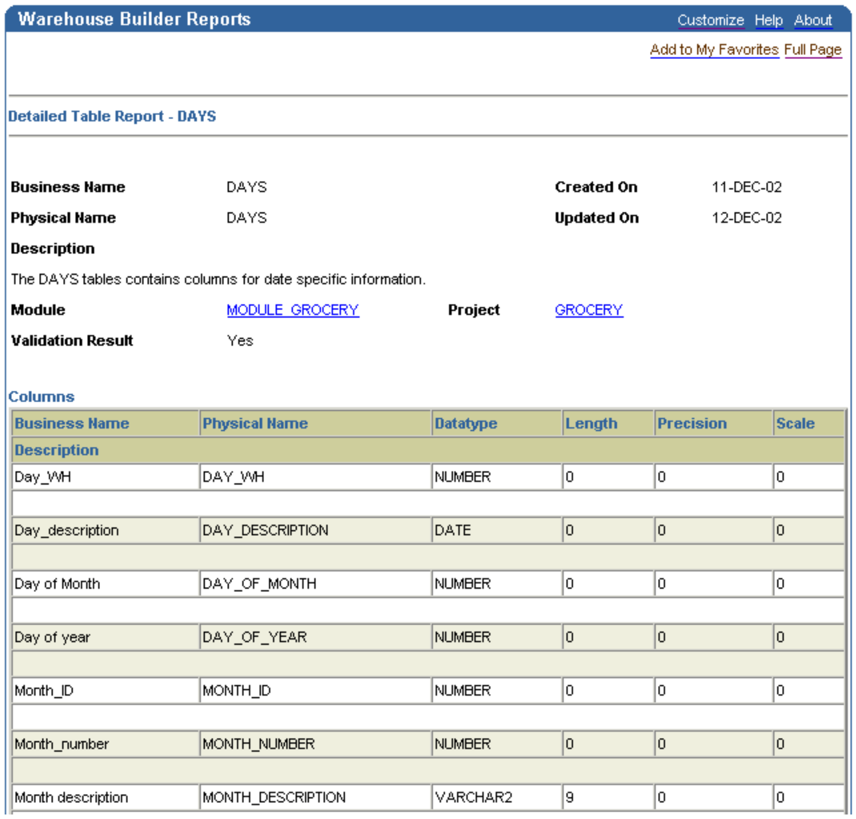
To view your favorite Navigator pages, select an item name under the column Name. You can also use the action links to view the properties, contents, related items, reports, or links associated with this item.

To view your favorite Reports, select a report name under the Report column. If you click the item name under the column Name, the Navigator page for that item displays.

Reports Portlet

The Reports Portlet displays a Warehouse Builder Report on your Oracle Portal home page, as shown in Figure 17–3. This portlet does not contain a default Report. To display a Report, you must add a report to your favorites and use the Customize link on this portlet to add it.

Figure 17–3 Reports Portlet



Detailed Table Report - DAYS

Business Name	DAYS	**Created On**	11-DEC-02
Physical Name	DAYS	**Updated On**	12-DEC-02

Description

The DAYS tables contains columns for date specific information.

Module	MODULE_GROCERY	**Project**	GROCERY
Validation Result	Yes		

Columns

Business Name / Description	Physical Name	Datatype	Length	Precision	Scale
Day_WH	DAY_WH	NUMBER	0	0	0
Day_description	DAY_DESCRIPTION	DATE	0	0	0
Day of Month	DAY_OF_MONTH	NUMBER	0	0	0
Day of year	DAY_OF_YEAR	NUMBER	0	0	0
Month_ID	MONTH_ID	NUMBER	0	0	0
Month_number	MONTH_NUMBER	NUMBER	0	0	0
Month description	MONTH_DESCRIPTION	VARCHAR2	9	0	0

Note: When you have a Reports portlet on your Oracle Portal home page, it refreshes each time you reload the home page. This can delay the home page display.

Using Warehouse Builder Design Browser

Use Warehouse Builder Browser to browse metadata reports and search for specific information.

Starting Warehouse Builder Design Browser

You can start either the client or the Portal version of the Design Browser from within Warehouse Builder client. You can also start either version outside of the Warehouse Builder client.

To start the Design Browser from the Warehouse Builder client:

1. From the menu, select **View** and then **Reports.**

Warehouse Builder launches your internet browser and displays the Warehouse Builder Browser Navigator.

If the internet browser displays and error message, the Design Browser may have been configured incorrectly. Refer to the Oracle Warehouse Builder Installation and Configuration Guide for instrucctions for configuring the Design Browser.

2. Use the Navigator page to define a metadata report.

   The Navigator page displays at root level and lists only your login repository. There is no Administration page. You have all the privileges. All Lineage and Impact Analysis options are now available under the Refresh Dependency Index link. Only English is supported. MLS is available. No custom reports. Favorites are available.

To start the client version of the Design Browser from outside Warehouse Builder client:

1. Start OC4J.

   For Windows: From your desktop **Start** menu, select **Programs**, then ***Your_ Warehouse_Builder_Home***, then **Warehouse Builder**, and **Start OC4J**. A DOS window opens.

   For UNIX: Locate `ORACLE_HOME/owb/bin/unix` and execute `startOwbbInst.sh`.

   A message reading “Oracle9iAS (9.0.2.0.0) Container for J2EE is initialized” appears.

2. Minimize the message window.
3. Start the Design Browser Client.

   For Windows: From your desktop **Start** menu, select **Programs**, then ***Your_ Warehouse_Builder_Home***, then **Warehouse Builder**, then **OWB Design Browser**.

   For UNIX: Locate `ORACLE_HOME/owb/bin/unix` and execute `openDB.sh`.

4. Log in to the Design Browser client by providing the following information:

   HTTP Server Host Port: The default is 7778. Older versions of Oracle HTTP Server have a default of 80.

   DAD: Name of the Warehouse Builder Design Repository you want to browse.

   Repository User Name and Password: These are the User Name and Password assigned to the Design Repository during installation. They may not be the same as your Warehouse Builder User Name and Password.

Starting the Portal Version of the Design Browser

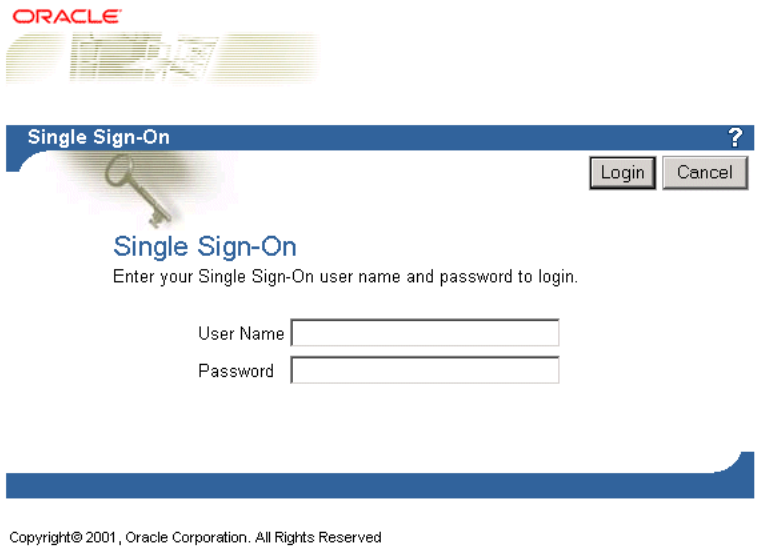
To start Warehouse Builder Design Browser if you have integrated it with Oracle Database Application Server, you need the Oracle Portal internet address and an Oracle Portal user account. Contact your Internet Application Server (iAS) Administrator for the required information.

To start the Warehouse Builder Browser:

1. Open your internet browser and set the URL to your Oracle Portal address.
2. Click **Login.**

   The Warehouse Builder Browser uses a standard Oracle Portal login screen, as shown in Figure 17–4. This is not specific to the Warehouse Builder Browser.

Figure 17–4 Single Sign-On Page


ORACLE

Single Sign-On ?

Login Cancel

Single Sign-On

Enter your Single Sign-On user name and password to login.

User Name

Password

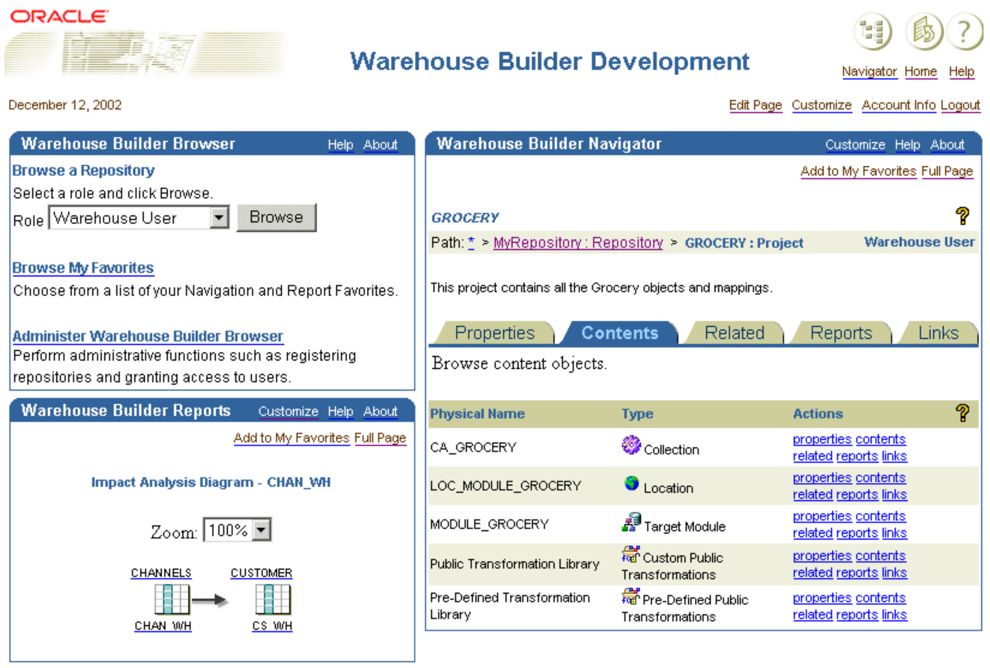
Copyright© 2001, Oracle Corporation. All Rights Reserved

3. Log on to Oracle Portal using a user name and password that has access privileges to the Warehouse Builder Browser portlets. The Internet Application Server (iAS) Administrator is responsible for providing user accounts. Each user has their own user name and password.

   After you log in, the default Oracle Portal home page displays. To display the Warehouse Builder Browser portlets, you must add them to a page such as the Oracle Portal home page and then navigate to that page. Use the Edit Page option to add portlets. For information on adding portlets, see "Adding Portlets" on page 18-4.

   Figure 17–5 shows an example Portal page showing Warehouse Builder portlets.

Figure 17–5 Example Portal Page Showing Warehouse Builder Portlets


ORACLE

Warehouse Builder Development

Navigator Home Help

December 12, 2002

Edit Page Customize Account Info Logout

Warehouse Builder Browser Help About

Browse a Repository

Select a role and click Browse.

Role Warehouse User Browse

Browse My Favorites

Choose from a list of your Navigation and Report Favorites.

Administer Warehouse Builder Browser

Perform administrative functions such as registering repositories and granting access to users.

Warehouse Builder Reports Customize Help About

Add to My Favorites Full Page

Impact Analysis Diagram - CHAN_WH

Zoom: 100%

Warehouse Builder Navigator Customize Help About

Add to My Favorites Full Page

GROCERY

Path: * > MyRepository : Repository > GROCERY : Project Warehouse User

This project contains all the Grocery objects and mappings.

Properties Contents Related Reports Links

Browse content objects.

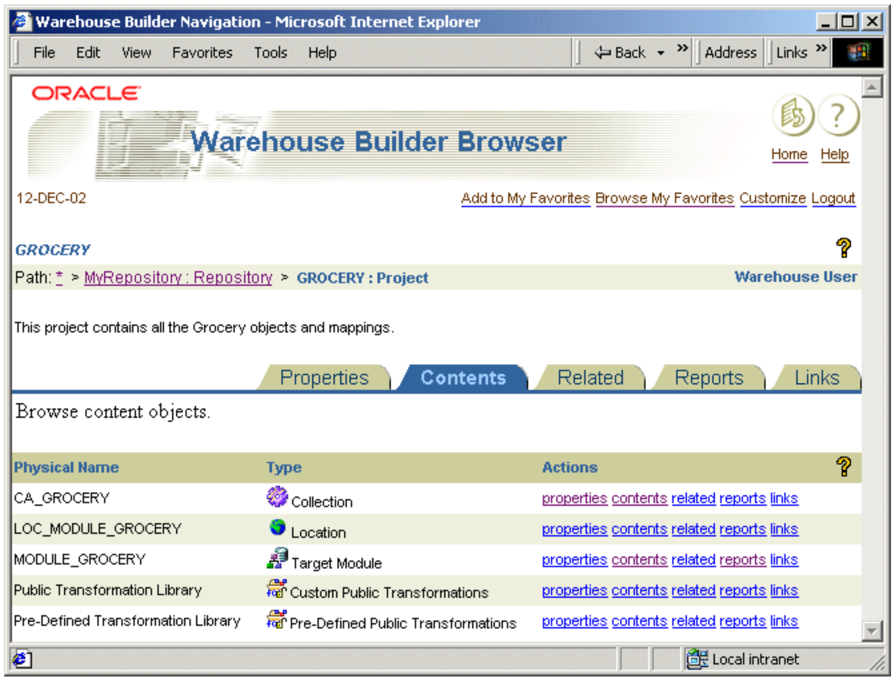
Physical Name	Type	Actions
CA_GROCERY	Collection	properties contents related reports links
LOC_MODULE_GROCERY	Location	properties contents related reports links
MODULE_GROCERY	Target Module	properties contents related reports links
Public Transformation Library	Custom Public Transformations	properties contents related reports links
Pre-Defined Transformation Library	Pre-Defined Public Transformations	properties contents related reports links

Navigating Warehouse Builder Browser

After you have started the Warehouse Builder Browser, you first see the Navigator page, as shown in Figure 17–6. This page includes the following parts:

- Header Section
- Path Section
- Tabbed sections containing the Properties Tab, Contents Tab, Related Tab, Reports Tab, and Links Tab.

Figure 17–6 Warehouse Builder Navigator


Warehouse Builder Navigation - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Address Links
ORACLE
Warehouse Builder Browser
Home Help
12-DEC-02
Add to My Favorites Browse My Favorites Customize Logout
GROCERY
Path: * > MyRepository : Repository > GROCERY : Project
Warehouse User
Properties
Contents
Related
Reports
Links
Local intranet

This project contains all the Grocery objects and mappings.

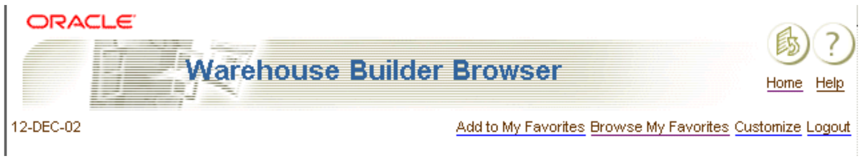
Browse content objects.

Physical Name	Type	Actions
CA_GROCERY	Collection	properties contents related reports links
LOC_MODULE_GROCERY	Location	properties contents related reports links
MODULE_GROCERY	Target Module	properties contents related reports links
Public Transformation Library	Custom Public Transformations	properties contents related reports links
Pre-Defined Transformation Library	Pre-Defined Public Transformations	properties contents related reports links

Header Section

Every Warehouse Builder Browser page displays a header at the top of the page similar to the header shown in Figure 17–7.

Figure 17–7 Navigator Header

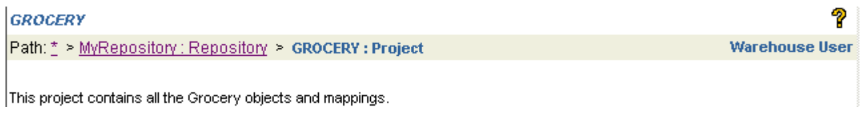

ORACLE
Warehouse Builder Browser
Home Help
12-DEC-02
Add to My Favorites Browse My Favorites Customize Logout

The header contains the title of the page and useful links. To the right of the title is the Home link. Below the title are links for Add to My Favorites, Browse My Favorites, Customize, and Logout.

Path Section

The Navigator page displays information about the currently selected path in the middle of the page.

Figure 17–8 Item Definition


GROCERY

Path: * > MyRepository : Repository > GROCERY : Project   Warehouse User

This project contains all the Grocery objects and mappings.

The path section contains the following:

- **Item Name:** On the left, Warehouse Builder Browser displays the item name. This is the name of the item to which you have navigated. In Figure 17–8, this is a project named `GROCERY`.
- **Help/Question Mark:** On the far right, Warehouse Builder Browser displays the help icon. The Help link opens online help for the Navigation Page.
- **Path:** Below the item name, Warehouse Builder Browser displays the path. The Path shows you the context of the item. It lists the items in the hierarchy and also shows the type of each item. You can click an item in the path to make it the current item. In Figure 17–8, the path is `Path: MyRepository: Repository > GROCERY:Project`.
- **Role:** Below the help icon on the right, Warehouse Builder Browser displays the role for the current user. In Figure 17–8 the role is `Warehouse User`.

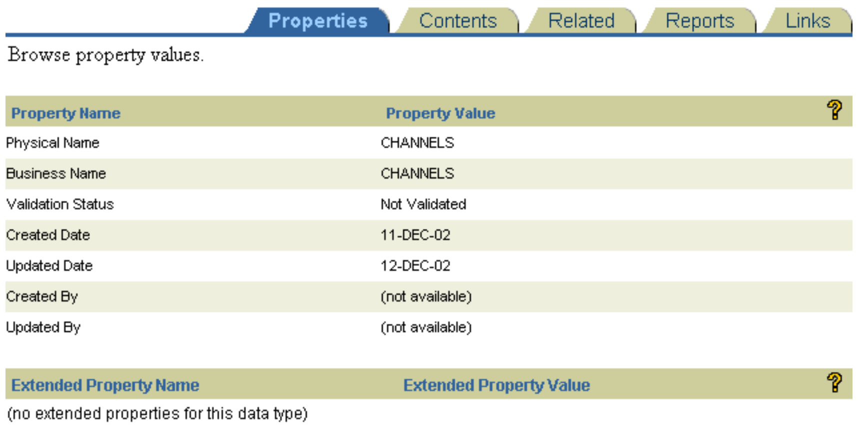
Body Section

Every Warehouse Builder Browser page displays detailed information at bottom of the page. Select a tab to view details specific to the tab.

Properties Tab

The Properties tab displays property name-value pairs for the current item, as shown in Figure 17–9. The property names are different depending on the type of the current item.

Figure 17–9 Properties Tab



Browse property values.

Property Name	Property Value
Physical Name	CHANNELS
Business Name	CHANNELS
Validation Status	Not Validated
Created Date	11-DEC-02
Updated Date	12-DEC-02
Created By	(not available)
Updated By	(not available)

Extended Property Name	Extended Property Value
(no extended properties for this data type)	

Table 17–1 describes the columns displayed in the Properties tab.

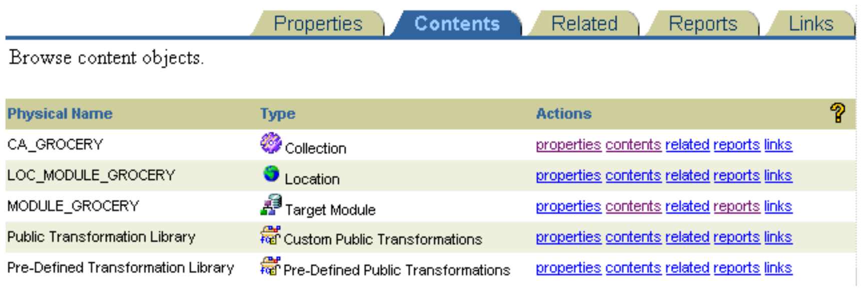
Table 17–1 Property Column Values

Column Name	Value
Property Name	The name of the property as defined in the Warehouse Builder Public Views. For more details on the Public views, see "Creating Custom Reports" on page 18-21.
Property Value	The value of the property from the Warehouse Builder repository for the current item.

Contents Tab

The Contents tab lists the items that the current item contains. For example, if the current item is a table, then the Contents tab lists the columns, keys, and foreign keys. Use this tab when you drill down into the Warehouse Builder repository. In Figure 17–10, the current item is the Warehouse Builder repository and the list of items that the repository contains. Table 17–2 lists the columns in the Contents tab and brief descriptions of their values.

Figure 17–10 Contents Tab



Browse content objects.

Physical Name	Type	Actions
CA_GROCERY	Collection	properties contents related reports links
LOC_MODULE_GROCERY	Location	properties contents related reports links
MODULE_GROCERY	Target Module	properties contents related reports links
Public Transformation Library	Custom Public Transformations	properties contents related reports links
Pre-Defined Transformation Library	Pre-Defined Public Transformations	properties contents related reports links

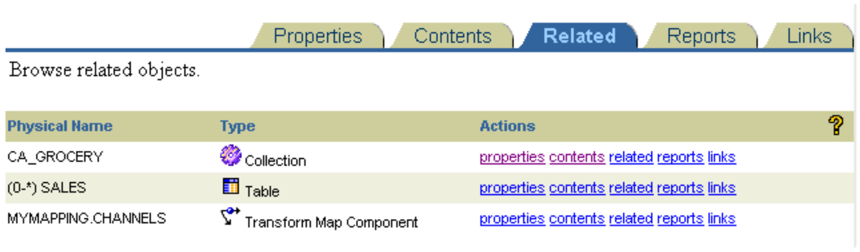
Table 17–2 Contents Column Values

Column Name	Value
Physical Name	The name of the item.
Type	An icon showing a graphical representation of the item type along with the item type.
Actions	A set of links that go to the tab named on the link for the current item.
	Properties: Select Properties to make the item on that row the current item and to view the Properties tab for that item.
	Contents: Select Contents to make the item on that row the current item and to view the Contents tab for that item.
	Related: Select Related to make the item on that row the current item and to view the Related tab for that item.
	Reports: Select Reports to make the item on that row the current item and to view the Reports tab for that item.
	Links: Select Links to make the item on that row the current item and to view the Links tab for that item.

Related Tab

The Related tab enables you to view relationships between items, as shown in Figure 17–11. For example, if the current item is a table, then you can identify tables related by foreign keys. You can use this tab to drill down into items in the repository. When you make one of these items the current item, the path switches to represent the path of the new item.

Figure 17–11 Related Tab



Browse related objects.

Physical Name	Type	Actions
CA_GROCERY	Collection	properties contents related reports links
(0-*) SALES	Table	properties contents related reports links
MYMAPPING.CHANNELS	Transform Map Component	properties contents related reports links

Table 17–3 lists the columns in the Related tab and brief descriptions of their values.

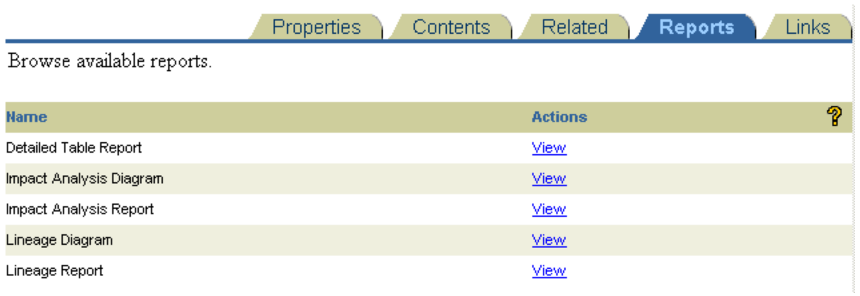
Table 17–3 Related Column Values

Column Name	Value
Name	The name of the item.
Type	An icon showing a graphical representation of the item type along with the item type name.
Actions	A set of links that go to the tab named on the link for the current item. **Properties:** Select Properties to make the item on that row the current item and to view the Properties tab for that item. **Contents:** Select Contents to make the item on that row the current item and to view the Contents tab for that item. **Related:** Select Related to make the item on that row the current item and to view the Related tab for that item. **Reports:** Select Reports to make the item on that row the current item and to view the Reports tab for that item. **Links:** Select Links to make the item on that row the current item and to view the Links tab for that item.

Reports Tab

The Reports tab shows you the available reports for the current item, as shown in Figure 17–12. The reports you can view from the Reports tab depend upon your user role. A Warehouse User sees a different set of reports than a Warehouse Engineer. Table 17–4 lists the columns in the Reports tab with brief descriptions of their values.

Figure 17–12 Reports Tab



Browse available reports.

Name	Actions
Detailed Table Report	View
Impact Analysis Diagram	View
Impact Analysis Report	View
Lineage Diagram	View
Lineage Report	View

Table 17–4 Reports Column Values

Column Name	Value
Name	The name of the report.

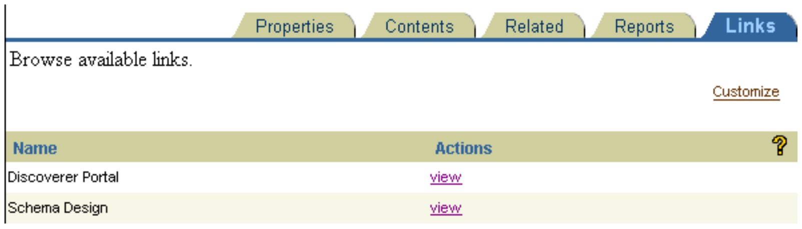
Table 17–4 (Cont.) Reports Column Values

Column Name	Value
Actions	**View:** Displays the report.

Links Tab

The Links tab enables you to associate other sources of information with an object or a type of object, as shown in Figure 17–13. You can link to anything that can be described with a URL, such as documents stored in Oracle Portal content areas or other web pages.

Figure 17–13 Links Tab



Browse available links.

Customize

Name	Actions
Discoverer Portal	view
Schema Design	view

Use the **Customize** link on the Links tab to edit, delete, or add links.

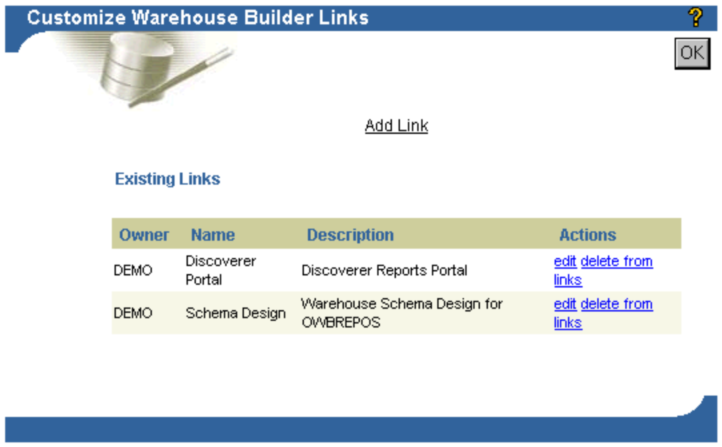
Note: You can only edit or delete links you created. You cannot edit or delete the links created by other repository users.

To edit a link:

1. Navigate to the object or object type that the link is associated with and select the Links tab.
2. From the Links tab, select **Customize.**

   The Customize Warehouse Builder Links page displays the Existing Links, as shown in Figure 17–14.

Figure 17–14 Customize Warehouse Builder Links Page



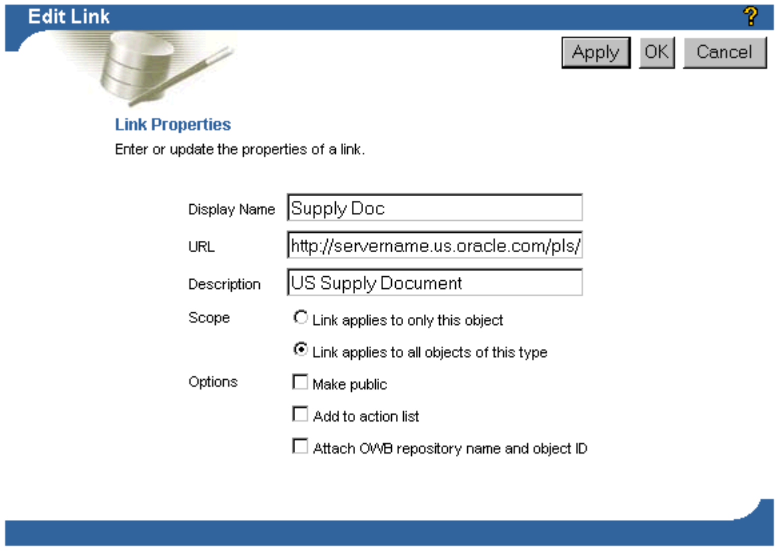
Existing Links

Owner	Name	Description	Actions
DEMO	Discoverer Portal	Discoverer Reports Portal	edit delete from links
DEMO	Schema Design	Warehouse Schema Design for OWBREPOS	edit delete from links

3. Select **edit** for the link you want to edit.

   The Edit Link page displays the current link settings, as shown in Figure 17–15.

Figure 17–15 Edit Link Page



Link Properties

Enter or update the properties of a link.

Display Name	Supply Doc
URL	http://servername.us.oracle.com/pls/
Description	US Supply Document
Scope	○ Link applies to only this object
	● Link applies to all objects of this type
Options	☐ Make public
	☐ Add to action list
	☐ Attach OWB repository name and object ID

4. Edit the link properties and click **OK.**

To delete a link:

1. Navigate to the object or object type that the link is associated with and select the Links tab.
2. From the Links tab, select **Customize.**

   The Customize Warehouse Builder Links page displays the Existing Links.
3. Select **delete from links** for the link you want to delete.

   The link is deleted and the Customize Warehouse Builder Links page displays without the deleted link in the list of existing links.
4. Click **OK** to go back to the Links tab.

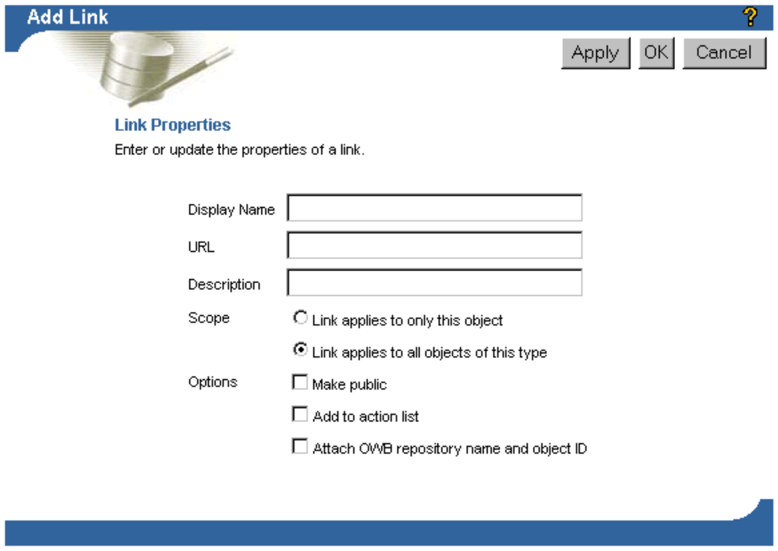
To add a link:

1. Navigate to the object or object type that you want to associate the new link with and select the Links tab.
2. From the Links tab, select **Customize.**

   The Customize Warehouse Builder Links page displays the Existing Links.
3. Select **Add Link.**

   The Add Link page displays, as shown in Figure 17–16.

Figure 17–16 Add Link Page



Link Properties

Enter or update the properties of a link.

Display Name	
URL	
Description	
Scope	○ Link applies to only this object
	● Link applies to all objects of this type
Options	☐ Make public
	☐ Add to action list
	☐ Attach OWB repository name and object ID

4. Specify the link properties using Table 17–5 and click **OK.**

Table 17–5 Link Properties

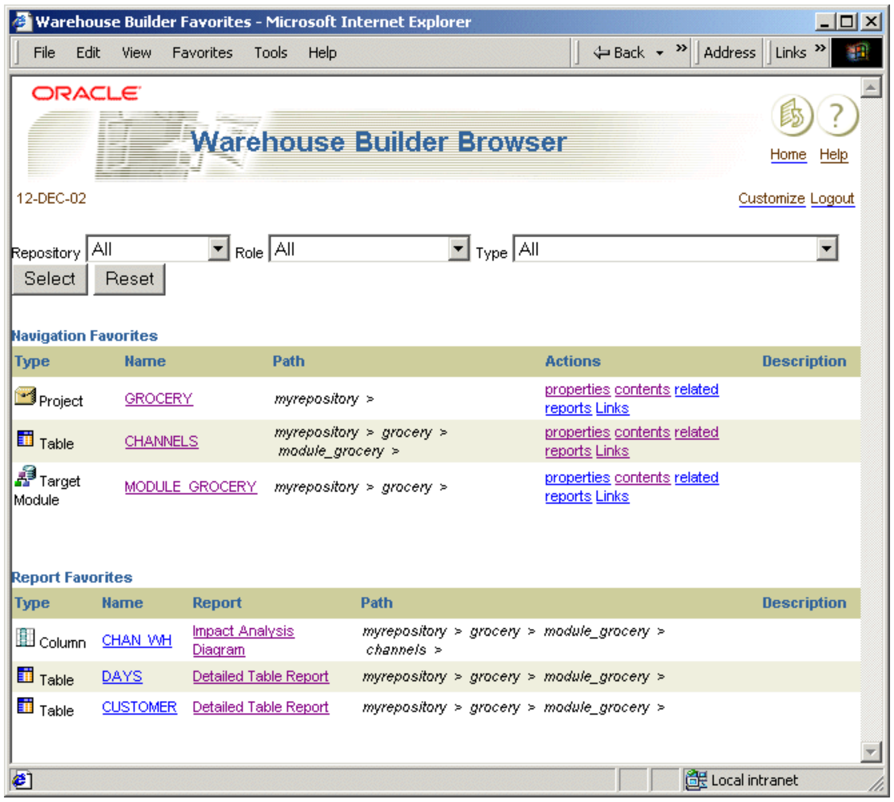
Field	Description
Display Name	Type a name for the link.
URL	For an external Web site, provide the complete URL. For files that are in an Oracle Portal content area, use the Oracle Portal Navigator to reach the content area where the file is listed. Right-click the file and copy the URL location. Then return to the Add Link page and paste the URL into the URL field.
Description	Type an optional description for the link.
Scope	Choose one of the options: ■ Link applies to only this object: For example, if the current item is the Customer Table, this option makes this link available to only the Customer Table. ■ Link applies to all objects of this type: For example, if the current item is the Customer Table, this option makes this link available to all tables.
Options	Choose from the following options: ■ Make Public: Select this option if you want other users to access this link. ■ Add to action list: Select this option to add a link on the Contents tab for the current item. ■ Attach Warehouse Builder repository name and object ID: Select this option to attach the Warehouse Builder repository name and object ID to the link.

Browsing Favorites

Select **Browse My Favorites** to view your Warehouse Builder Favorites page. Every navigation page and every report has an Add to My Favorites link on the top right corner. When you select this link, a link to the current report or Navigation page is added to your Favorites list. You can also use the features on the Favorites pages to delete items from the list and to change the formatting of the list.

The Favorites Page contains a selection of reports and navigation that you select, as shown in Figure 17–17.

Figure 17–17 Warehouse Builder Favorites Page


Warehouse Builder Favorites - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address
Links
ORACLE
Warehouse Builder Browser
Home Help
Customize Logout
Repository All
Role All
Type All
Select Reset
Local intranet

12-DEC-02

Navigation Favorites

Type	Name	Path	Actions	Description
Project	GROCERY	myrepository >	properties contents related reports Links	
Table	CHANNELS	myrepository > grocery > module_grocery >	properties contents related reports Links	
Target Module	MODULE_GROCERY	myrepository > grocery >	properties contents related reports Links	

Report Favorites

Type	Name	Report	Path	Description
Column	CHAN_WH	Impact Analysis Diagram	myrepository > grocery > module_grocery > channels >	
Table	DAYS	Detailed Table Report	myrepository > grocery > module_grocery >	
Table	CUSTOMER	Detailed Table Report	myrepository > grocery > module_grocery >	

General Page Information

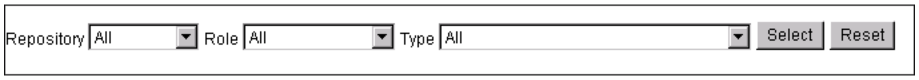
The top of every Warehouse Builder Browser page has a header that defines the type of page you are looking at. The header has links to your default home page, help page, customize page, and log out.

If you want to customize your Favorites page, select the Customize link. For details about customizing your page, see "Customizing Your Favorites Page" on page 17-14.

Favorites Filter

You can narrow the content of the two Favorite tables by selecting a particular Repository, Role, or Item Type. Each of these Items has an All option. Figure 17–18 shows the filter on the Favorites page.

Figure 17–18 Favorites Filter

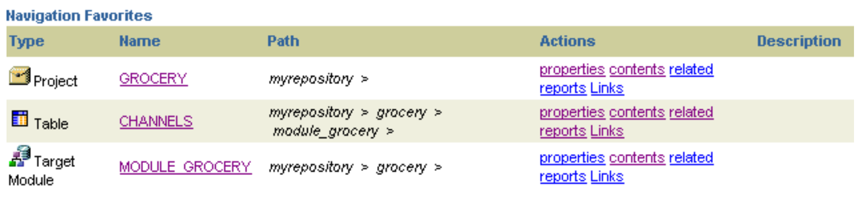


Navigation Favorites

The Favorites page displays Navigation Favorites in the following format. Figure 17–19 shows the Navigation Favorties page.

- **Type:** Type shows the item type and an icon representing the type.
- **Name:** The name of the item. Select the name to view the Navigation content page for that item.
- **Path:** The Path shows the fully qualified name of the item as it appears in the Path field on the Navigator page. You can remove this column using the Favorites Customize Options Page.
- **Actions:** Actions provide links to tabs in the Navigator page for that item.
- **Description:** Description shows the optional descriptive text you added to a favorite in the Favorites Customize page.

Figure 17–19 Navigation Favorites



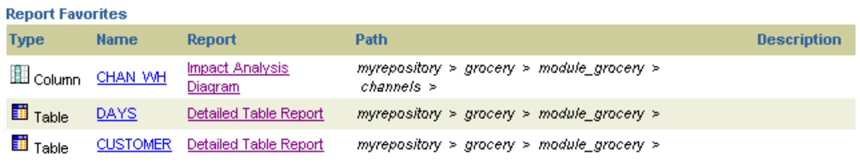
Navigation Favorites

Type	Name	Path	Actions	Description
Project	GROCERY	myrepository >	properties contents related reports Links	
Table	CHANNELS	myrepository > grocery > module_grocery >	properties contents related reports Links	
Target Module	MODULE_GROCERY	myrepository > grocery >	properties contents related reports Links	

Report Favorites

The Favorites page displays Report Favorites in the same format as Navigation Favorites. Figure 17–20 shows the Navigation Favorites page.

Figure 17–20 Reports Favorites



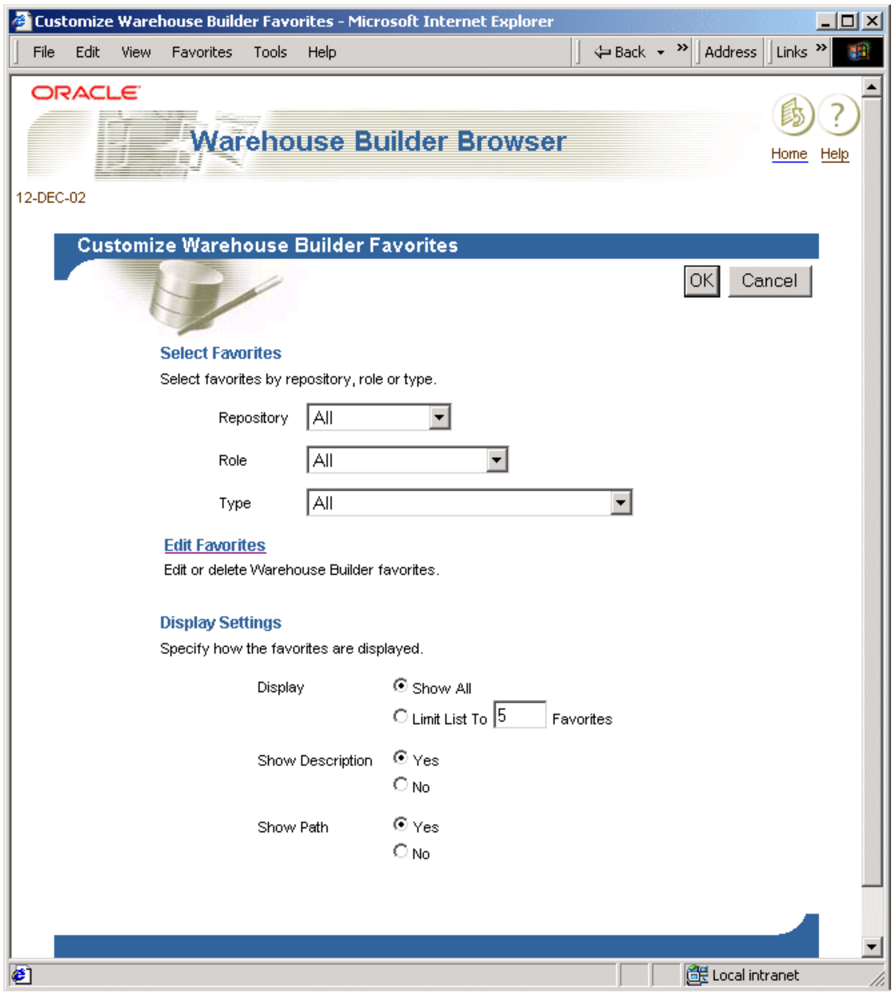
Report Favorites

Type	Name	Report	Path	Description
Column	CHAN_WH	Impact Analysis Diagram	myrepository > grocery > module_grocery > channels >	
Table	DAYS	Detailed Table Report	myrepository > grocery > module_grocery >	
Table	CUSTOMER	Detailed Table Report	myrepository > grocery > module_grocery >	

Customizing Your Favorites Page

To customize your favorites page, select **Options** on the Favorites Customize page. The Customize Display Options page displays, as shown in Figure 17–21 and Figure 17–22. Use this page to change the layout of your favorites on the Favorites page.

Figure 17–21 Favorites Customize Display Options Page


Customize Warehouse Builder Favorites - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Address Links
ORACLE
Warehouse Builder Browser
Home Help
Local intranet

12-DEC-02

Customize Warehouse Builder Favorites

OK Cancel

Select Favorites

Select favorites by repository, role or type.

Repository	All
Role	All
Type	All

Edit Favorites

Edit or delete Warehouse Builder favorites.

Display Settings

Specify how the favorites are displayed.

Display: [✓ Show All] Limit List To 5 Favorites

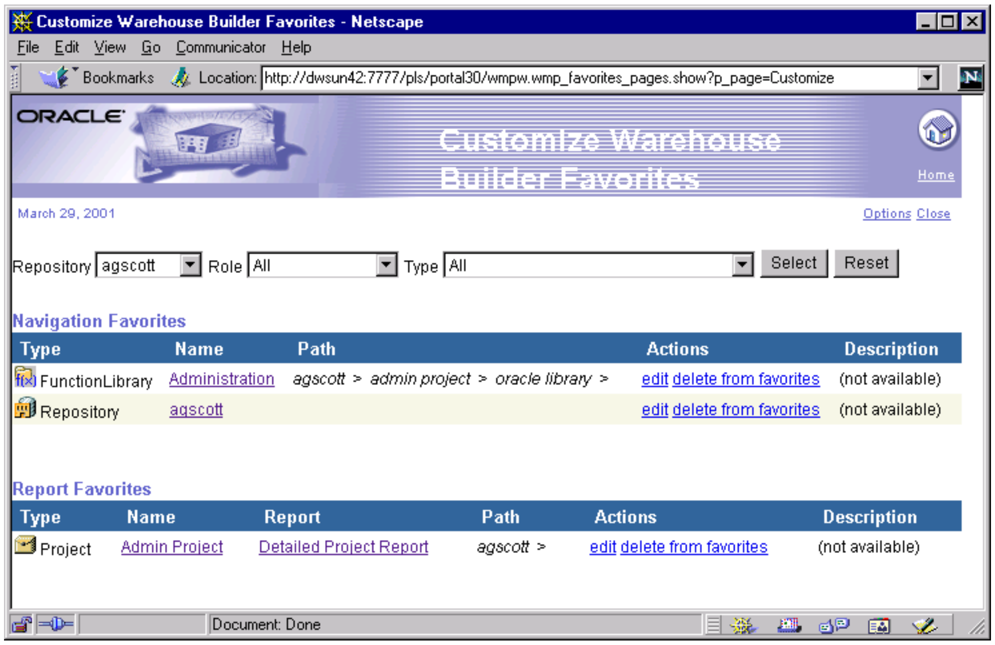
Show Description: [✓ Yes] No

Show Path: [✓ Yes] No

This page contains the following options:

- **Apply:** Applies the changes without returning to the previous page.
- **OK:** Applies the changes and returns to the previous page.
- **Cancel:** Ignores any changes and returns to the previous page.
- **Show:** Sets the maximum number of rows to display in the tables. If there are more items than can be shown, you click **Next** and **Previous** to see the remaining items.
- **Show Descriptions:** Select **Yes** to show the description column or **No** to hide the column.
- **Show Path:** Yes, shows the path column. No, hides the column.

Figure 17–22 Customize Favorites Page


Customize Warehouse Builder Favorites - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://dwsun42:7777/pls/portal30/wmpw.wmp_favorites_pages.show?p_page=Customize
ORACLE
Customize Warehouse Builder Favorites
Home
March 29, 2001
Options Close
Repository agscott Role All Type All Select Reset
Document: Done

Navigation Favorites

Type	Name	Path	Actions	Description
FunctionLibrary	Administration	agscott > admin project > oracle library >	edit delete from favorites	(not available)
Repository	agscott		edit delete from favorites	(not available)

Report Favorites

Type	Name	Report	Path	Actions	Description
Project	Admin Project	Detailed Project Report	agscott >	edit delete from favorites	(not available)

Table 17–6 describes the columns displayed in the Customize Favorites page.

Table 17–6 Customize Favorites Page Actions

Action	Description
Edit	Use the Edit action to enter a description for a favorite. The description appears as a column in the Favorites list. Choose **OK** or **Cancel** to return to the Customize Page.
Delete from Favorites	Use this action to remove an entry from the Favorites table. This does not remove the item from the Warehouse Builder Repository.

Viewing Warehouse Builder Reports

You can view Browser reports for Warehouse Builder repositories using the following tools:

- Warehouse Builder Client
- Warehouse Builder Browser

Viewing Reports in the Warehouse Builder Client

You can view metadata reports using the Warehouse Builder client. However, you must first ensure that Oracle Database Portal is installed. To view metadata reports, first complete the steps described in "Configuring the Warehouse Builder Client" on page 18-20.

To access reports from the Warehouse Builder client:

1. Open a project in Warehouse Builder and, from the navigation tree, expand any node and select any object. Or, open any editor such as the Mapping Editor or the Process Flow Editor.
2. From the **View** menu, select **Reports.**

Warehouse Builder launches the logon screen for the Oracle Portal site you defined in the Preferences window. For more information, see "Starting Warehouse Builder" on page 2-9.

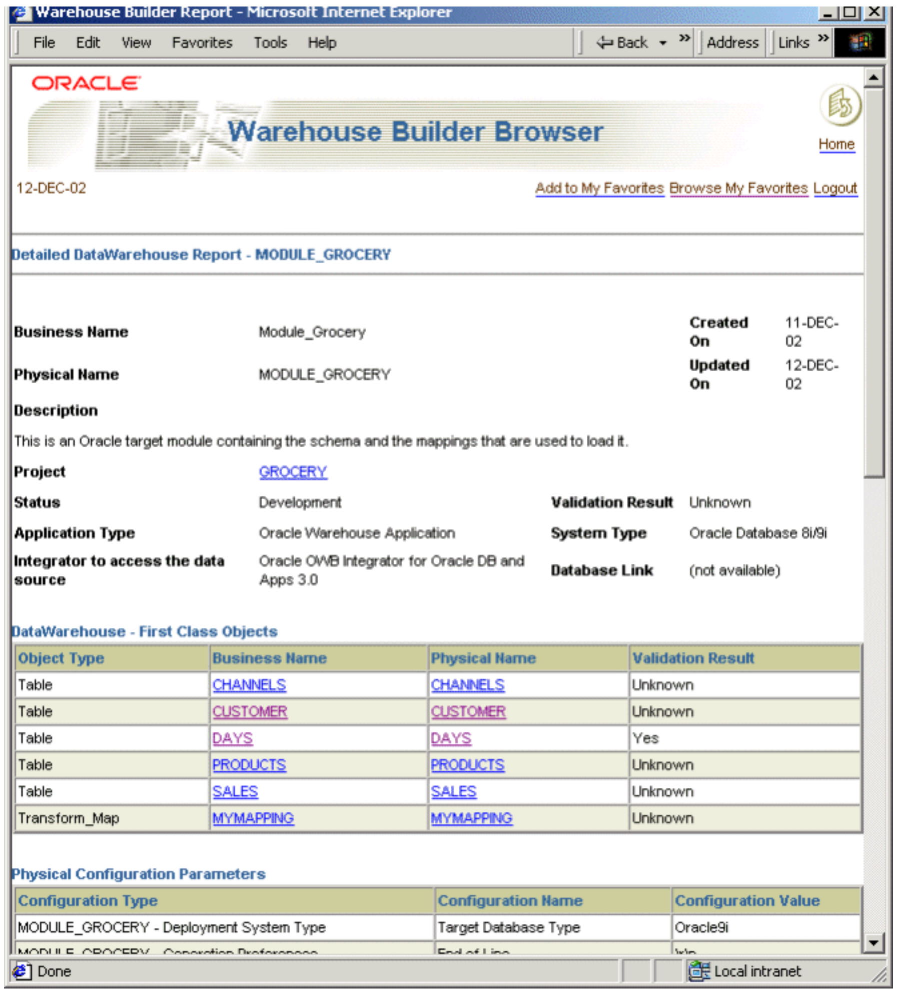
3. Enter your Internet Application Server (*i*AS) Single Sign-On user name and password.

   The Warehouse Builder Browser displays a list of reports available for the selected object as shown in Figure 17–25 on page 17-19.

4. Click the **view** link for a report.

   The report displays in a separate window. Figure 17–23 shows an example of the Table Summary Report.

Figure 17–23 Warehouse Builder Report


Warehouse Builder Report - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address
Links
ORACLE
Warehouse Builder Browser
Home
Done
Local intranet

12-DEC-02

Add to My Favorites Browse My Favorites Logout

Detailed DataWarehouse Report - MODULE_GROCERY

Business Name	Module_Grocery	**Created On**	11-DEC-02
Physical Name	MODULE_GROCERY	**Updated On**	12-DEC-02

Description

This is an Oracle target module containing the schema and the mappings that are used to load it.

Project	GROCERY		
Status	Development	**Validation Result**	Unknown
Application Type	Oracle Warehouse Application	**System Type**	Oracle Database 8i/9i
Integrator to access the data source	Oracle OWB Integrator for Oracle DB and Apps 3.0	**Database Link**	(not available)

DataWarehouse - First Class Objects

Object Type	Business Name	Physical Name	Validation Result
Table	CHANNELS	CHANNELS	Unknown
Table	CUSTOMER	CUSTOMER	Unknown
Table	DAYS	DAYS	Yes
Table	PRODUCTS	PRODUCTS	Unknown
Table	SALES	SALES	Unknown
Transform_Map	MYMAPPING	MYMAPPING	Unknown

Physical Configuration Parameters

Configuration Type	Configuration Name	Configuration Value
MODULE_GROCERY - Deployment System Type	Target Database Type	Oracle9i
MODULE_GROCERY - Generation Preferences	End of Line	\r\n

To print the report, select the print operation from the browser menu or toolbar.

Viewing a Custom Report

To view a custom report from the Warehouse Builder Browser:

1. Log on to the Warehouse Builder Browser, select the role that has access to the report, and click **Browse.**
2. Navigate to an instance of the report type and select the Reports tab.

3. The custom report appears in the reports list. Select the **view** action link.

   The report displays.

Viewing Reports in Warehouse Builder Browser

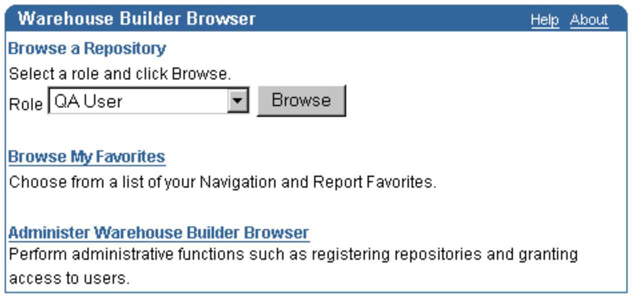
You can view reports from the Warehouse Builder Browser without installing Warehouse Builder Client. You start by launching Oracle Portal. From there you can select the Warehouse Builder Browser portlets.

To view metadata reports in Warehouse Builder Browser:

1. Select a Warehouse Builder Repository from the Repository drop-down list in the Warehouse Builder Browser Launcher portlet, as shown in Figure 17–24.

   The Repository list contains repositories that have been previously set up by your Warehouse Administrator using the Warehouse Builder Browser Administration pages.

Figure 17–24 Selecting a Repository



Warehouse Builder Browser

Help About

Browse a Repository

Select a role and click Browse.

Role: QA User

Browse

Browse My Favorites

Choose from a list of your Navigation and Report Favorites.

Administer Warehouse Builder Browser

Perform administrative functions such as registering repositories and granting access to users.

2. Select the role you want to use: Warehouse Engineer, QA User, Warehouse User.

   The role determines the reports you can access. Roles are set up by your Warehouse Administrator in the Warehouse Builder Browser Administration pages.

3. Click **Browse.**

   This displays the Warehouse Builder Navigation Pages. You can then navigate around your repository and find the item to report on.

4. Select the Reports tab.

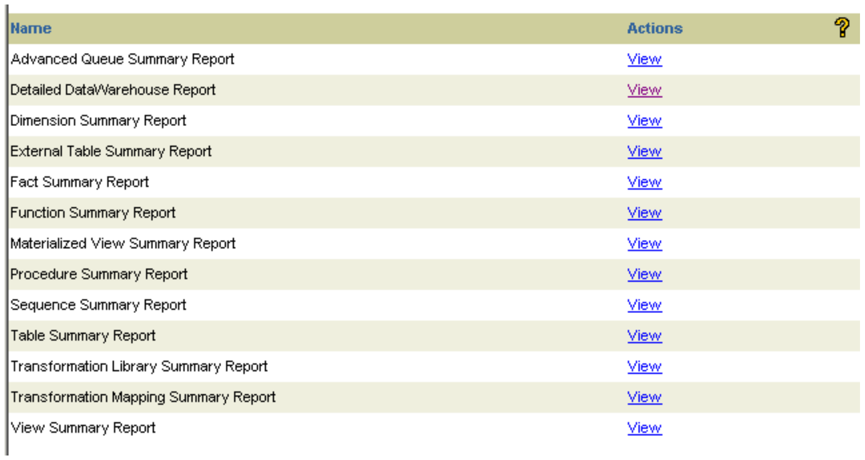
   Warehouse Builder Browser displays the reports available for that item as shown in Figure 17–25 on page 17-19.

5. Select the **view** link to view the report.

Warehouse Builder Metadata Reports

Warehouse Builder contains a set of reports that use Warehouse Builder public views, as shown in Figure 17–25.

Figure 17–25 Warehouse Builder Metadata Reports



Name	Actions
Advanced Queue Summary Report	View
Detailed DataWarehouse Report	View
Dimension Summary Report	View
External Table Summary Report	View
Fact Summary Report	View
Function Summary Report	View
Materialized View Summary Report	View
Procedure Summary Report	View
Sequence Summary Report	View
Table Summary Report	View
Transformation Library Summary Report	View
Transformation Mapping Summary Report	View
View Summary Report	View

Use these reports to examine your metadata. The following reports are available:

- Summary Reports on page 17-19
- Detailed Reports on page 17-20
- Implementation Reports on page 17-22
- Lineage and Impact Analysis Reports and Diagrams on page 17-22
- Lineage and Impact Analysis Diagrams on page 17-23

For related information, see:

- Creating Custom Reports on page 18-21

Summary Reports

You can run summary reports against source modules and target systems. Each summary report presents a list of items contained within the module. The type of items displayed is determined by the summary report you select. 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 also displays. Selecting the name of an item displays the detailed report for that item.

You can run the summary reports 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

Figure 17–26 shows a section from a Dimension Summary Report.

Figure 17–26 Dimension Summary Report

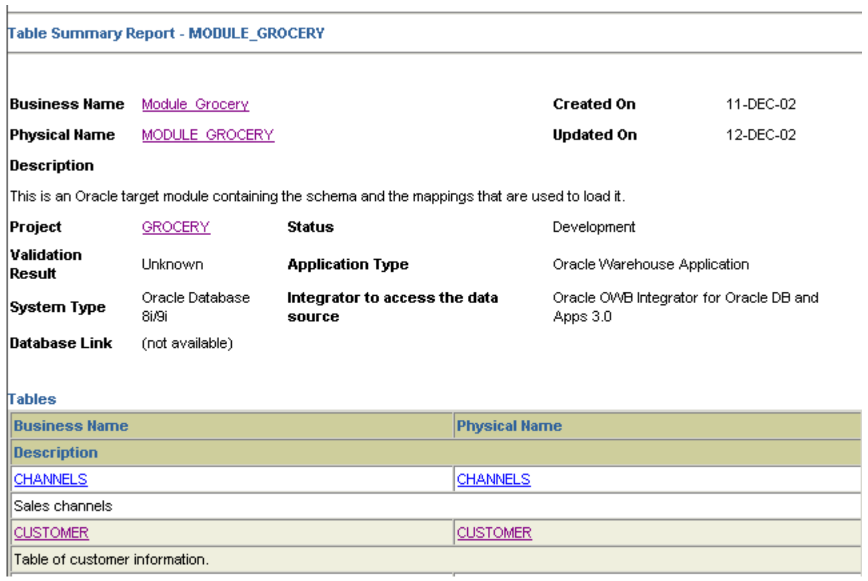


Table Summary Report - MODULE_GROCERY

Business Name	Module_Grocery	**Created On**	11-DEC-02
Physical Name	MODULE_GROCERY	**Updated On**	12-DEC-02

Description

This is an Oracle target module containing the schema and the mappings that are used to load it.

Project	GROCERY	**Status**	Development
Validation Result	Unknown	**Application Type**	Oracle Warehouse Application
System Type	Oracle Database 8i/9i	**Integrator to access the data source**	Oracle OWB Integrator for Oracle DB and Apps 3.0
Database Link	(not available)		

Tables

Business Name	Physical Name
Description	
CHANNELS	CHANNELS
Sales channels	
CUSTOMER	CUSTOMER
Table of customer information.	

Detailed Reports

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. Table 17–7 list the Detailed Reports and their contents.

Table 17–7 Detailed Reports

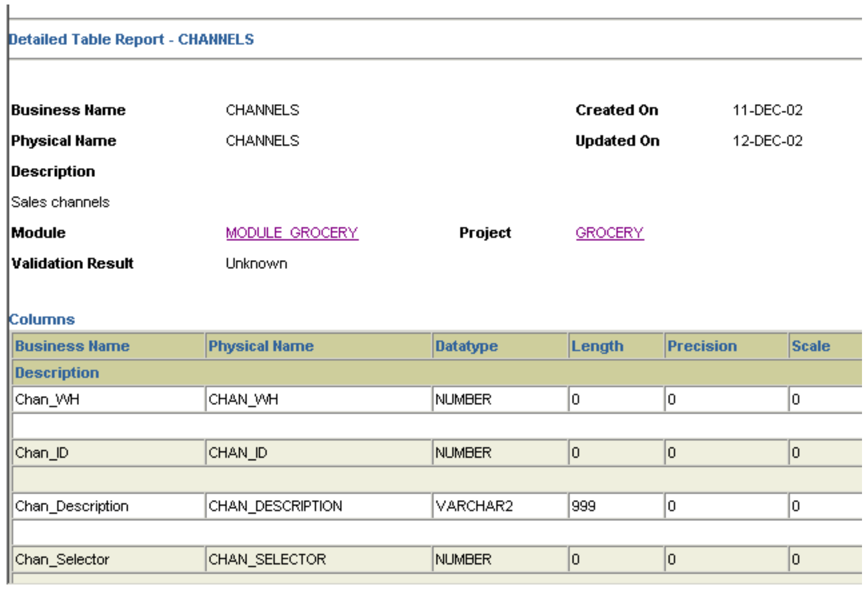
Report	Contents
Detailed Advanced Queue Report	
Detailed Collection Report	Items in a collection.
Detailed Connector Report	
Detailed Cube Implementation Report	
Detailed Cube Report	Measures, Dimensions, and physical configuration parameters
Detailed Dimension Implementation Report	
Detailed Dimension Report	Hierarchies, Levels, Level Attributes, and physical configuration parameters
Detailed External Table Report	

Table 17–7 (Cont.) Detailed Reports

Report	Contents
Detailed File Module Report	
Detailed File Report	Records and physical configuration parameters
Detailed Function Library Report	Functions and physical configuration parameters
Detailed Function Report	Parameters and Function Implementations
Detailed Installation Report	Projects
Detailed Location Report	
Detailed Materialized View Report	Columns, Keys, Foreign Keys, and physical configuration parameters
Detailed Module Report	Cubes, Dimensions, Function Libraries, Materialized Views, Sequences, Tables, Views, Transform Maps, Files, and physical configuration parameters
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	Modules and Business Areas
Detailed Record Report	Fields
Detailed Repository Report	
Detailed Runtime Location Report	
Detailed Sequence Report	Columns and physical configuration parameters
Detailed Table Function Report	
Detailed Table Report	Columns, Keys, Foreign Keys, and physical configuration parameters
Detailed Transform Map Report	Object Uses, Map Components, Item Maps, and physical configuration parameters
Detailed Transform Map Report	
Detailed View Report	Columns, Keys, Foreign Keys, and physical configuration parameters

Figure 17–27 shows a section from a Detailed Cube Report.

Figure 17–27 Detailed Cube Report



Detailed Table Report - CHANNELS

Business Name	CHANNELS	Created On	11-DEC-02
Physical Name	CHANNELS	Updated On	12-DEC-02
Description			
Sales channels			
Module	MODULE_GROCERY	Project	GROCERY
Validation Result	Unknown		

Columns

Business Name	Physical Name	Datatype	Length	Precision	Scale
Description					
Chan_WH	CHAN_WH	NUMBER	0	0	0
Chan_ID	CHAN_ID	NUMBER	0	0	0
Chan_Description	CHAN_DESCRIPTION	VARCHAR2	999	0	0
Chan_Selector	CHAN_SELECTOR	NUMBER	0	0	0

Implementation Reports

Implementation Reports can be run on Dimensions and Cubes. They provide information on how physical objects are used to implement logical objects. Table 17–8 lists the available Implementation Reports.

Table 17–8 Implementation Reports

Report	Content
Cube Implementation Report	Tables that implement the Cube, Columns that implement the Measures, Foreign Keys that implement the Cube to Dimension Use
Dimension Implementation Report	Levels and the tables that implement them, Level Attributes and the Columns that implement them

Lineage and Impact Analysis Reports and Diagrams

Lineage and Impact Analysis Reports and Diagrams are available for Cubes, Dimensions, Materialized Views, Tables, Views, and Records.

Impact Analysis Reports Impact Analysis Reports list all items belonging to the subject of the report. The name of the mapping and the name of the item that it is mapped to is also displayed. The report provides a one-step impact analysis for all items related to the selected item.

For example, if you want a list of all the columns in a table used as sources in any mappings, use this report.

Lineage Reports Lineage Reports are similar to Impact Analysis Reports. They list items that are used as targets in a mapping.

Figure 17–28 shows a section from an Impact Analysis report.

Figure 17–28 Impact Analysis Report

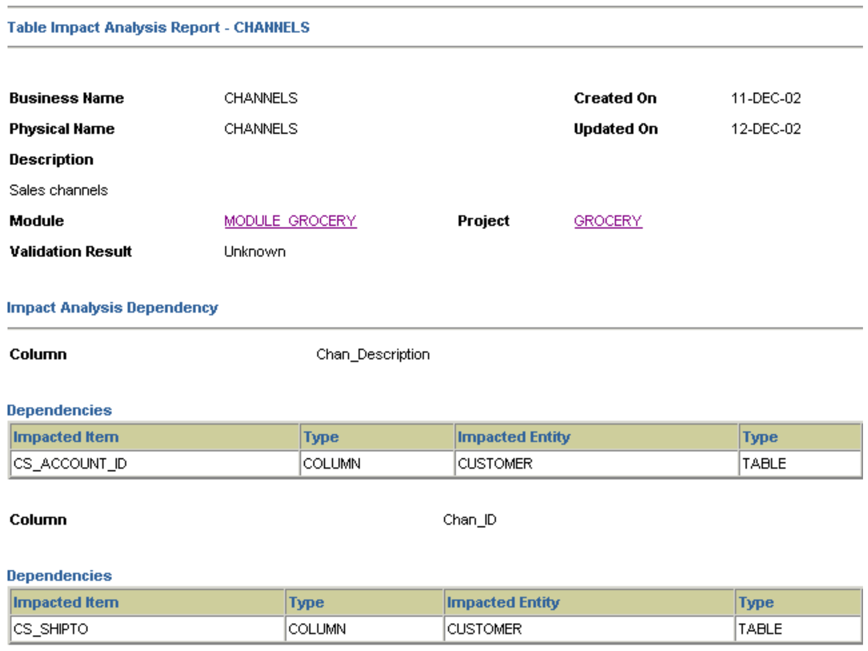


Table Impact Analysis Report - CHANNELS

Business Name	CHANNELS	**Created On**	11-DEC-02
Physical Name	CHANNELS	**Updated On**	12-DEC-02

Description

Sales channels

Module	MODULE_GROCERY	**Project**	GROCERY
Validation Result	Unknown		

Impact Analysis Dependency

Column Chan_Description

Dependencies

Impacted Item	Type	Impacted Entity	Type
CS_ACCOUNT_ID	COLUMN	CUSTOMER	TABLE

Column Chan_ID

Dependencies

Impacted Item	Type	Impacted Entity	Type
CS_SHIPTO	COLUMN	CUSTOMER	TABLE

Lineage and Impact Analysis Diagrams

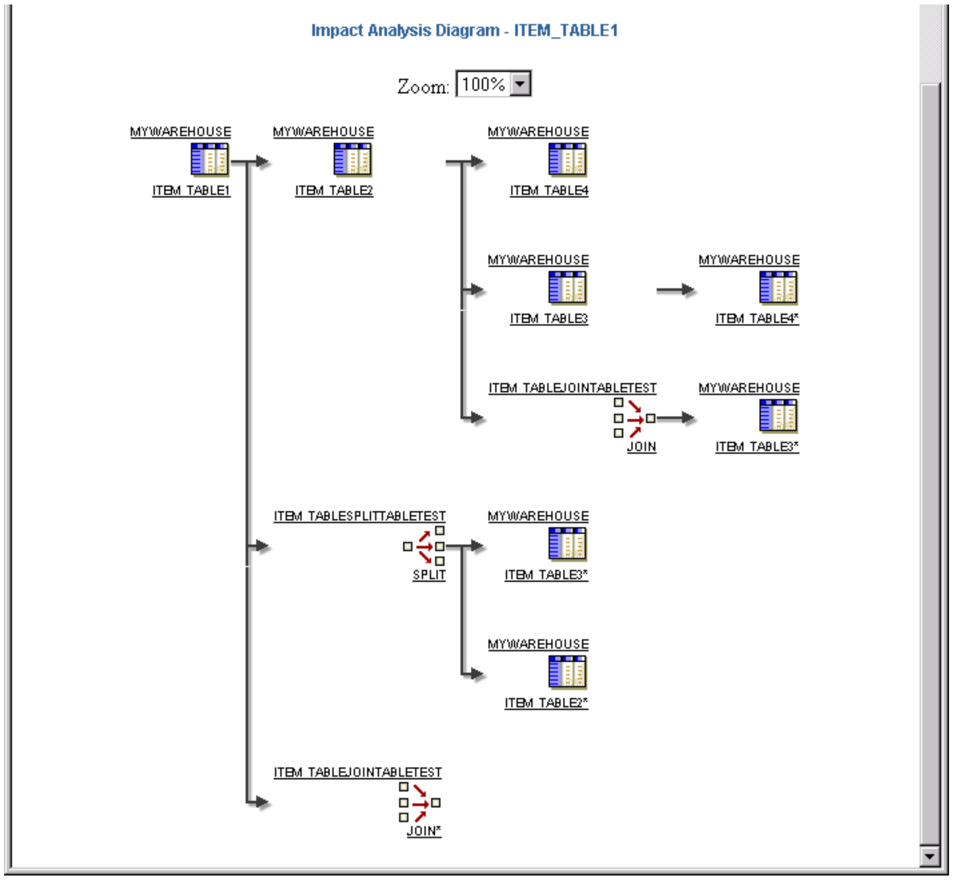
A Lineage Diagram graphically displays all the objects and transformations that are used to make up the subject of the Diagram. Lineage can be performed at either the object level or the item level. At the Object Level, the diagram can contain Tables, Views, Materialized Views, Dimensions, Cubes, Records, and Operators. At the item level the diagram can contain Columns, Measures, Fields, Operator Parameters, and Level Attributes.

The Lineage Diagram is displayed with the subject on the right side of the screen.

An Impact Analysis Diagram is identical except it shows all objects and transformations that might be affected by a change to the subject. The subject is displayed on the left side of the screen.

Figure 17–29 shows an example of an Impact Analysis Diagram.

Figure 17–29 Impact Analysis Diagram



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. You can also create custom reports that are based on Warehouse Builder Public Views. For more information, see "Creating Custom Reports" on page 18-21.

Note: To access the Warehouse Builder metadata reports, you must have access to an Oracle Portal site with a configured Warehouse Builder Browser portlet.

---

18 Managing Warehouse Builder Browser

Warehouse Builder Browser integrates with Oracle Portal to enable you to add and customize Warehouse Builder portlets. The Warehouse Builder Browser uses Warehouse Builder Public Views to create pre-built reports on all repository objects and relationships between objects. You can also use the Public Views to create your own custom reports.

This chapter contains the following information:

Warehouse Builder Browser Overview

Warehouse Builder Browser is a web based application that you can use to extend, access, and run reports on your repository metadata. To view the metadata and access these reports, you must have access to Oracle DatabaseAS Portal.

When you first install or upgrade Warehouse Builder, you can use the Warehouse Builder Browser Assistant available on the start menu to install the Warehouse Builder Browser. The Browser Assistant enables you to set up an Oracle DatabaseAS Portal from which you can access and create metadata reports. You can run the Browser Assistant during your installation process or defer it to a later time.

After Warehouse Builder Browser is installed, you can add the Warehouse Builder portlets to your Portal home page. You can run Metadata reports using the Browser from either the Warehouse Builder client or Oracle Portal.

About Oracle DatabaseAS Portal

Oracle DatabaseAS Portal enables you to create and view database objects using an HTML-based interface. It provides tools for creating HTML-based interfaces. Portal provides you with a centralized and personalized view of relevant applications and data in a single Web site.

The fundamental building blocks of an Oracle DatabaseAS Portal site are called portlets. A portlet is a re-usable information component that summarizes or provides access to an information source. Portlets can standalone in a Portal site, link to other portlets or portal sites, or they can be nested within one another.

Warehouse Builder Browser integrates with Oracle DatabaseAS Portal to obtain metadata reporting portlets.

Portlets for Managing Warehouse Builder Browser

The Warehouse Builder Browser consists of portlets that you can add to customize your portal page. You can add more than one of each portlet. For example, you can add two Reports portlets with each running a report on a different repository.

The portlets used for managing Warehouse Builder Browser include the following:

- Launcher Portlet
- Administration Portlet
- Reports Portlet

To access any portlet, you must first configure the Warehouse Builder Browser to access a Warehouse Builder Design Repository.

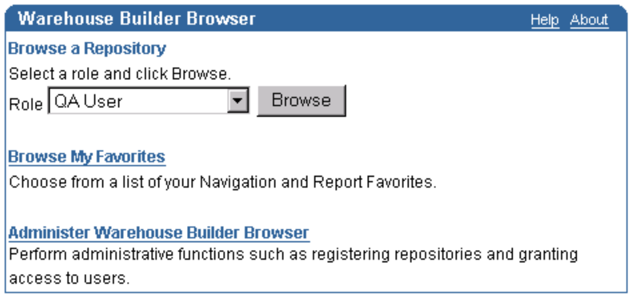
You can also add custom portlets to suit your business intelligence needs. For more information, see "Adding Portlets" on page 18-4.

Launcher Portlet

The Launcher Portlet provides access to all the available functions, as shown in Figure 18–1. Other portlets contain subsets of all available functions.

- **Browse a Repository:** Select a role and click **Browse** to browse the repositories available to the current user. The main Warehouse Builder Browser page displays in full page mode. You can select from a list of repositories, and use the Navigator to view the detail of that repository.
- **Browse My Favorites:** Select this link to view your Warehouse Builder Favorites in full page mode.
- **Administer Warehouse Builder Browser:** Select this link to view the Warehouse Builder Administration in full page mode. Use these pages to configure your Browser.

Figure 18–1 Launcher Portlet


Warehouse Builder Browser
Help About
Browse a Repository
Select a role and click Browse.
Role: QA User
Browse
Browse My Favorites
Choose from a list of your Navigation and Report Favorites.
Administer Warehouse Builder Browser
Perform administrative functions such as registering repositories and granting access to users.

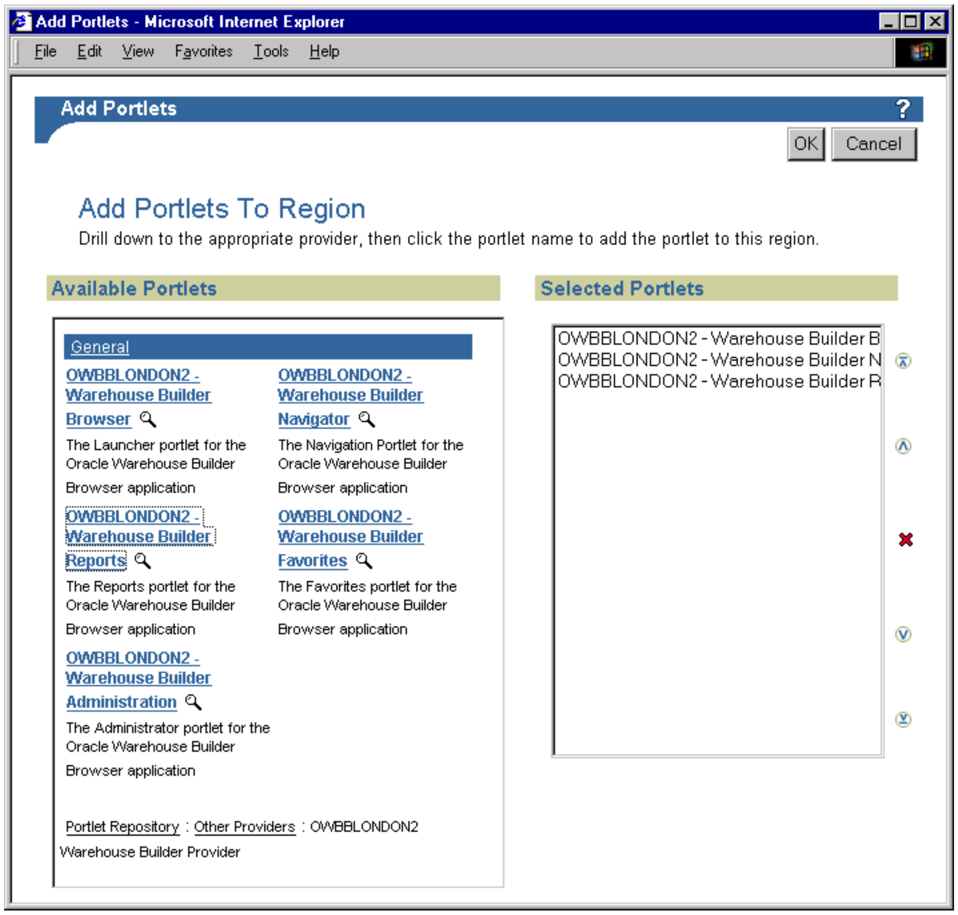
Administration Portlet

The Administration Portlet provides access to the following Administration features from your Oracle Portal home page, as shown in Figure 18–2:

- **Register an OWB Repository:** Register Warehouse Builder repositories and maintain database links.
- **Register a Custom Report:** Register custom reports.
- **Purge Stale User Information:** Purge obsolete Warehouse Builder Browser settings.

- **Resource Management:** Manage access privileges to Warehouse Builder Browser resources.
- **Manage Preferences:** Save and load preference settings from files or an existing schema.
- **Manage Dependency Index:** Improve performance of Impact Analysis by specifying how often to refresh the Dependency Index.

Figure 18–2 Administration Portlet


Add Portlets - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Add Portlets
OK
Cancel

Add Portlets To Region

Drill down to the appropriate provider, then click the portlet name to add the portlet to this region.

Available Portlets

General

OWBBLONDON2 - Warehouse Builder Browser

The Launcher portlet for the Oracle Warehouse Builder Browser application

OWBBLONDON2 - Warehouse Builder Navigator

The Navigation Portlet for the Oracle Warehouse Builder Browser application

OWBBLONDON2 - Warehouse Builder Reports

The Reports portlet for the Oracle Warehouse Builder Browser application

OWBBLONDON2 - Warehouse Builder Favorites

The Favorites portlet for the Oracle Warehouse Builder Browser application

OWBBLONDON2 - Warehouse Builder Administration

The Administrator portlet for the Oracle Warehouse Builder Browser application

Portlet Repository : Other Providers : OWBBLONDON2 Warehouse Builder Provider

Selected Portlets

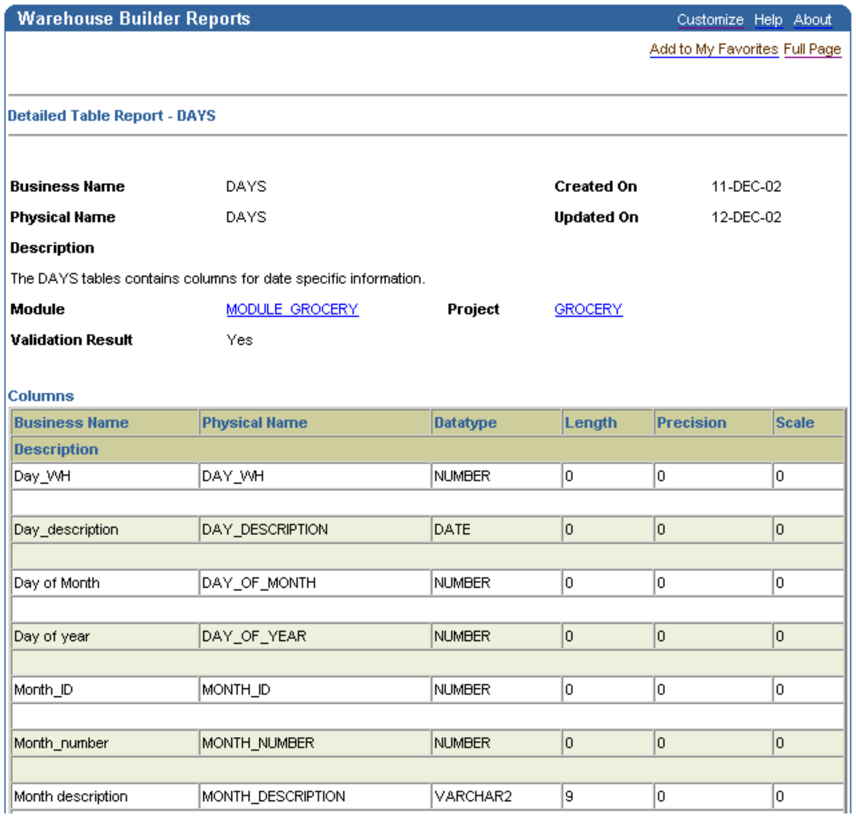
OWBBLONDON2 - Warehouse Builder B
OWBBLONDON2 - Warehouse Builder N
OWBBLONDON2 - Warehouse Builder R

Reports Portlet

The Reports Portlet displays a Warehouse Builder Report on your Oracle Portal home page, as shown in Figure 18–3. The Reports Portlet displays one of your favorite reports. When you first add this portlet to a page, it does not contain a default report. You must first use the customize option to select and add a report from you favorites list.

When you have a Reports portlet on your Oracle Portal home page, it refreshes each time you reload the home page. This can delay the home page display.

Figure 18–3 Reports Portlet



Detailed Table Report - DAYS

Business Name	DAYS	**Created On**	11-DEC-02
Physical Name	DAYS	**Updated On**	12-DEC-02

Description

The DAYS tables contains columns for date specific information.

Module	MODULE_GROCERY	**Project**	GROCERY
Validation Result	Yes		

Columns

Business Name	Physical Name	Datatype	Length	Precision	Scale
Day_WH	DAY_WH	NUMBER	0	0	0
Day_description	DAY_DESCRIPTION	DATE	0	0	0
Day of Month	DAY_OF_MONTH	NUMBER	0	0	0
Day of year	DAY_OF_YEAR	NUMBER	0	0	0
Month_ID	MONTH_ID	NUMBER	0	0	0
Month_number	MONTH_NUMBER	NUMBER	0	0	0
Month description	MONTH_DESCRIPTION	VARCHAR2	9	0	0

Adding Portlets

Warehouse Builder Browser portlets can be added to Oracle Portal after they have been installed on the machine running Oracle Portal.

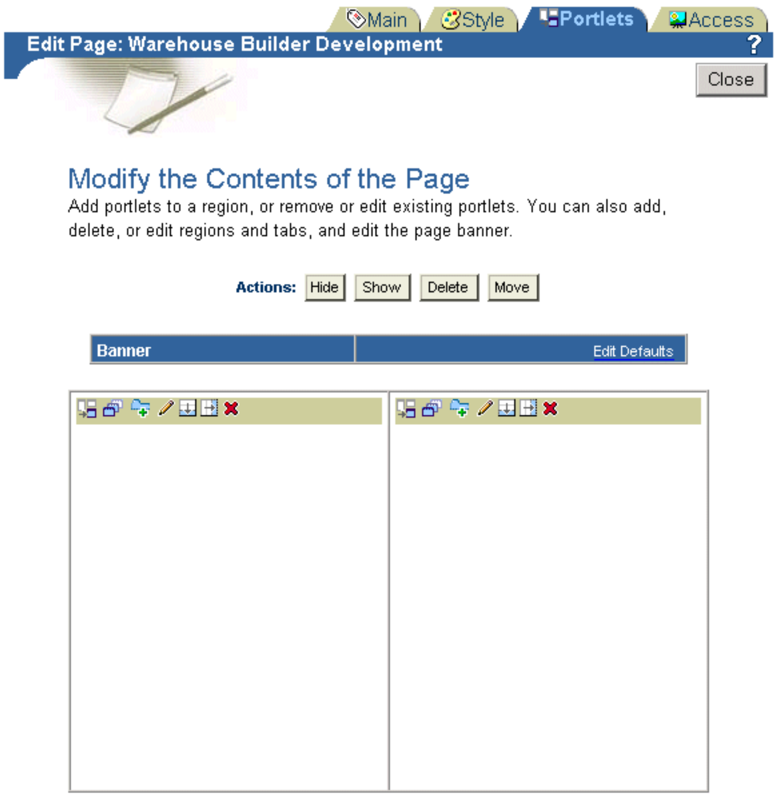
For information about customizing Oracle Portal pages, see the Oracle Portal documentation.

To add a portlet to a page:

1. From the Oracle Portal page, select the Edit Page link from the upper-right corner of the page.

   The Edit Page displays the contents of the Oracle Portal page, as shown in Figure 18–4.

Figure 18–4 Oracle Portal Edit Page


Main
Style
Portlets
Access
Edit Page: Warehouse Builder Development
?
Close

Modify the Contents of the Page

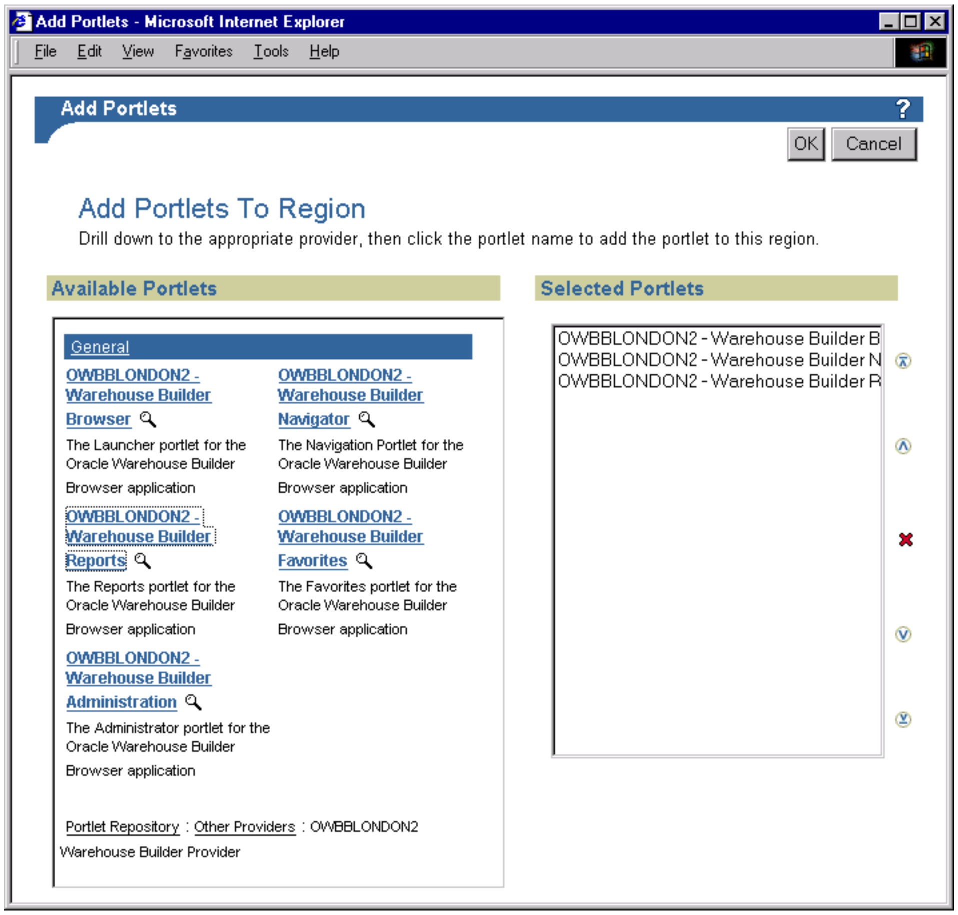
Add portlets to a region, or remove or edit existing portlets. You can also add, delete, or edit regions and tabs, and edit the page banner.

Actions: Hide Show Delete Move
Banner
Edit Defaults

2. Select the Add Portlets icon.

   The Add Portlets page displays a list of available portlets on the left side, and a list of selected portlets on the right side, as shown in Figure 18–5.

Figure 18–5 Add Portlets Page


Add Portlets - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Add Portlets
?
OK
Cancel

Add Portlets To Region

Drill down to the appropriate provider, then click the portlet name to add the portlet to this region.

Available Portlets

General

OWBBLONDON2 - Warehouse Builder Browser

The Launcher portlet for the Oracle Warehouse Builder Browser application

OWBBLONDON2 - Warehouse Builder Navigator

The Navigation Portlet for the Oracle Warehouse Builder Browser application

OWBBLONDON2 - Warehouse Builder Reports

The Reports portlet for the Oracle Warehouse Builder Browser application

OWBBLONDON2 - Warehouse Builder Favorites

The Favorites portlet for the Oracle Warehouse Builder Browser application

OWBBLONDON2 - Warehouse Builder Administration

The Administrator portlet for the Oracle Warehouse Builder Browser application

Portlet Repository : Other Providers : OWBBLONDON2 Warehouse Builder Provider

Selected Portlets

OWBBLONDON2 - Warehouse Builder B
OWBBLONDON2 - Warehouse Builder N
OWBBLONDON2 - Warehouse Builder R

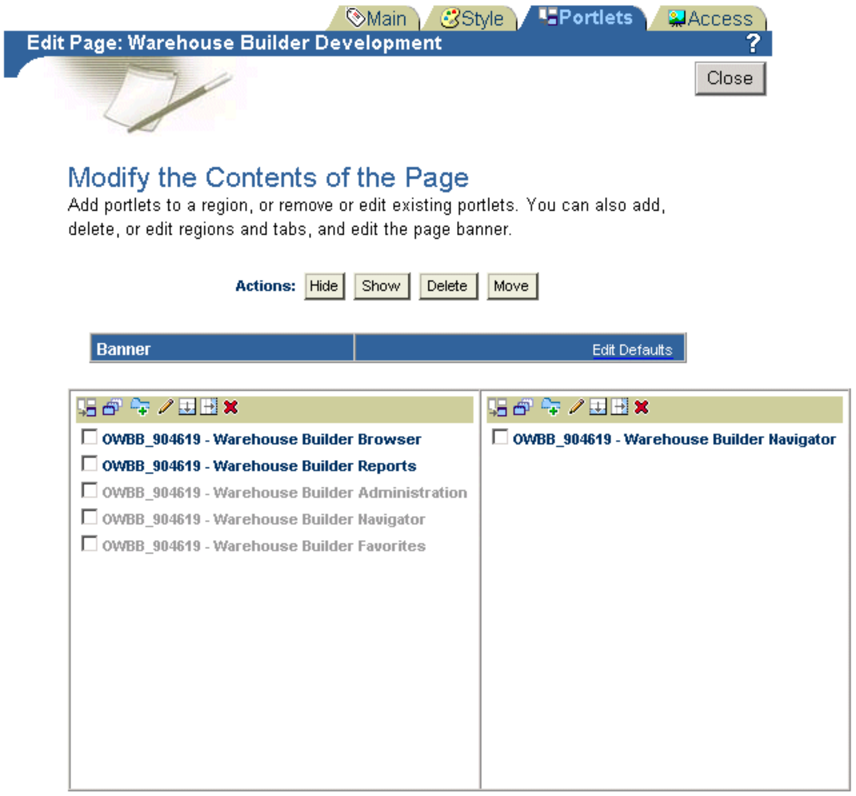
3. Select the portlets to add to the Oracle Portal home page.

The portlets you have added display in the right column. You can organize them by using the arrow buttons and delete them by using the X button.

4. Click **OK** when you are done.

The Edit Page displays the Warehouse Builder portlets that you added, as shown in Figure 18–6.

Figure 18–6 Edit Page



Edit Page: Warehouse Builder Development

Modify the Contents of the Page

Add portlets to a region, or remove or edit existing portlets. You can also add, delete, or edit regions and tabs, and edit the page banner.

Actions:

Banner

OWBB_904619 - Warehouse Builder Browser
OWBB_904619 - Warehouse Builder Reports
OWBB_904619 - Warehouse Builder Administration
OWBB_904619 - Warehouse Builder Navigator
OWBB_904619 - Warehouse Builder Favorites

OWBB_904619 - Warehouse Builder Navigator

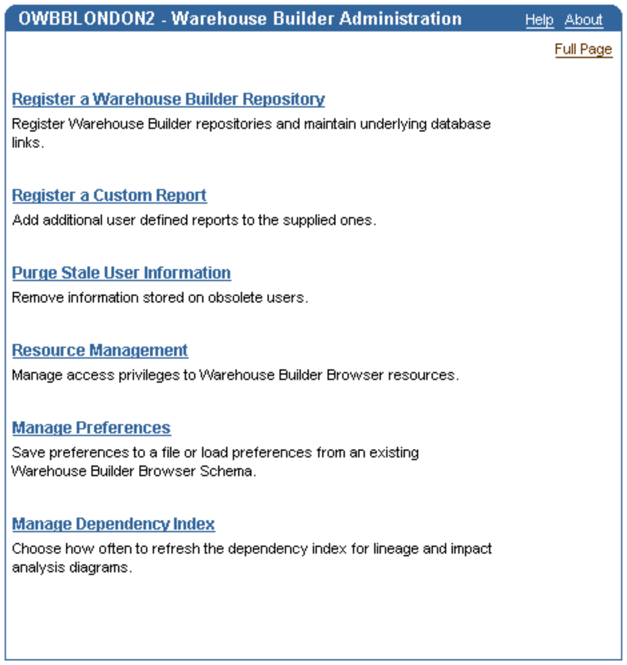
Administering Warehouse Builder Browser

The Warehouse Builder Administration pages can only be accessed by Oracle Portal users with full administrator privileges.

From the Oracle Portal Home page, select the **Administer Warehouse Builder Browser** link to access the Warehouse Builder Administration pages.

Figure 18–7 shows the Administration Portlet in Warehouse Builder Browser.

Figure 18–7 Administration Portlet in Warehouse Builder Browser


OWBBLONDON2 - Warehouse Builder Administration

Help About

Full Page

Register a Warehouse Builder Repository
Register Warehouse Builder repositories and maintain underlying database links.

Register a Custom Report
Add additional user defined reports to the supplied ones.

Purge Stale User Information
Remove information stored on obsolete users.

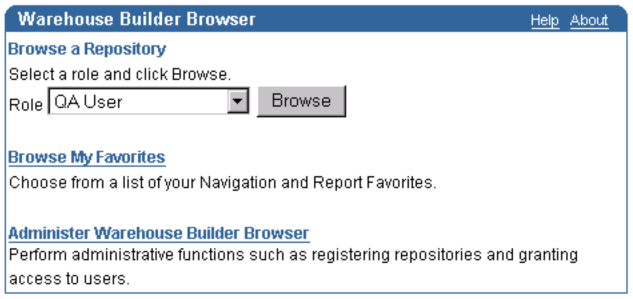
Resource Management
Manage access privileges to Warehouse Builder Browser resources.

Manage Preferences
Save preferences to a file or load preferences from an existing Warehouse Builder Browser Schema.

Manage Dependency Index
Choose how often to refresh the dependency index for lineage and impact analysis diagrams.

Figure 18–8 shows the launcher portlet in Warehouse Builder Browser.

Figure 18–8 Launcher Portlet in Warehouse Builder Browser


Warehouse Builder Browser

Help About

Browse a Repository
Select a role and click Browse.
Role QA User Browse

Browse My Favorites
Choose from a list of your Navigation and Report Favorites.

Administer Warehouse Builder Browser
Perform administrative functions such as registering repositories and granting access to users.

The Administration page contains links for the following administration actions:

- Register a Warehouse Builder Repository
- Register a Custom Report
- Resource Management
- Managing Preferences
- Managing the Dependency Index

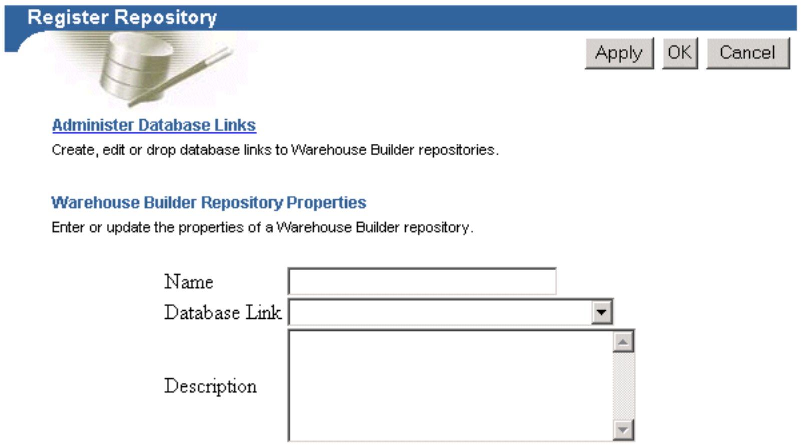
Register a Warehouse Builder Repository

Before you can access repository metadata reports, register the Warehouse Builder repository with the Warehouse Builder Browser.

To register a Warehouse Builder Design Repository:

1. Click **Register an OWB Repository** on the Warehouse Builder Administration home page. The Register Repository page displays as shown in Figure 18–9.

Figure 18–9 Register Repository Page


Register Repository

Apply OK Cancel

Administer Database Links

Create, edit or drop database links to Warehouse Builder repositories.

Warehouse Builder Repository Properties

Enter or update the properties of a Warehouse Builder repository.

Name	
Database Link	
Description	

2. Specify the Warehouse Builder repository properties. Table 18–1 lists the properties.

Table 18–1 Warehouse Builder Repository Properties

Field	Description
Name	The user-defined name to identify the repository in the browser system. This name is displayed in the navigation pages.
Database Link	The name of the database link used to access the repository. The link must already be created using the Administer Database Links page. This field must be specified even if the repository is in the same database as the browser system.
Description	The user-defined descriptive text. This appears in the navigation pages for the repository.

3. Click **Apply** to register the repository.
4. Click **OK.**

   The repository displays in the Warehouse Builder Administration home page.

Managing Repositories

The Resource Management page lists all registered repositories. The actions listed next to the repository are described in Table 18–2.

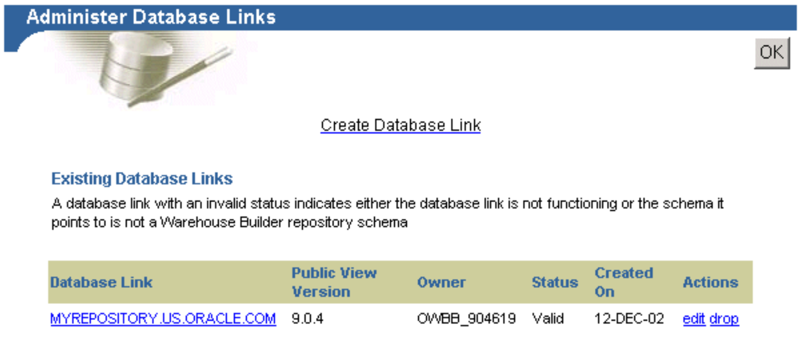
Table 18–2 Repository Management

Actions	Description
Access	Use this to grant or revoke repository access privileges to the users.
Edit	Use this to edit repository properties.
Unregister	Use this to unregister the repository. After unregistering the repository, it can no longer be browsed using the Browser system. You must re-register the repository if you want to browse it again.

Creating Database Links

Use the Administer Database Links page, as shown in Figure 18–10, to connect to Warehouse Builder repositories from the Browser.

Figure 18–10 Administer Database Links Page



Existing Database Links

A database link with an invalid status indicates either the database link is not functioning or the schema it points to is not a Warehouse Builder repository schema

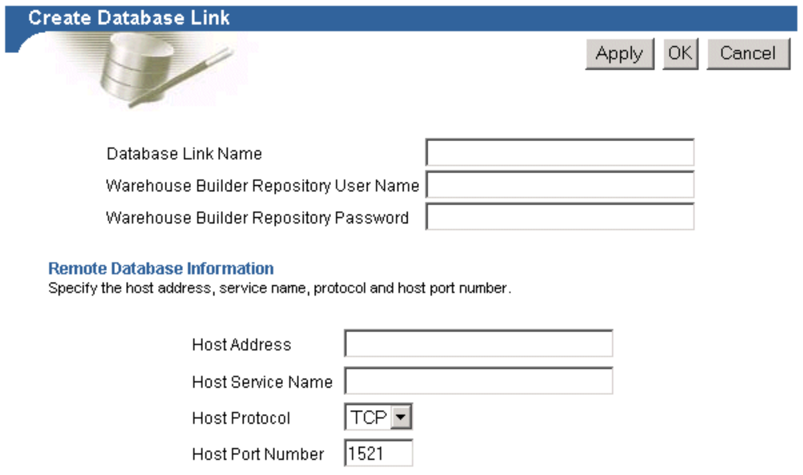
Database Link	Public View Version	Owner	Status	Created On	Actions
MYREPOSITORY.US.ORACLE.COM	9.0.4	OWBB_904619	Valid	12-DEC-02	edit drop

To create a database link:

1. Click **Register an OWB Repository** on the Warehouse Builder Administration home page. The Register Repository page displays.
2. Click the link for Administer Database Links.
3. Select **Create Database Link** from the Administer Database Links page.

   Figure 18–11 shows the Create Database Links page.

Figure 18–11 Create Database Links Page



Database Link Name	
Warehouse Builder Repository User Name	
Warehouse Builder Repository Password	

Remote Database Information

Specify the host address, service name, protocol and host port number.

Host Address	
Host Service Name	
Host Protocol	TCP
Host Port Number	1521

4. Specify the database link name.
5. Specify the Warehouse Builder Design repository user name and password.
6. Specify the remote database information.

   Provide the host address, service name, protocol, and host port number.
7. Click **Apply** to connect the link.
8. Click **OK.**

   The new link displays on the Administer Database Links page.

Viewing a Database Link

To view a database link:

1. Select the name of the database link from the Administer Database Links page.

   The View Database Link page displays with a detailed report on the database link you selected.

2. Click **OK.**

   The browser returns to the Administer Database Links page.

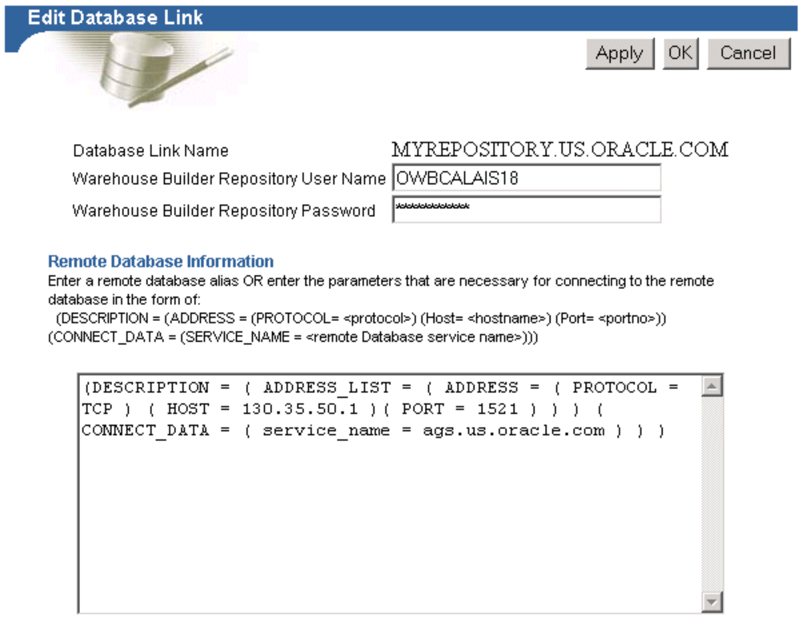
Editing a Database Link

To edit a database link:

1. From the Administer Database Links page, select **edit** for the database link you want to alter. The edit link is located under the Actions column.

   The Edit Database Link page displays as shown in Figure 18–12.

Figure 18–12 Edit Database Link Page



Database Link Name	MYREPOSITORY.US.ORACLE.COM
Warehouse Builder Repository User Name	OWBCALAIS18
Warehouse Builder Repository Password	**********

Remote Database Information

Enter a remote database alias OR enter the parameters that are necessary for connecting to the remote database in the form of:
(DESCRIPTION = (ADDRESS = (PROTOCOL= <protocol>) (Host= <hostname>) (Port= <portno>)) (CONNECT_DATA = (SERVICE_NAME = <remote Database service name>)))

```
(DESCRIPTION = ( ADDRESS_LIST = ( ADDRESS = ( PROTOCOL =
TCP ) ( HOST = 130.35.50.1 )( PORT = 1521 ) ) ) (
CONNECT_DATA = ( service_name = ags.us.oracle.com ) ) )
```

2. Edit the database link and click **Apply.**
3. Click **OK.**

Dropping a Database Link

Dropping a database link deletes it permanently. You must create a new link to use it again.

To drop a database link:

1. If the database link has been used to register Warehouse Builder repositories, unregister the Warehouse Builder repositories.
2. From the Administer Database Links page, select **drop** for the database link you want to drop. The drop link is under the Actions column.

   The database link is dropped and the browser returns to the Administer Database Links page.

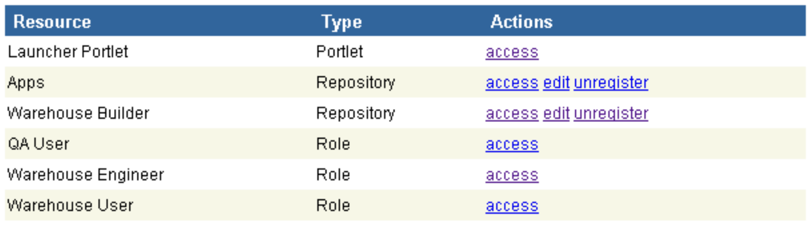
Unregistering a Repository

To unregister a Repository:

1. Select the Administer Warehouse Builder Browser link from the Browser page.
2. Select **Manage Resources.**

   The Warehouse Builder Administration page displays as shown in Figure 18–13. The table at the bottom of the page lists the registered repositories listed.

Figure 18–13 Registered Repositories and Roles



Resource	Type	Actions
Launcher Portlet	Portlet	access
Apps	Repository	access edit unregister
Warehouse Builder	Repository	access edit unregister
QA User	Role	access
Warehouse Engineer	Role	access
Warehouse User	Role	access

3. Select the repository to unregister and click the **unregister** link.

   The repository is unregistered and no longer appears in the list of registered repositories. You can no longer browse it using the Browser.

Register a Custom Report

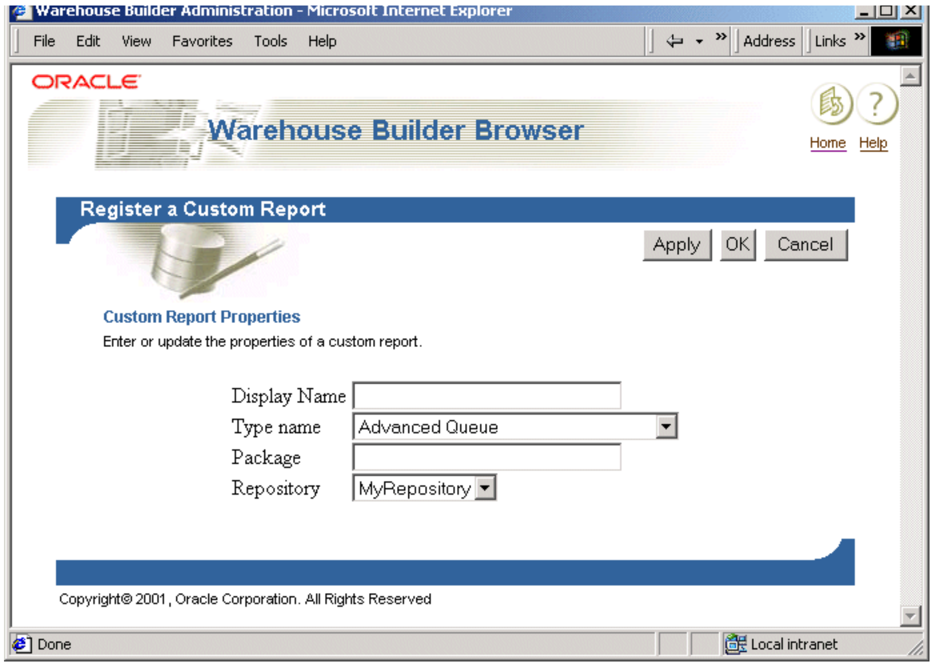
A custom report is an application component created in a tool such as the Oracle Portal facilities. Registration of the report provides the browser system with the information required to invoke the report. For more information, see "Creating Custom Reports" on page 18-21.

To register a custom report:

1. From the launcher portlet, click the **Administer Warehouse Builder Browser** link, and then select the **Register a Custom Report** link.

   The Register a Custom Report page displays as shown in Figure 18–14.

Figure 18–14 Register Custom Report Page


Warehouse Builder Administration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address Links
ORACLE
Warehouse Builder Browser
Home Help
Register a Custom Report
Apply OK Cancel
Done
Local intranet

Custom Report Properties

Enter or update the properties of a custom report.

Display Name	
Type name	Advanced Queue
Package	
Repository	MyRepository

Copyright© 2001, Oracle Corporation. All Rights Reserved

2. Enter a display name for the report.
3. Select the type and repository from the drop-down lists. Table 18–3 lists the custom report properties.

Table 18–3 Custom Report Properties

Field	Description
Display Name	Name of the report. This name is displayed on the Reports List page.
Type Name	Name of the data type reported on by this report.
Package	The full name of the PL/SQL package which implements this report.
Repository	Name of the repository containing target objects for this report.

4. Enter the qualified package name for the report in the format `<schema>.<package>`. The package name is displayed on the Develop page for the report.
5. Click **Apply** or **OK** to complete the registration.

   The report appears in the resource list of the administration page.

Adding a Custom Report to a Role

To add a custom report to a Warehouse Builder Browser role:

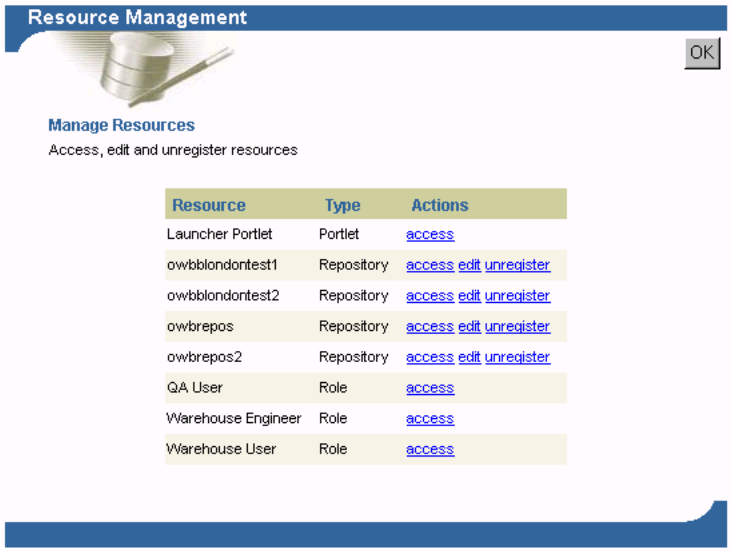
1. Click the **roles** action link from the resource list entry for the custom report.
2. Click the **add** action link for each role that you want to access the report.

Resource Management

Resource Management enables you to access rights to many aspects of the Warehouse Builder Browser as well as editing, registering, and unregistering repositories and reports.

The Resource Management page contains a table listing all resources that have been registered in the Warehouse Builder Browser, as shown in Figure 18–15. The following sections describe how to modify these resources.

Figure 18–15 Resource Management



Manage Resources

Access, edit and unregister resources

Resource	Type	Actions
Launcher Portlet	Portlet	access
owbblondontest1	Repository	access edit unregister
owbblondontest2	Repository	access edit unregister
owbrepos	Repository	access edit unregister
owbrepos2	Repository	access edit unregister
QA User	Role	access
Warehouse Engineer	Role	access
Warehouse User	Role	access

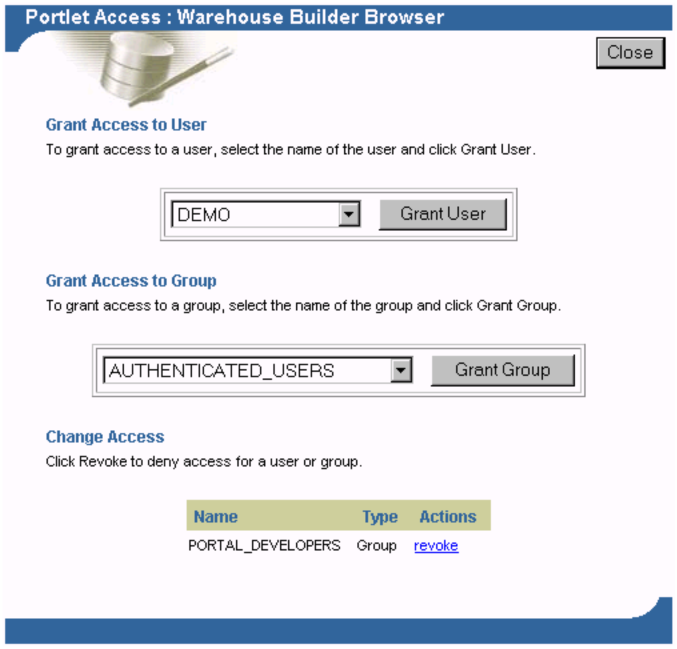
Adding User Accounts

To add a user account for Warehouse Builder Browser:

1. From the Resource Management page, select the **Access** action listed next to the resource entitled Launcher Portlet, as listed first in Figure 18–15.

   Warehouse Builder Browser displays the Portlet Access page as shown in Figure 18–16.

Figure 18–16 Portlet Access


Portlet Access : Warehouse Builder Browser

Close

Grant Access to User

To grant access to a user, select the name of the user and click Grant User.

DEMO | Grant User

Grant Access to Group

To grant access to a group, select the name of the group and click Grant Group.

AUTHENTICATED_USERS | Grant Group

Change Access

Click Revoke to deny access for a user or group.

Name	Type	Actions
PORTAL_DEVELOPERS	Group	revoke

From the Portlet Access page, you can perform the following tasks:

- Grant an access right to a Single Sign On user by selecting the name of the user from the drop-down list and clicking **Grant User.**
- Grant an access right to an Oracle Portal group by selecting the name of the group from the drop-down list and clicking **Grant Group.**
- Revoke an access right by clicking **revoke** in the appropriate table row.

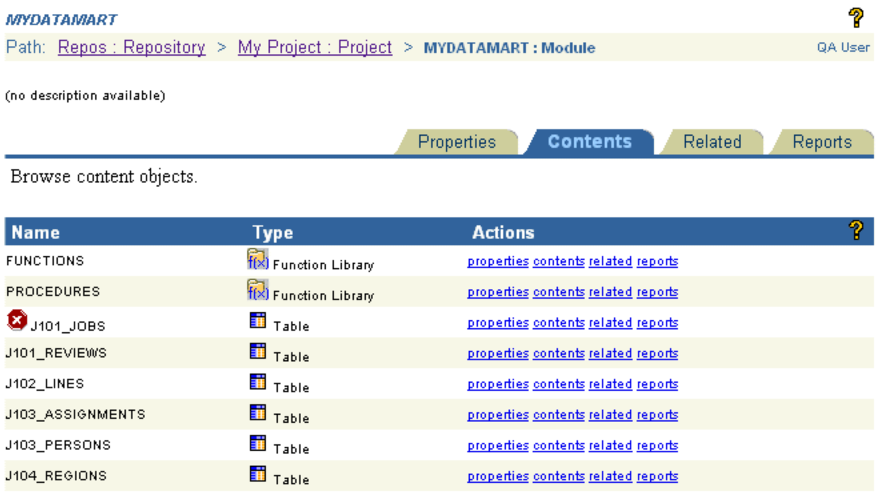
To use the Warehouse Builder Browser, your single sign on user or a group associated with that user must have rights to one of these Warehouse Builder pre-defined roles:

- **Warehouse Developer:** A user who uses Warehouse Builder to create the warehouse.
- **QA User:** A user who tests the quality of the warehouse before it is deployed.
- **Warehouse User:** A user who uses the deployed warehouse to understand the underlying metadata.

The Administrator can assign these roles to users and groups. All of the pre-defined Reports and Navigation pages are available to all roles. When you add custom reports, you can assign them to different roles.

For the QA User role, objects that fail validation display an error icon in the Contents tab of the Navigation page, as shown in Figure 18–17. This error icon is not displayed in reports for the other roles.

Figure 18–17 Validation Error


MYDATAMART

Path: Repos : Repository > My Project : Project > MYDATAMART : Module QA User

(no description available)

Properties | Contents | Related | Reports

Browse content objects.

Name	Type	Actions
FUNCTIONS	Function Library	properties contents related reports
PROCEDURES	Function Library	properties contents related reports
J101_JOBS	Table	properties contents related reports
J101_REVIEWS	Table	properties contents related reports
J102_LINES	Table	properties contents related reports
J103_ASSIGNMENTS	Table	properties contents related reports
J103_PERSONS	Table	properties contents related reports
J104_REGIONS	Table	properties contents related reports

Managing Custom Reports

The Resource Management page lists all registered custom reports. The actions listed next to the repository are described in Table 18–4.

Table 18–4 Custom Report Management

Actions	Description
Role	Use this to assign a report to one or more roles. When the report is assigned to a role, it appears in the appropriate report list for that role.
Edit	Use this to edit custom report properties.
Unregister	Use this to unregister the report. After unregistering the report, it can no longer be browsed using the browser system.

Assigning a custom report to a role adds the report to the appropriate report list page for that role. The name and subject type of the report are indicated at the top left corner of the page. A list of the available roles is provided in the table at the bottom of the page.

Managing Preferences

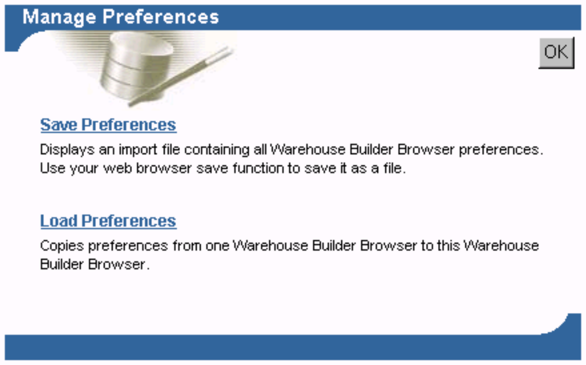
Use the Manage Preferences page to save and load Warehouse Builder Browser preferences. This lets you retain your preferences when you upgrade to a new version of Warehouse Builder Browser. You can copy schema preferences across schemas.

The preferences you can save include:

- Favorites.
- Registered custom reports.
- Registered Warehouse Builder Repositories.
- Access rights associated with roles, repositories, custom reports, and the Launcher Portlet.
- External Links.

Figure 18–18 shows the Manage Preferences page.

Figure 18–18 Manage Preferences Page


Manage Preferences

OK

Save Preferences

Displays an import file containing all Warehouse Builder Browser preferences. Use your web browser save function to save it as a file.

Load Preferences

Copies preferences from one Warehouse Builder Browser to this Warehouse Builder Browser.

Saving Preferences

To save preferences to a file:

1. Select **Save Preferences** from the Manage Preferences page.

   The preferences display in a separate window in text format

2. From the browser menu bar, select **File,** and then **Save As** to save the file.

   This file can be loaded into Warehouse Builder Browser using a tool such as SQL* Plus.

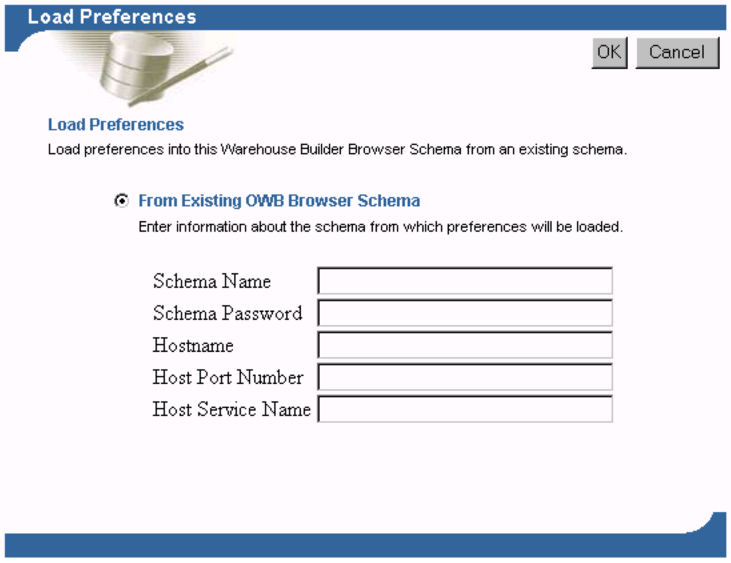
Loading Preferences

To load preferences from an existing schema:

1. Select **Load Preferences** from the Manage Preferences page.

   Figure 18–19 shows the Load Preferences page.

Figure 18–19 Load Preferences Page


Load Preferences

OK Cancel

Load Preferences

Load preferences into this Warehouse Builder Browser Schema from an existing schema.

(●) From Existing OWB Browser Schema

Enter information about the schema from which preferences will be loaded.

Schema Name	
Schema Password	
Hostname	
Host Port Number	
Host Service Name	

2. Specify the following information from an existing Warehouse Builder Browser schema: Schema Name, Schema Password, Hostname, Host Port Number, Host Service Name.

3. Click **OK** to load the preferences into the current Warehouse Builder schema.

   A status page displays the preferences that were loaded and any errors that occurred. All errors must be resolved to load the preferences. Errors due to missing database links provide links to the Create Database Links page.

   Database links are not automatically created. If the preferences you are loading contain references to repositories, the database links to those repositories must be created before the load can be successful.

Managing the Dependency Index

Use the Manage Dependency Index page to specify the refresh frequency options for the dependency index for each repository. The dependency index is used to increase performance when running lineage and impact analysis diagrams. You can refresh the dependency index at any time from the Repository page of the Warehouse Builder Navigator.

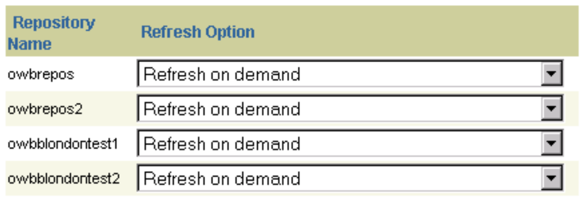
Setting the Refresh Options

To specify the dependency refresh option:

1. Open the Warehouse Builder Browser, and select **Manage Dependency Index** from the Administration page or portlet.

   The Manage Dependency Index page displays the available repositories and refresh options, as shown in Figure 18–20.

Figure 18–20 Setting Refresh Options



Repository Name	Refresh Option
owbrepos	Refresh on demand
owbrepos2	Refresh on demand
owbblondontest1	Refresh on demand
owbblondontest2	Refresh on demand

2. Choose one of the options from the drop-down list and click **OK.** Table 18–5 describes each option.

Table 18–5 Dependency Index Options

Option	Description
Refresh on demand	You must activate the refresh dependency index link to refresh the index. This link is located on Navigator page that lists all accessible repositories. The dependency index is only refreshed when this action link is activated. This is the best option when using a repository that changes infrequently.
Refresh on first diagram request of the session	Refreshes the dependency index when the first Lineage or Impact Analysis diagram for a repository is run during a session. This is the best option if you want current information, but are not concerned with repository updates that occur during the session.

Table 18–5 (Cont.) Dependency Index Options

Option	Description
Refresh on every diagram request	Refreshes the dependency index every time a Lineage or Impact Analysis diagram is requested. This is the best option if you want to display your diagrams and reports with the latest information in the repository.

Refreshing the Dependency Index on Demand

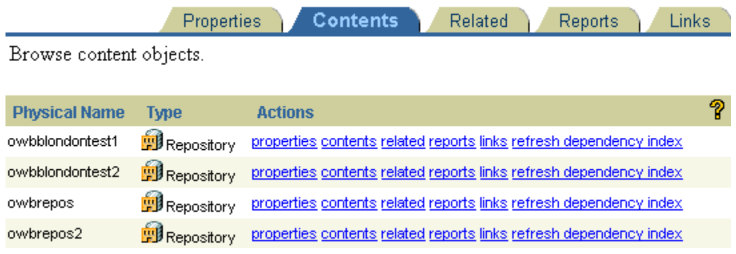
You can refresh the dependency index at any time. If you run a Lineage or Impact Analysis diagram that has never been refreshed, an automatic refresh occurs prior to displaying the diagram.

To refresh the dependency index:

1. Open the Warehouse Builder Browser from the Launcher portlet.

   The Contents tab lists the available repositories, as shown in Figure 18–21.

Figure 18–21 Refreshing the Dependency Index



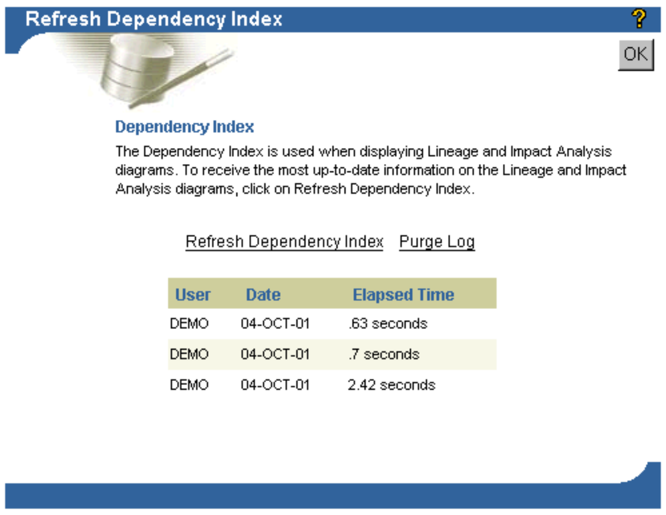
Browse content objects.

Physical Name	Type	Actions
owbblondontest1	Repository	properties contents related reports links refresh dependency index
owbblondontest2	Repository	properties contents related reports links refresh dependency index
owbrepos	Repository	properties contents related reports links refresh dependency index
owbrepos2	Repository	properties contents related reports links refresh dependency index

2. Select **refresh dependency index.**

   The Refresh Dependency Index page displays with a log of previous refreshes at the bottom of the page, as shown in Figure 18–22. The elapsed time helps you determine how long the operation will take.

Figure 18–22 Dependency Index Refresh Log



Dependency Index

The Dependency Index is used when displaying Lineage and Impact Analysis diagrams. To receive the most up-to-date information on the Lineage and Impact Analysis diagrams, click on Refresh Dependency Index.

Refresh Dependency Index   Purge Log

User	Date	Elapsed Time
DEMO	04-OCT-01	.63 seconds
DEMO	04-OCT-01	.7 seconds
DEMO	04-OCT-01	2.42 seconds

3. Select **Refresh Dependency Index** to refresh the dependency index based on the latest data in the repository.

After the refresh is complete, the log displays the user name, date, and elapsed time of the refresh. You can purge the log by selecting **Purge Log.** This purges the log of the refreshes except for the last refresh.

Other Administration Tasks

To refresh the Portlet Repository:

1. From the Oracle Portal Home Page, click the **Administer** tab.
2. Scroll down to locate the Portlet named Portlet Repository.
3. Click **Refresh Portlet Repository.**
4. Click **Refresh.**

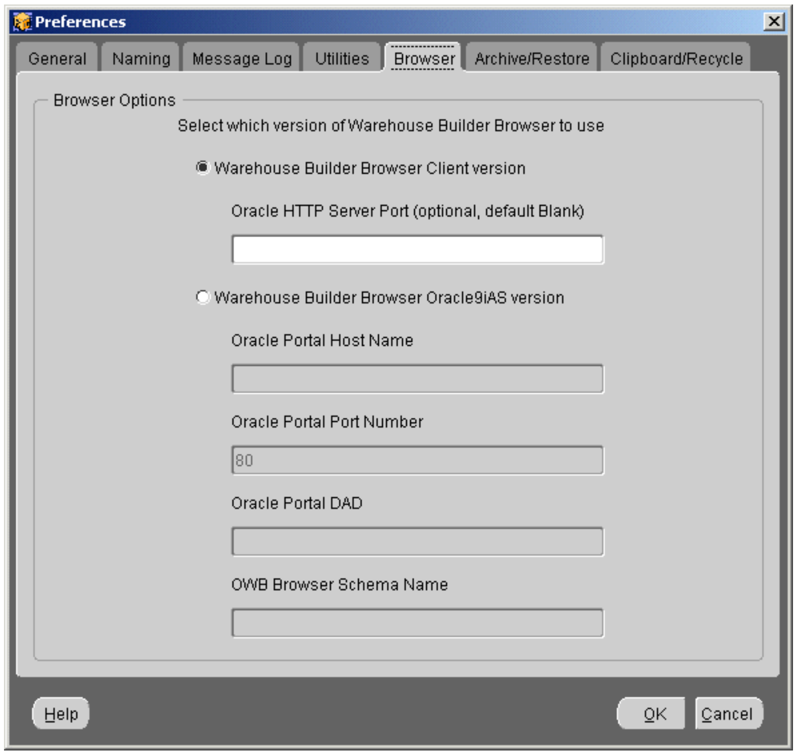
Configuring the Warehouse Builder Client

The Warehouse Builder Console includes a Preferences dialog containing tabs that you use to configure the Warehouse Builder environment. Use the Browser tab to set the network and IP connection information for Oracle Portal. This enables you to view metadata reports using the Warehouse Builder Browser.

To access the preferences dialog:

1. From the **Project** menu, click **Preferences.**
2. From the **Preferences** dialog, select the **Browser** tab as shown in Figure 18–23.

Figure 18–23 Browser Tab



Browser Options

Select which version of Warehouse Builder Browser to use

[✓ Warehouse Builder Browser Client version]

Oracle HTTP Server Port (optional, default Blank)

Warehouse Builder Browser Oracle9iAS version

Oracle Portal Host Name

Oracle Portal Port Number: 80

Oracle Portal DAD

OWB Browser Schema Name

3. Specify the following information:

   Oracle Portal Host Name

Oracle Portal Port Number

Oracle Portal DAD

Warehouse Builder Browser Schema Name

4. Click **OK.**

Creating Custom Reports

You can create custom reports on your metadata using the Warehouse Builder Public Views and Standard Query Language (SQL). These views provide access to your metadata repository tables and report on your data definitions, transformations, and deployment areas. You can use the Warehouse Builder Browser or another reporting tool to view the public views.

For a complete list of available Public Views and the objects they contain, see Appendix D, "Warehouse Builder Public Views".

Creating a Custom Report in Oracle Portal

To create a custom report using Oracle Portal:

1. Log on to Oracle Portal and select the Database Objects tab from the Navigator page.
2. Select the Applications tab, and click the **Create New Application** link.
3. Create a new application and click **OK.**
4. Select the application you just created and select the **Create New Report** link.
5. Select the **Report from SQL Query** link from the page that displays next.
6. Type the report name and display name and click **Next.**
7. Type the SQL query to define the report and click **Finish.**

Click **Next** to continue to pages where you can customize the appearance of the report. To customize the report at a later time, select the Edit action.

Your SQL queries must reference a database link to the Warehouse Builder repository. You can use the default_owb_link created during the Warehouse Builder Browser installation. The SQL query for a report can only call PUBLIC database link or links within the application schema where a report resides.

Although a reports can reside in a schema other than Warehouse Builder Browser schema, the report must be executable by the Warehouse Builder Browser schema. To grant execution privilege for a portal report, go to **Oracle Portal Home Page > Database Objects > Database Schemas > Report Schema > Report Package > Grant Access.**

The following query provides a simple project report that lists the information systems it contains:

```
select * from all_iv_information_systems@default_owb_link where project_id = :id
```

When run from the Warehouse Builder Browser Navigation pages, the marker `:id` is automatically substituted with the appropriate value.

Verify that the report can be run in the following environments:

- **SQL*Plus:** Log on as the user who owns the report. The owner is displayed on the Develop page for the report. In SQL* Plus, replace the marker `:id` with a valid project_id.
- **Oracle Portal:** From the Develop page, select the **Customize** link, enter a valid project_id in the edit box labelled Id, and click **Run Report.**

Part V

Integrating and Extending Warehouse Builder

This part contains the following chapters:

19

Extending Warehouse Builder Functionality

This chapter describes how to extend current Warehouse Builder functionality. This chapter includes the following topics:

About Oracle Metabase (OMB) Plus

This chapter discusses tasks you can perform to extend Warehouse Builder functionality using Oracle Metabase (OMB) Plus.

OMB Plus is the flexible, high-level command line metadata access tool for Oracle9i 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. You can access the Warehouse Builder metadata repository and the runtime repository. Using OMB Plus, you can navigate repositories and manage and manipulate metadata in repositories.

The remainder of this chapter discusses how to perform specialized tasks in Warehouse Builder using the OMB scripting language. For syntax information on specific OMB Plus commands, see Oracle Warehouse Builder Scripting Reference .

User-Defined Properties

Warehouse Builder enables you to extend its design repository through User Defined Properties (UDP). Each repository object has a pre-defined property set. You can add custom properties to an object by creating a UDP.

You create and manage UDPs using the Oracle MetaBase (OMB) Scripting Language. You can view UDPs using OMB or the Warehouse Builder client. In the client, UDPs display on property sheets and in the Warehouse Builder Browser.

UDPs behave like native properties and follow Warehouse Builder rules for object locking, multiuser access, transactions, and security. When you take metadata snapshots of the object, Warehouse Builder captures the associated UDPs. You can also import and export UDPs using the Metadata Loader (MDL).

Managing User Defined Properties

As the Warehouse Builder administrator, you should define all user-defined properties into the Warehouse Builder repository before allowing end users to access it. In doing so, you avoid the task of supplying values for UDPs on existing objects.

You should register all user-defined properties centrally in the design repository, and not locally on the client. To create or edit a UDP, you must be the single user accessing to the Warehouse Builder repository.

You can use the following OMB Plus commands for creating and manipulating user defined properties:

- OMBDEFINE
- OMBDESCRIBE

When you create and commit a UDP, OMB performs the following validations:

- A namespace check ensures that you did not define two identically named properties within the same class hierarchy. Prefix the user defined property with 'UDP_' to avoid conflicts with the future names introduced by Warehouse Builder.
- A property value check ensures that you defined default values consistent with the data types you specified.
- A user access check ensures that you have single-user access to the entire repository.

OMBDEFINE

The OMBREDEFINE CLASS_DEFINITION enables you to manipulate UDPs. To create a UDP on the Dimension object, issue the following statement. This adds a UDP definition to class definition 'DIMENSION':

```
OMBREDEFINE CLASS_DEFINITION 'DIMENSION_TABLE'
     ADD PROPERTY_DEFINITION 'UDP_Dim' SET PROPERTIES (TYPE, DEFAULT_VALUE)
     VALUES ('INTEGER', '100')
```

The following command adds a property to the 'COLUMN' type. This property displays in the Table, View, Materialized View, External Table and Sequence Property Sheets:

```
OMBREDEFINE CLASS_DEFINITION 'COLUMN'
     ADD PROPERTY_DEFINITION 'UDP_Col' SET PROPERTIES (TYPE, DEFAULT_VALUE)
     VALUES ('STRING', 'foo')
```

The following command enables you to change the name or the default value of a given property.

```
OMBREDEFINE CLASS_DEFINITION 'TABLE' MODIFY PROPERTY_DEFINITION 'UDP_Tbl'
     SET PROPERTIES (DEFAULT_VALUE, BUSINESS_NAME)
     VALUES ('99', 'UDP_Tbl')
```

The following command deletes the tbl_udp property from the 'Table' class. This is a very destructive and highly deprecated action since it cannot be undone. It renders all property value customizations made for this property definition in your repository irretrievable:

```
OMBREDEFINE CLASS_DEFINITION 'TABLE' DELETE PROPERTY_DEFINITION 'UDP_Tbl'
```

OMBDESCRIBE

You can use OMBDESCRIBE on a Class Definition to view the attributes for a metadata element. OMBDESCRIBE enables you to list the user defined properties you for a given object type. For instance, the following command lists the user defined properties for a dimension:

```
OMBDESCRIBE CLASS_DEFINITION 'DIMENSION_TABLE' GET PROPERTY_DEFINITIONS
```

You can also use OMBDESCRIBE to introspect the properties of a Property Definition. For instance, for a user defined property called `UDP_Dim` under the Dimension Class Definition, you can learn the datatype, default value and business name with the following command:

```
OMBDESCRIBE CLASS_DEFINITION 'DIMENSION_TABLE' PROPERTY_DEFINITION  'UDP_Dim'
     GET PROPERTIES (TYPE, DEFAULT_VALUE, BUSINESS_NAME)
```

You can specify the data type of a user-defined property such as char, number and date.

Viewing User Defined Properties

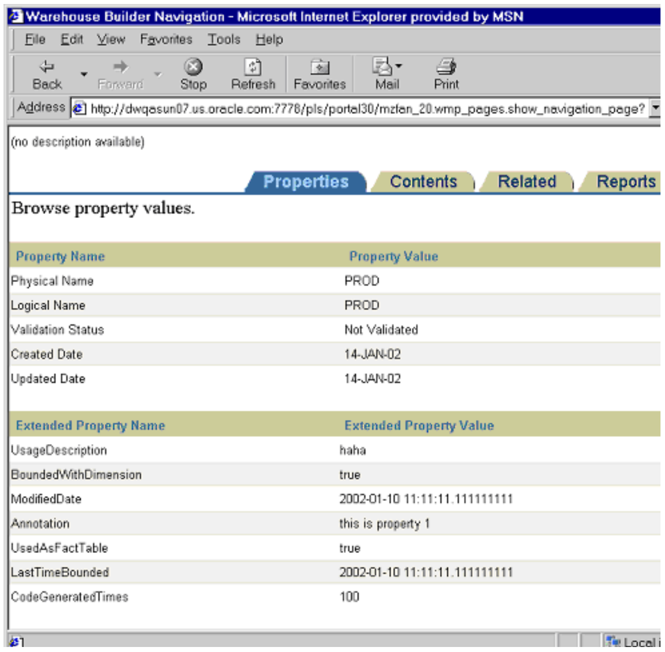
In the user interface, you can view UDPs in the following components:

- Warehouse Builder Client
- Warehouse Builder Design Browser

Warehouse Builder Client

Once you create a UDP using scripting, Warehouse Builder displays the UDP in the User Defined Properties tab on the associated properties sheet. The User Defined Properties tab does not appear until you create a UDP. For example, the Dimension Properties sheet typically does not display the User Defined Properties tab. However, once you add a UDP to a dimension, the UDP appears on the Dimension Properties sheet shown in Figure 19–1.

Figure 19–1 Sample Properties Sheet with User-Defined Properties


Warehouse Builder Navigation - Microsoft Internet Explorer provided by MSN
File Edit View Favorites Tools Help
Back Forward Stop Refresh Favorites Mail Print
Address
Properties
Contents
Related
Reports
Local i

http://dwqasun07.us.oracle.com:7778/pls/portal30/mzfan_20.wmp_pages.show_navigation_page?

(no description available)

Browse property values.

Property Name	Property Value
Physical Name	PROD
Logical Name	PROD
Validation Status	Not Validated
Created Date	14-JAN-02
Updated Date	14-JAN-02

Extended Property Name	Extended Property Value
UsageDescription	haha
BoundedWithDimension	true
ModifiedDate	2002-01-10 11:11:11.111111111
Annotation	this is property 1
UsedAsFactTable	true
LastTimeBounded	2002-01-10 11:11:11.111111111
CodeGeneratedTimes	100

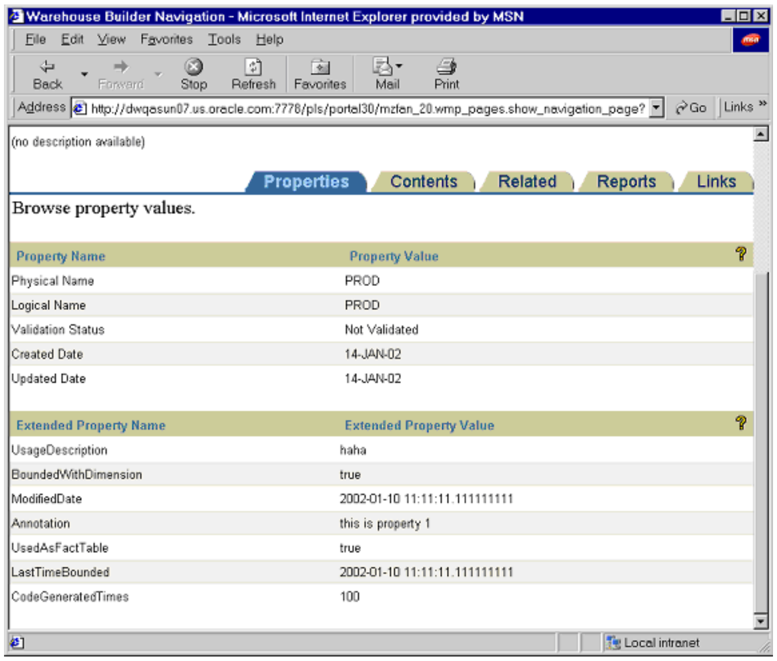
In the left panel, the tab displays the object navigation tree. In the right panel, the tab lists all the contained objects and corresponding extended properties. It shows the values and categories you specified when you created the UDP in scripting. You can modify the values but not the categories in the User Defined Properties tab. You must use OMB Plus to edit categories.

Warehouse Builder Design Browser

The Warehouse Builder Browser also displays UDPs. Warehouse Builder Browser is a metadata management and reporting portal for Warehouse Builder. Browser displays object properties, object relationships, and lineage and impact analysis reports.

If you define a UDP for a given object, the Browser lists the UDP name and values as Extended Property Name and Extended Property Value as shown in Figure 19–2.

Figure 19–2 Sample Properties Sheet with User-Defined Properties


Warehouse Builder Navigation - Microsoft Internet Explorer provided by MSN
File Edit View Favorites Tools Help
Back Forward Stop Refresh Favorites Mail Print
Address http://dwqasun07.us.oracle.com:7778/pls/portal30/mzfen_20.wmp_pages.show_navigation_page?
Go Links
Properties
Contents
Related
Reports
Links
Local intranet

(no description available)

Browse property values.

Property Name	Property Value
Physical Name	PROD
Logical Name	PROD
Validation Status	Not Validated
Created Date	14-JAN-02
Updated Date	14-JAN-02

Extended Property Name	Extended Property Value
UsageDescription	haha
BoundedWithDimension	true
ModifiedDate	2002-01-10 11:11:11.111111111
Annotation	this is property 1
UsedAsFactTable	true
LastTimeBounded	2002-01-10 11:11:11.111111111
CodeGeneratedTimes	100

Transferring UPDs to Other Repositories

The primary method for propagating changes from one repository to another is using MDL. The MDL enables you to export and import the metadata definition of the user-defined properties and its contents.

Exporting UDPs

You can export UDPs from the command line only. In the MDL Control file, the option is `DEFINITIONFILE=`*`filename`* to export the metadata definition. For example:

```
## Sample Export file
USERID=UserName/Password@HostName:PortID:OracleServiceName
#
DEFINITIONFILE=Drive:\DirectoryName\filename.mdd

FILE=Drive:\DirectoryName\filename.mdl
LOG=Drive:\DirectoryName\filename.log
```

Importing UDPs

You can import UDPs from the command line only. During import, MDL updates the user-defined properties for all objects.In the MDL Control file, the option is DEFINITIONFILE=filename to import the metadata definition. For example:

```
## Sample Import file
USERID=UserName/Password@HostName:PortID:OracleServiceName
#
DEFINITIONFILE=Drive:\DirectoryName\filename.mdd

FILE=Drive:\DirectoryName\filename.mdl
LOG=Drive:\DirectoryName\filename.log
```

You can import UDPs using one of the following search criteria:

- **Universal ID:** The metadata definition contains a Universal Object ID (UOID). The UOID uniquely identifies objects across repositories. If you import the MDL file by UOID, then MDL looks up the metadata definition by UOID. If the metadata definition name in the source MDL file is different from the metadata definition in the repository, then MDL renames it.
- **Physical Name:** MDL looks up the metadata definition by physical name.

Regardless of the import mode, MDL either adds the metadata definition if it does not exist in the repository, or updates the metadata definition if it already exists. MDL will not delete metadata definitions in the repository.

When updating the metadata definition, MDL only renames the object if the names are different (search criteria is by UOID), and update the default value. MDL does not change the datatype.

Managing Security with PL/SQL

This section contains the following topics:

- Maintaining Repository Users on page 19-5
- Plug-in Interface for the Security PL/SQL Package Specification on page 19-6
- Definitions of the Constants in the Package Specification on page 19-11
- Implementing the PL/SQL Interface on page 19-13
- Implementing Sample Security Policies on page 19-16

Maintaining Repository Users

Multiple identifiable Warehouse Builder users can access the same central repository schema when they are registered by the repository owner.

Warehouse Builder includes utility procedures for the following maintenance tasks:

- Update the role protecting password

  Although the repository owner does not explicitly use the protecting password for the role, it is recommended that the repository owner change the password often.

  The repository owner must connect to the repository schema using SQL Plus and issue the following statement:

  ```
  Call WBSecurityHelper.updateRolePwd('newpwd');
  ```

where `'newpwd'` is the new password used to protect the role chosen by repository owner. The changed password encryption appears in the table called OWB_ROLE_INFO table.

- Register repository users

  Warehouse Builder users must be database users located in the same database instance as the repository. The database user's default role should not be ALL. Use 'After user xxx DEFAULT Role x' to set the default role property.

  To register a repository user, execute the following statement in SQL Plus:

  ```
  call WBSecurityHelper.registerOWBUser ('username')
  ```

- Unregister repository users

  To unregister a user from a repository, execute the following statement in SQL Plus:

  ```
  Call WBSecurityHelper.unregisterOWBUser('username');
  ```

- List all repository users

  The repository owner must connect to the database containing the repository and issue the following statement:

  ```
  Set serveroutput on;
  Call WBSecurityHelper.listOWBUser( );
  ```

  `Set serveroutput on` is required since `listOWBUser()` uses the `DBMS_OUTPUT.put_line` for the output to dump the data to the user interface. Otherwise, `DBMS_OUTPUT.put_line` only dumps the output into an intermediate data structure.

  The preceding user maintenance tasks should be performed in the repository by the repository owner.

Plug-in Interface for the Security PL/SQL Package Specification

This section describes the plug-in interface specification for the PL/SQL security package provided by Warehouse Builder. You need to substitute the dummy PL/SQL package body provided by Warehouse Builder by implementing this interface in your Warehouse Builder repository. The interface specification and the dummy implementation are also available from your repository schema.

The functions and procedures described in the following section are invoked for all the Warehouse Builder operations if the function `isSecurityServiceCustomized` is changed to return a value of 1. The default return value of this function, if you do not customize the package, is 0.

The functions and procedures described in the following section are empty when you install Warehouse Builder. To implement these functions and procedures, you must implement a security logic within them and build your own security data model to connect objects such as operations, users, and object types.

```
CREATE OR REPLACE PACKAGE WBSecurityServiceImpl AS

FUNCTION isSecurityServiceCustomized RETURN NUMBER;

/*
```

Use Function: isSecurityServiceCustomized() to differentiate the implementation of security service, whether you customize the security PL/SQL implementation or use the implementation provided by Warehouse Builder.

Return value: this function returns 1 if you want to implement the PL/SQL package of this specification, otherwise it returns 0;

```
*/
PROCEDURE securityCheckForCreation(outcome OUT NUMBER,
userId IN VARCHAR2,
objectUOIDOperationInvokedOn IN VARCHAR2,
status IN VARCHAR2,
parentModuleUOID IN VARCHAR2,
parentProjUOID   IN VARCHAR2,
repos_Schema IN VARCHAR2,
objectType IN NUMBER);

/*
```

Procedure: securityCheckForCreation: Used to create an operation security check. When you try to create an object, Warehouse Builder calls this procedure to ask the implementation whether the creation operation is acceptable or not.

Argument explanation:

Outcome: 1: The creation operation is acceptable.

Outcome: 0: The creation operation is not acceptable.

userId: The login user's database user name.

objectUOIDOperationInvokedOn: The parent folder UOID, where you create a new object. For objects, such as Projects and Snapshots, that do not have folder parent, this argument is NULL.

status: An attribute of a module: WB_DEV_STATUS, WB_QA_STATUS, WB_PROD_STATUS defined in this specification. This attribute describes the status of the module. For objects, such as Projects, Modules, or Snapshots, that are not children of any module in the hierarchy, this argument is NULL.

parentModuleUOID: The UOID of the module. For objects such as Projects or Snapshots that are not children of any module in the hierarchy, this argument is NULL.

parentProjUOID: The UOID of the project. For objects such as Projects or Snapshots that are not children of any module in the hierarchy, this argument is NULL.

repos_schema: The central repository schema name you are working on.

objectType: The type of object you want to create. It is one of the object type constants defined in this specification.

```
*/
PROCEDURE securityCheck(outcome OUT NUMBER,
           userId IN VARCHAR2,
    operation IN NUMBER,
    objectUOIDOperationInvokedOn IN VARCHAR2,
    objectTypeOperationInvokedOn IN NUMBER,
    status IN VARCHAR2,
    parentModuleUOID IN VARCHAR2,
    parentProjUOID   IN VARCHAR2,
    repos_Schema IN VARCHAR2);
/*
```

PROCEDURE: securityCheck is used for the following operations:

```
WB_EDIT
WB_DELETE
WB_VALIDATE
WB_GENERATION
WB_VERSION
```

Whenever you invoke one of the preceding listed operations, Warehouse Builder calls this procedure to see whether the operation is acceptable or not.

Argument explanation:

Outcome: 1: The creation operation is acceptable.

Outcome: 0: The creation operation is not acceptable.

userId: The login user's database user name.

operation: One of constants listed earlier.

objectUOIDOperationInvokedOn: The target object's UOID.

objectTypeOperationInvokedOn: The type of object the operation is invoked on (one of the object type constants defined in this specification).

status: An attribute of a module.

WB_DEV_STATUS

WB_QA_STATUS,

WB_PROD_STATUS

This attribute is used to describe the status of the module. If you operate on a project, module, or snapshot, this argument is NULL.

parentModuleUOID: The UOID of the module. If you invoke the operation on a module, then the objectUOIDOperationInvokedOn and the parentModuleUOID are the same. If you invoke an operation on a project or snapshot that is not a child of any module in the hierarchy, this argument is NULL.

parentProjUOID: The UOID of the project. If you invoke the operation on a project, then the objectUOIDOperationInvokedOn and the parentProjUOID are the same. If you invoke the operation on a snapshot that is not the child of any project in the hierarchy, this argument is NULL.

repos_schema: The central repository schema name you are working on

```
*/
PROCEDURE securityCheckForService(outcome OUT NUMBER,
           userId IN VARCHAR2,
    serviceOp IN NUMBER,
    moduleUOID IN VARCHAR2,
    projUOID   IN VARCHAR2,
    repos_Schema IN VARCHAR2);

/*
```

PROCEDURE securityCheckForService is used for the following service operations:

WB_DEPLOY

WB_MDL_IMPORT

WB_MDL_EXPORT

WB_BRIDGE_IMPORT

WB_BRIDGE_EXPORT

WB_SOURCE_IMPORT

WB_RUNTIME_EXECUTE

WB_SNAPSHOT_RESTORE

Argument explanation:

Outcome: 1: The creation operation is acceptable.

Outcome: 0: The creation operation is not acceptable.

userId: The login user's database user name.

serviceOp: One of constants listed earlier.

moduleUOID: The UOID of a module on which the user is invoking operation:serviceOp. The result is NULL for serviceOp:WB_DEPLOY,WB_BRIDGE_EXPORT, WB_RUNTIME_EXECUTE. It is valid for: WB_MDL_IMPORT, WB_MDL_EXPORT, WB_SOURCE_IMPORT, WB_SNAPSHOT_RESTORE.

projUOID: The UOID of a project on which the user is invoking the operation: serviceOp on it. The result is NULL for the serviceOp:WB_DEPLOY, WB_BRIDGE_EXPORT, WB_RUNTIME_EXECUTE. It is valid for: WB_MDL_IMPORT, WB_MDL_EXPORT, WB_SOURCE_IMPORT, WB_SNAPSHOT_RESTORE.

repos_schema: The central repository schema name you are working on

*/

--BEGIN CONSTANT DEFINITION

```
CUSTOM_SHARED_LIBRARY CONSTANT VARCHAR2(100):='9E012195D16211D48D7100B0D02A59E8';
/*
```

CUSTOM_SHARED_LIBRARY: The UOID constant for a predefined Warehouse Builder folder: Custom shared library. On the UI, this folder is called Custom located under the Public Transformation node.

This folder contains all global shared library transformations created by the user.

To control the users who have permission to create transformations under the Custom folder, you can check if the argument objectUOIDOperationInvokedOn in procedure securityCheckForCreation equals to this constant or not. To control users who can invoke the operations such as WB_EDIT or WB_DELETE, on a shared function or a procedure, check if the argument parentModuleUOID in the procedure securityCheck equals this constant or not.

Since this module is predefined, you cannot change the status or use the status of this module for access control.

*/

--Definition of constants for all basic operations

```
WB_EDIT   CONSTANT INTEGER := 0;
WB_DELETE   CONSTANT INTEGER := 1;
WB_REFERENCE   CONSTANT INTEGER := 2;
WB_CREATE        CONSTANT INTEGER := 3;
WB_VALIDATE   CONSTANT INTEGER := 4;
WB_GENERATION   CONSTANT INTEGER := 5;
WB_VERSION  CONSTANT INTEGER := 6;
/*
```

Use this group of operation constants in the SecurityCheck procedure. Use one of the preceding constants for the argument "operation". You can also use it to control which user can invoke this operation.

*/

--Definition of constants for all service type operations

```
WB_DEPLOY   CONSTANT INTEGER := 100;
WB_MDL_IMPORT   CONSTANT INTEGER := 101;
WB_MDL_EXPORT   CONSTANT INTEGER := 102;
WB_BRIDGE_IMPORT CONSTANT INTEGER := 103;
WB_BRIDGE_EXPORT CONSTANT INTEGER := 104;
WB_SOURCE_IMPORT CONSTANT INTEGER := 105;
WB_RUNTIME_EXECUTE  CONSTANT INTEGER :=106;
WB_SNAPSHOT_RESTORE   CONSTANT INTEGER := 107;
/*
```

Use this group of service operation constants in SecurityCheckForService procedure. Use one of these constants for the argument "serviceOp". You can also use it to control which user can invoke this service operation.

*/

--Definition of the module status

```
WB_DEV_STATUS  CONSTANT VARCHAR2(100) := 'DEV_STATUS';
WB_QA_STATUS  CONSTANT VARCHAR2(100) := 'QA_STATUS';
WB_PROD_STATUS  CONSTANT VARCHAR2(100) := 'PROD_STATUS';
/*
```

Use this group of module status constants should be used in the securityCheckForCreation and SecurityCheck procedures. Use the preceding constants if you want to create a child of a module (for securityCheckForCreation) or if you invoke an operation on a module or the child of a module (for SecurityCheck), otherwise the argument "status" will be null. You can also use it to implement your access control based on the status of the module.

*/

--Definition of object type

```
WB_PROJECT   CONSTANT INTEGER := 1;
WB_ORACLE_MODULE   CONSTANT INTEGER := 2;
WB_GATEWAY_MODULE CONSTANT INTEGER := 3;
WB_SAP_MODULE  CONSTANT INTEGER := 4;
WB_FLAT_FILE_MODULE CONSTANT INTEGER := 5;
WB_SHARED_MODULE  CONSTANT INTEGER := 6;
WB_REPOS_MODULE   CONSTANT INTEGER := 7;
WB_COLLECTION  CONSTANT INTEGER := 8;
WB_WAREHOUSE   CONSTANT INTEGER := 9;
WB_TABLE   CONSTANT INTEGER := 10;
WB_VIEW    CONSTANT INTEGER := 11;
WB_MATERIALIZED_VIEW  CONSTANT INTEGER := 12;
WB_SEQUENCE   CONSTANT INTEGER := 13;
WB_DIMENSION_TABLE  CONSTANT INTEGER := 14;
WB_CUBE_TABLE   CONSTANT INTEGER := 15;
WB_FLAT_FILE   CONSTANT INTEGER := 16;
WB_PACKAGE   CONSTANT INTEGER := 17;
WB_TRANSFORMATION   CONSTANT INTEGER := 18;
WB_MAPPING   CONSTANT INTEGER := 19;
WB_MIV_MODULE   CONSTANT INTEGER := 20;
WB_CONNECTOR   CONSTANT INTEGER := 21;
```

```
WB_LOCATION   CONSTANT INTEGER := 22;
WB_RUNTIME_REPOSITORY   CONSTANT INTEGER := 23;
WB_BUSINESS_AREA CONSTANT INTEGER := 24;
WB_INTELLIGENCE_MODULE CONSTANT INTEGER := 25;
WB_PROCESS_FLOW  CONSTANT INTEGER := 26;
WB_PROCESS_FLOW_MODULE  CONSTANT INTEGER := 27;
WB_PROCESS_FLOW_PACKAGE CONSTANT INTEGER:=  28;
WB_QUERY_OBJECT  CONSTANT INTEGER:=  29;
WB_ADVANCED_QUEUE  CONSTANT INTEGER:=  30;
WB_EXTERNAL_TABLE  CONSTANT INTEGER:=  31;
WB_REPORT   CONSTANT INTEGER:=  32;
WB_REPORT_GROUP   CONSTANT INTEGER:=  33;
WB_REPORT_MODULE  CONSTANT INTEGER:=  34;
WB_OBJECT_TYPE   CONSTANT INTEGER:=  35;
WB_SNAPSHOT  CONSTANT INTEGER:=  36;
/*
```

This group of object type constants must be used in procedures securityCheckForCreation and securityCheck. The argument "objectType" in securityCheckForCreation and argument "objectTypeOperationInvokedOn" in securityCheck will be one of the preceding constants. You can use this constant to control which user can create which kind of object or which user can invoke operations such as WB_EDIT or WB_DELETE on what types of objects.

*/

/*

Because there are many arguments, the procedures will depend on whether the object on which the user invokes an operation is the child of the project or a module. The information is listed in the following section.

*/

--Children of Project

/*

While WB_SNAPSHOT and WB_PROJECT are not children of a project, other objects in the list of type constants are.

*/

--Children of Module

/* The following are not children of any module: WB_PROJECT, WB_ORACLE_MODULE, WB_GATEWAY_MODULE, WB_SAP_MODULE, WB_FLAT_FILE_MODULE, WB_SHARED_MODULE, WB_REPOS_MODULE, WB_COLLECTION, WB_WAREHOUSE, WB_MIV_MODULE, WB_LOCATION, WB_CONNECTOR, WB_RUNTIME_REPOSITORY, WB_BUSINESS_AREA, WB_INTELLIGENCE_MODULE, WB_PROCESS_FLOW_MODULE, WB_REPORT_MODULE, WB_SNAPSHOT.

Other object types can be the children of module.

*/

```
END WBSecurityServiceImpl;
```

Definitions of the Constants in the Package Specification

The following section lists the constant definitions of basic operations. For these operations, Warehouse Builder checks if the operation is acceptable or not on an object instance level

- **`WB_EDIT`**: Edit (update) operation
- **`WB_DELETE`**: Delete operation
- **`WB_REFERENCE`**: The operation you want to reference an object within another object. For example, when you want to drag a table into a mapping as an operation.
- **`WB_CREATE`**: Create operation
- **`WB_VALIDATE`**: Validate operation used to check the accuracy of the object definitions
- **`WB_GENERATION`**: Generate the SQL scripts for the definitions in Warehouse Builder
- **`WB_VERSION`**: Versioning objects which means user can add an object to a snapshot version, remove an object from a snapshot version, or replace an existing versioned object with a new copy.

The following section lists the constant definitions for service operations. For these operations, Warehouse Builder checks if the operation is acceptable or not on a system wide level. If a user has the privilege to invoke a service operation, he should invoke this operation on any object. For operations such as WB_MDL_IMPORT, WB_MDL_EXPORT, WB_BRIDGE_IMPORT, WB_SOURCE_IMPORT, WB_SNAPSHOT_RESTORE, Warehouse Builder provides security at the project or module level.

- **`WB_DEPLOY`**: First generates the SQL script (if does not exist) and then deploy the SQL scripts to a runtime database schema.
- **`WB_MDL_IMPORT`**: Imports the metadata into Warehouse Builder from a flat file format.
- **`WB_MDL_EXPORT`**: Exports the metadata from Warehouse Builder into a flat file.
- **`WB_BRIDGE_IMPORT`**: Another importing solution. Warehouse Builder also provides Discoverer, OLAP, CWM, and third party bridges that use a different format from MDL.
- **`WB_BRIDGE_EXPORT`**: Another exporting solution. Warehouse Builder also provides Discoverer, OLAP, CWM, and third party bridges that use a different format from MDL.
- **`WB_SOURCE_IMPORT`**: Imports the metadata information of database objects, such as tables and views, into Warehouse Builder from a given database link.
- **`WB_RUNTIME_EXECUTE`**: Executes the deployed SQL script from the runtime database schema.
- **`WB_SNAPSHOT_RESTORE`**: Restores the design space from a given snapshot, selectively or fully.

Constant definitions of the module status. You can change the module status from the Warehouse Builder client using the property page of a module.

- **`WB_DEV_STATUS`**: The module is under development.
- **`WB_QA_STATUS`**: The module is in QA status.
- **`WB_PROD_STATUS`**: The module is in production.

Constant definitions of object types.

- WB_PROJECT: Project
- WB_ORACLE_MODULE: Oracle Module

- WB_GATEWAY_MODULE: Gateway Module
- WB_SAP_MODULE: SAP Module
- WB_FLAT_FILE_MODULE: Flat File Module
- WB_SHARED_MODULE: Shared Module
- WB_REPOS_MODULE: Repository Module
- WB_COLLECTION: Collection
- WB_WAREHOUSE: Warehouse Module
- WB_TABLE: Table
- WB_VIEW: View
- WB_MATERIALIZED_VIEW: Materialized View
- WB_SEQUENCE: Sequence
- WB_DIMENSION_TABLE: Dimension Table
- WB_CUBE_TABLE: Cube Table
- WB_FLAT_FILE: Flat File
- WB_PACKAGE: Package
- WB_TRANSFORMATION: Transformation
- WB_MAPPING: Mapping
- WB_MIV_MODULE: MIV Module
- WB_CONNECTOR: Connector
- WB_LOCATION: Location
- WB_RUNTIME_REPOSITORY: Runtime Repository
- WB_BUSINESS_AREA: Business Area
- WB_INTELLIGENCE_MODULE: Intelligence Module
- WB_PROCESS_FLOW: Process Flow
- WB_PROCESS_FLOW_MODULE: Process Flow Module
- WB_PROCESS_FLOW_PACKAGE: Process Flow Package
- WB_QUERY_OBJECT: Query Object
- WB_ADVANCED_QUEUE: Advanced Queue
- WB_EXTERNAL_TABLE: External Table
- WB_REPORT: Report
- WB_REPORT_GROUP: Report Group
- WB_REPORT_MODULE: Report Module
- WB_OBJECT_TYPE: Object Type
- WB_SNAPSHOT: Snapshot

Implementing the PL/SQL Interface

When you implement the PL/SQL interface, you decide whether an operation is accepted or rejected based on the arguments passed by the Warehouse Builder client.

Warehouse Builder also provides a public view named `ALL_IV_FIRSTCLASS_OBJECTS` in its repository. This view contains the following information about an object:

- Object UOID and class type
- Object name and business name
- Created by and updated by
- Creation and update timestamps

Given the object UOID, you can reference the public view for the preceding information. You can use this information from the public view to decide whether the operation is acceptable or not.

The arguments passed by the caller provide you with different levels of security granularity:

- **Repository level:** Check the `repos_Schema` argument as to whether to freeze the whole repository.
- **Project level:** Check the argument `parentProjUOID` to freeze the whole project or not.
- **Module level:** Check the argument `parentModuleUOID` to freeze the whole module or not.
- **Object class type level:** Check the argument `objectTypeOperationInvokedOn` to accept or prevent access to the entire type.
- **Object instance level:** Check the object UOID.
- **Development process level:** Check the argument: `status` (development, quality assurance, or production) to make access control decisions for an entire module. The status is an attribute of any module that you can change using the Warehouse Builder user interface through the module properties page.

Considerations

To implement the security framework, you need to deploy your own security data model. For example, for object level security, you need to create a table or a series of tables that contain information on which user can perform which operation on an object. You also need to implement your own procedures or user interface to maintain this data model.

The following are considerations for implementing the PL/SQL interface:

- If you base your access control decisions on the status of a module, Warehouse Builder recommends that only users in the administration group have the access to edit the module.
- If a user has the validate/generate privilege on an object instance, such as a Table, and has no edit privilege on the object instance, the user can only validate/generate the object without the generated result persisting.
- When you invoke an operation (operation A) from the Warehouse Builder client, in rare cases, operation B is invoked underneath. This can result in a situation where you have granted a user permission to invoke operation A, but the user cannot successfully finish operation A because s/he has no permission to invoke operation B.

To resolve this issue, you need to either grant the user privileges for both operations A and B, or not grant the user privileges on either operation.

For example, in the mapping editor, when a bound object is added to a mapping, or when inbound reconcile is performed on an object, Warehouse Builder automatically creates a connector to the bound object's location if one does not already exist. To allow the user to finish adding a bound object into mapping editor, or to finish the inbound reconcile, you need to grant the user `EDIT` permission on the mapping and `CREATE` permission for the connector.

- To implement the PL/SQL procedures, you must not issue any commit or rollback command.

  Because the connection used to call the security PL/SQL procedures is the connection used to persist all the metadata, this connection is controlled by Warehouse Builder through its own transaction manager to determine when to commit or rollback.

  If you invoke commit or rollback in the procedure, it will corrupt the repository. If you want to record debug information in your PL/SQL procedure, you can do it in a separate child procedure, with autonomous transactions (use progma autonomous_transaction at the beginning of the procedure). In this procedure, you can issue a commit or rollback command.

- If you grant a user the permission to `CREATE` an object, the user also needs permission to `EDIT` the object. After an object is created, you can change your policy to revoke the user's `EDIT` permission on the object.

- Do not perform any DML operations on the repository database objects created by Warehouse Builder when you implement the security PL/SQL procedures. This may damage the repository and make the Warehouse Builder client malfunction or stop functioning.

  If you install the implementation package for security while another user is logged into the Warehouse Builder repository, you may receive the following SQL Error: ORA-04061: Existing state of package body "OWB_REPO_ 513.WBSECURITYSERVICEIMPL" has been invalidated. The user must exit Warehouse Builder and restart the installation.

- For the following procedure:

```
PROCEDURE securityCheckForService(outcome OUT NUMBER,
           userId IN VARCHAR2,
    serviceOp IN NUMBER,
    moduleUOID IN VARCHAR2,
    projUOID   IN VARCHAR2,
repos_Schema IN VARCHAR2)
```

  The moduleUOID and projUOID refer to the UOID of modules and projects in . Warehouse Builder. The security only applies to objects in the Warehouse Builder repository.

  If you try to import an MDL file or if you try to restore a snapshot containing projects or modules that do not exist in the repository, then Warehouse Builder will not perform any security check on those projects or modules. It is recommended that the repository owner perform MDL import or snapshot restore on folder objects such as projects and modules.

Implementing Sample Security Policies

This section describes three types of advanced security policies available through Warehouse Builder:

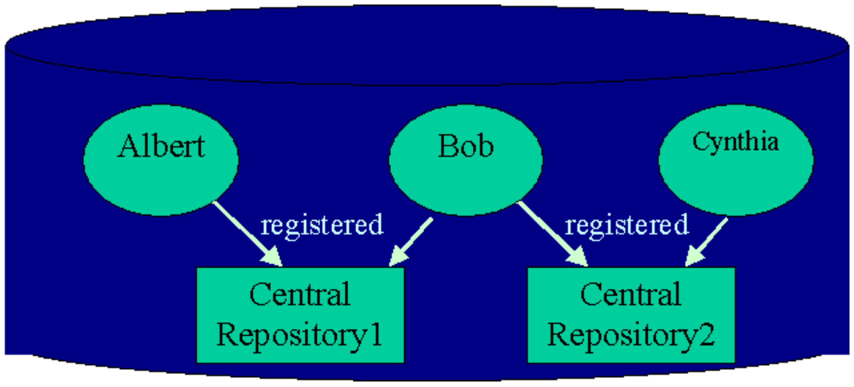
- **Proactive Security:** Warehouse Builder enables you to plug in a customized security PL/SQL implementation package in the Warehouse Builder repository to provide tailored access control to users according to the security rules defined by your organization.
- **Reactive Security:** Tracks audit information using metadata history and also determines security policies from such audit information.
- **Data Stewardship:** Warehouse Builder enables an individual or a group of individuals to "own" portions of the metadata rather than the technical administrators. Metadata ownership thus becomes an important component of metadata security management. The data steward's role is important when the metadata repository is used to model the source, target, and ETL operations for your enterprise, and to capture the structure of your enterprise for business users through Portals or ad-hoc query tools. Warehouse Builder enables you to incorporate the responsibility of the data steward into a predefined metadata security authorization policy.

The Warehouse Builder Security Architecture

The following sections describe the Warehouse Builder security model and architecture, and the various ways in which the tool can leverage existing infrastructure security systems.

The server-side security architecture of Warehouse Builder is shown in Figure 19–3.

Figure 19–3 Warehouse Builder Server-Side Security Architecture



The repository database has five schemas including two Warehouse Builder repositories (also known as Central Repositories or Warehouse Builder Repositories) and three repository users: Albert, Bob and Cynthia. Repository user schemas are always located in the same database instance as the central repository. Warehouse Builder requires that these design repositories have an assigned Administrator to administer the registration and deregistration of repository users. Only the Administrator of a Warehouse Builder repository can access the central repository schema. Although Albert, Bob and Cynthia have schemas in the database, in order for them to have access to a specific Warehouse Builder repository, they need to be registered by the Administrator of that specific central repository. Because they are not the owners of the Warehouse Builder metadata tables, these users cannot modify the metadata through other access tools such as SQL Plus.

For additional information, see "Maintaining Repository Users" on page 19-5.

Using a Customizable Security Authorization Framework

Warehouse Builder publishes a PL/SQL Security Package Specification and loads it into the central repository. By default, the package implementation that is loaded when you create a new Warehouse Builder repository gives all permissions to all registered users. However, Warehouse Builder administrators can design their own security policies and implement these policies according to the framework of the security package specification. This customized security implementation can be plugged into the Warehouse Builder central schema. All subsequent Warehouse Builder actions in the central repository pass through this new security policy.

Security checks are made at every point in the tool when you try to modify an object. For example, when you create, update, or delete an object. The security checks call the PL/SQL security implementation package in the central repository. For details on the specification for the security implementation package, see "Implementing the PL/SQL Interface" on page 19-13.

The security implementation package can use the information available in the central repository. For example, an Excel spreadsheet can be used to generate the relational lookup tables to indicate to the Warehouse Builder client whether a permission is granted or not.

By default, the Warehouse Builder Security Implementation Package allows all operations. Table 19–1 shows sample security permissions.

Table 19–1 Permissions for User Albert

Object	Create	Edit	Delete	Generate	Deploy
Table	Y	Y	Y	N	N
View	Y	Y	Y	N	N
Materialized View	Y	Y	Y	N	N
Dimension	N	N	N	N	N
Cube	N	N	N	N	N
Mapping	N	Y	N	N	N
Process Flow	N	Y	N	N	N

Freezing Projects

If you want to freeze the project MY_PROJECT and prevent access to all its contents, the following restrictions will apply:

- You cannot create, edit, or delete any objects under a frozen project.
- You cannot invoke any of the services that modify objects within this frozen project. For example, you cannot perform an MDL import, a source import, or a snapshot restore in this project.
- You can deploy, export, and execute runtime procedures within a frozen project.
- You can validate and generate within a frozen project.
- You cannot add or remove any objects from a frozen project to a snapshot.

The frozen project security policy is implemented within Warehouse Builder through the following files. These files are located on your installation CD under: *samples/security_feature/frozenproject*.

frozenProject.pkb: Holds the implementation of the security policy.

frozenProject.sql: Contains a table of the structure as shown in Table 19–2. The administrator can freeze projects by inserting them into this table and setting the isFrozen flag to 1.

For Table 19–2, the repository owner must issue the following SQL statements to frozen and unfrozen projects from SQL Plus:

```
insert into frozen_projects (projectName,isFrozen) values('SAMPLEPROJECT1', '1');
insert into frozen_projects (projectName,isFrozen) values('SAMPLEPROJECT2', '0');
insert into frozen_projects (projectName,isFrozen) values('SAMPLEPROJECT3', '1');
commit;
```

Table 19–2 Example File Structure for frozenProject.sql

Project Name	isFrozen
SampleProject 1	1
SampleProject 2	0
SampleProject 3	1

Development Cycle Based Security

If your data warehousing project passes through cycles of Development, QA, and Production phases, Warehouse Builder provides flexibility in determining your security policy, with regards to metadata definitions and changes during these different phases. For example, you have classified your users into one or more of the following groups: Engineering, Quality Assurance, and Sustaining Engineering. Depending on the various phases in the cycle, only certain actions are permitted to certain groups. Table 19–3 lists the permissible actions for each group.

Legend for Table 19–3: A = Administrators (the repository owner), E = Engineers, Q = Quality Assurance, S = Sustaining Engineers.

Table 19–3 Permissible Actions for Different Groups

Action	Development	QA	Production
Create	A,E	A,Q	A,S
Edit	A,E	A,Q	A,S
Delete	A,E	A,Q	A,S
Reference	A,E	A,Q	A,S
Validate	A,E	A,Q	A,S
Generate	A,E	A,Q	A,S
Version	A,E	A,Q	A,S
MDL Import	A,E	A,Q	A,S
Bridge Import	A,E	A,Q	A,S
Source Import	A,E	A,Q	A,S
Snapshot Restore	A,E	A,Q	A,S
Runtime Execute	A,E,Q,S	A,E,Q,S	A,E,Q,S
Deployment	A,E,Q,S	A,E,Q,S	A,E,Q,S
MDL Export	A,E,Q,S	A,E,Q,S	A,E,Q,S

Table 19–3 (Cont.) Permissible Actions for Different Groups

Action	Development	QA	Production
Bridge Export	A,E,Q,S	A,E,Q,S	A,E,Q,S

For example, only Administrators and Sustaining Engineers are allowed to create, edit, or delete objects under a module that has been marked Production.

This security policy contains the following exceptions:

- Only the administrators (repository owners) can invoke operations on objects that are at a higher level than modules. For example, projects and snapshots.
- QA and Sustaining Engineering cannot create or delete modules.
- Any user can perform operations on non-module dependent objects such as locations, collections, connectors, and business areas.
- The global shared library and its transformation objects are accessible to all users.
- Only a member of a group can change the module status to a different status. For example, only a QA member can change a module's development status from QA to something else. Only Engineers can change the module's status from Development to something else. This can be used to implement a disciplined bug-fixing policy within a development group.

The following files are used to implement this particular security policy: These files are located on your installation CD under: *samples/security_feature/developcycle.*

developCyclePolicyLoader.sql: Contains the information specified in Table 19–3.

developCyclePolicy.sql: Contains a table to track Users to Groups. For example, if your table is populated as shown in Table 19–4:

For Table 19–4, the repository owner must issue the following SQL statements to assign user groups using SQL Plus:

```
insert into user_group_assignment(userName,groupID) values('ALBERT1', 1);
insert into user_group_assignment(userName,groupID) values('ANDREW', 3);
insert into user_group_assignment(userName,groupID) values('BOB', 2);
insert into user_group_assignment(userName,groupID) values('CYNTHIA', 1);
```

Table 19–4 Sample Table Tracking Users to Groups

Username	OWBGroup
Albert	Engineer
Albert	Sustaining Engineer
Bob	Quality Assurance
Cynthia	Engineer
...	...

developCyclePolicy.pkb: Contains the main business logic to implement the development cycle based security policy.

In the implementation, the status flag is used to check whenever an object is manipulated. In Warehouse Builder, because the development status can be defined at the granularity of a module, the development status is only relevant for objects that are descendents of a module.

Warehouse Builder only provides security for objects in the repository if a user tries to import an MDL file or to restore a snapshot containing projects or modules that do not exist inside the repository. The sample implementation `developCyclePolicy.pkb` will not prevent the user from creating projects or modules and their children through an MDL import or snapshot restore. It is recommended that the repository owner perform the MDL import or snapshot restore on folder objects such as projects and modules.

Reactive Security and Audit-based Security

Reactive security pertains to the identification and tracking of potential security security breaches. Auditing is used for security, to track ownership, and to ensure quality. In Warehouse Builder, auditing is performed at an object level. Thus, auditing trails can be defined at the finest granularity. For example, who and when a particular column, constraint, or level attribute was created or updated. You can view this information using the Warehouse Builder Browser if the users are set up using the Warehouse Builder security architecture described earlier.

This audit information can also be used to enforce security policies. You can infer ownership of a particular object by the user that created it. Your security policy can consist of preventing anyone other than object's owner's group from performing any action on an object. For example, Albert from Marketing has created a new Promotions cube. He has added all the measures and dimensions to it. Bob, who is in Inventory has no permissions to modify, delete, generate, or deploy the Promotions cube. However, Cynthia, also from Marketing, has the same permissions as Albert and can make any modifications (including deleting) on the Promotions cube.

The following security implementation requires you to locate the object in the central repository to find out who created the object. The object creator is captured in the audit column createdBy, which is available for any object in Warehouse Builder.

You can locate the object within the Warehouse Builder Public Views system. If you want to write generic code for any type of Warehouse Builder object, then a join to the ALL_IV_FIRSTCLASS_OBJECTS view is recommended.

To locate an object within the public views, you need to join the argument object_ UOID to the UOID exposed in the ALL_IV_FIRSTCLASS_OBJECTS view. The UOID stands for the Unique Object Identifier and enables you to uniquely identify an object within and across any repository.

Use the following policies for basic operations:

- Object X can only be created if the container object's (parent object's) createdBy points to a user who belongs to the same division as the Object X's creator's division.
- Object X can only be modified, deleted, validated, generated, or versioned if the createdBy points to a user who belongs to the same division as the Object X's creator's division.
- Regular users cannot invoke service operations such as deploy, MDL import or export, Bridge import or export, source import, runtime execution, and snapshot restore. Although this can be relaxed by modifying the security package implementation, it is important to keep this restriction for imports and snapshot restore services. These operations can only be invoked by administrators.

This security policy requires you to define Administrators for each division. These Group Administrators are allowed to perform critical operations, such as MDL import, source import, and snapshot restore. These operations can also modify or delete objects that are not within a particular Administrator's group. Therefore, this is an important role. This restriction is imposed because Warehouse Builder does not perform security checks at the first class object level when service operations such as MDL import, source import, and snapshot restore are invoked. Administrators are defined according to the structure shown in Table 19–5.

Table 19–5 Administrator Roles

User	Group (Foreign Key)	IsAdmin
Albert	Marketing	0
Bob	Sales	0
Cynthia	Marketing	1
Derrick	Finance	0
Eeyore	Sales	1

The central Administrator (or repository owner) is allowed to perform any operation. Objects that the central Administrator creates do not belong to any group and can be shared across all groups. This is recommended if divisions need to share container objects such as projects or modules.

Rules for the Administrator role are listed in the following section:

- Division Administrators are responsible for performing service operations that modify a group of metadata objects (that may not necessarily be within the Division Administrator's division). After the objects are created using MDL import, then anyone in the Administrator's division can edit or delete the object.
- The Division Administrator can also perform any operation normally restricted to the regular user, such as modifying or deleting objects from another division.
- The Central Administrator is responsible for creating the container objects, such as projects and warehouse modules. Users from any division can create objects under this shared container.
- The Central Administrator can also perform any operation normally restricted to the regular user, such as modifying or deleting objects from any division.

Two following files can be used to enforce this policy. These files are located on your installation CD under: *samples/security_feature/creatorIsOwner.*

creatorIsOwner.sql: Contains the tables illustrated in Table 19–5.

creatorIsOwner.pkb: Contains the security implementation that enforces this policy.

Data Stewardship

The creator of an object belongs to a pool of data stewards responsible for the quality of metadata in a specific subject area such as Marketing and Inventory, Sales, Budget, Production, and Finance. In Warehouse Builder, these subject areas are captured using the Warehouse Builder collection object. The collection objects consist of shortcuts to actual objects in Warehouse Builder, such as modules, tables, facts, mappings, process flows, dimensions, and files, that appear in the Warehouse Builder navigation tree. These collection objects enable you to classify Warehouse Builder objects according to subject areas. A collection does not imply ownership. Certain pieces of metadata can belong to more than one collection. For example, the metadata for the Customers dimension may be shared by data stewards in the Marketing, Sales, and Order Fulfillment divisions.

When there are multiple owners for certain types of metadata, you cannot implement the Creator Is Owner policy implementation described in the previous section. Many data stewards can have security privileges on the same metadata object. Also, the creator of an object's metadata may not necessarily be the assigned data steward for that metadata. For example, if the Warehouse Builder Administrator imports a number of tables from a schema.

The Warehouse Builder collection object is the pivotal element in the data stewardship security policy. Access to data stewardship must be restricted to prevent users from adding new objects to that list. In the preceding example, only the Warehouse Builder Administrator is allowed to add to the collections lists. Every time Developers create an object, they need to inform the Administrator to add it to the collections list. A condition must be added to the security policy to allow the creator of the object to modify that object. Therefore, there is only one Administrator in this policy, rather than one administrator for each division.

To understand this policy, you need to understand the behavior of the shortcuts defined under a collection. There are two types of shortcuts defined in a collection. An explicit shortcut is one that the user has explicitly created and an implicit shortcut is one that is created by Warehouse Builder when a user creates a child shortcut without the parent. For example, when you add a table to a collection, Warehouse Builder automatically adds all the parent objects of this table in the collection. The table here is an explicit shortcut and its parent folder, such as the Oracle Module, is an implicit shortcut. Implicit shortcuts are deleted when the last child is deleted. For example, if a user deletes the last shortcut to a table and if the shortcut to its parent module was implicit, the module shortcut is also deleted.

Users and their corresponding division collections must be set up according to the structure described in Table 19–6.

Table 19–6 Users and Corresponding Division Collections

Collection ID (UK)	Collection Name	Project Name	Coll ID (FK)	User Name
1	Sales Collection	My Project	1	Arthur
2	Finance Collection	My Project	2	Billy
3	Marketing Collection	My Project	3	Caroline
4	AR Collection	My Project	4	David
5	AP Collection	My Project	3	Edward
6	Inventory Collection	My Project	4	Frederick
7	Manufacturing Collection	My Project	2	George
8	HR Collection	My Project	3	George

In Table 19–6, user George belongs to more than one division, and therefore has access to objects in both collections. The collections must match the collection name and id as defined in Warehouse Builder. By using the preceding table, you can limit the number of collections that can be used for administrative security purposes. Warehouse Builder can define other collections that have nothing to do with the security infrastructure.

This security policy is described as follows:

- When a user creates an object, only that user and the Central Administrator can modify, delete, validate, or generate that object until that object is registered in a collection.
- Only the Central Administrator has the privilege to modify collections.
- Only the Central Administrator is allowed to perform service tasks such as deploy, MDL import/export, Bridge import/export, source import, runtime execution and snapshot restore.
- Users can only edit, delete, validate, or generate an object that they have not created if that object is explicitly registered under a collection accessible to that user. An explicit shortcut must be used in the collection to prevent users from deleting a module that is registered through an implicit shortcut.
- A user can create an object within a container that has not been explicitly registered in a collection. After registration, a user can only create objects under this container if the user has permissions on the collection that owns or references the container object. For example, if a user has an implicit shortcut to Module X (because of several tables under Module X), the user will have no privilege to create objects under Module X. However, if the user has an explicit shortcut to the module, then the user is allowed to create a child object within that module.

The following two files implement this security policy. These files are located on your installation CD under: *samples/security_feature/stewardship.*

stewardship.sql: Contains the DDL to create tables with the same structure as shown in Table 19–6.

stewardship.plb: The security package implementation.

These are sample implementations of the security interface. They can be modified or combined to provide a more sophisticated security policy for your organization. Use these implementations as templates to implement your own security policies.

20 Data Quality: Name and Address Cleansing

This chapter contains additional information about using the Name and Address operator, discussed in Chapter 8, "Using Mapping Operators". This chapter contains the following topics:

Input Roles

Input roles indicate what kind of name or address information resides in a line of data. Each input attribute in the Name and Address operator must have an input role that most closely matches the data contained in the source attribute. Input roles can either be non-discrete, or line oriented, input roles for free-form data (such as 'Line1'); or they can be discrete roles for specific input attributes (such as 'Person', 'First Name', 'Primary Address', or 'City'). Discrete roles give the Name and Address operator more information about the content of the source attribute.

Table 20–1 lists input roles and descriptions for the Name and Address Operator.

Table 20–1 Name and Address Operator Input Roles

Input Role	Description
First Name	First name. This can be a 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 (such as “Herbert” in George Herbert Walker Bush).
Middle Name 2	Second middle name; for example, “Walker” in George Herbert Walker Bush.
Middle Name 3	Third middle name; for example, “Louise” in Ethel May Roberta Louise Mertz.
Last Name	Last name, or surname.

Table 20–1 (Cont.) Name and Address Operator Input Roles

Input Role	Description
First Part Name	First part of the Person name, including: ■ Pre name ■ First name ■ Middle name(s) Use when these components are contained in one source column.
Last Part Name	Last part of Person Name, including: ■ Last name ■ Post Name Use when these components are all contained in one source column.
Pre Name	Information that precedes and qualifies the name; for example, Ms., Mr., or Dr.
Post Name	Generation or other information qualifying the name; for example, Jr. or Ph.D.
Person	Full person name, including: ■ First Part Name (consisting of Pre Name, First Name, and Middle Names) ■ Last Part Name (consisting of Last Name and Post Name) Use when these components are all contained in one source column.
Person 2	Designates a second person if the input includes multiple personal contacts.
Person 3	Designates a third person if the input includes multiple personal contacts.
Firm Name	Name of the company or organization.
Primary Address	Box, route, or street address, including: ■ Street name ■ House number ■ City map grid direction; for example, SW or N ■ Street type; for example, avenue, street, or road. This does not include the Unit Designator or the Unit Number.
Secondary Address	The second part of the street address, including: ■ Unit Designator ■ Unit Number For example, in a secondary address of Suite 2100, the Unit Designator is 'STE' (a standardization of "Suite") and the Unit Number is '2100'.
Address	Full address line, including: ■ Primary Address ■ Secondary Address Use when these components share one column.
Address 2	Generic address line.
Neighborhood	Neighborhood or barrio, common in South and Latin American addresses.
City	Name of city.
State	Name of state or province.

Table 20–1 (Cont.) Name and Address Operator Input Roles

Input Role	Description
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.
Line1, Line2, Line3, Line4, Line5, Line6, Line7, Line8, Line9, Line10	Non-discrete roles intended for use as free form name, business, personal, and address text. These generic selections can store any type of name and address data. They do not provide the parser with any information about the data content. Whenever possible, use the discrete input roles provided instead.

Output Components

Each output component represents a discrete name or address entity, such as a title, a standardized first name, a street number, a street name, or a street type. An output component is assigned to the corresponding output attribute, and identifies the portion of a name or address that the attribute constitutes. Table 20–2 lists the Name and Address operator output components and their descriptions.

Table 20–2 Name and Address Operator Output Components

Parent Node(s)	Output Component	Description
Name	Pre Name[1]	Title or salutation appearing before a name; for example, Ms. or Dr.
Name	First Name Standardized[1]	Standard version of first name; for example, Theodore for Ted or James for Jim.
Name	Middle Name Standardized[1]	Standardized version of the middle name; for example, Theodore for Ted or James for Jim. Used when there is only one middle name, or for the first of several middle names (such as "Herbert" in George Herbert Walker Bush).
Name	Middle Name 2 Standardized[1]	Standardized version of the second middle name; for example, Theodore for Ted or James for Jim.
Name	Middle Name 3 Standardized[1]	Standardized version of the third middle name; for example, Theodore for Ted or James for Jim.
Name	Post Name[1]	Name suffix indicating generation; for example, Sr., Jr., or III.
Name	Other Post Name[1]	Name suffix indicating certification, academic degree, or affiliation; for example, Ph.D., M.D., or R.N.
Name	Name Designator[1]	Personal name designation; for example, "ATTN" (to the attention of) or "C/O" (care of).

Table 20–2 (Cont.) Name and Address Operator Output Components

Parent Node(s)	Output Component	Description
Name	Relationship[1]	Information related to another person; for example, "Trustee for."
Name	Person[1]	First name, middle name, and last name.
Name: Person	First Name[1]	The first name found in the input name.
Name: Person	Middle Name[1]	Middle name or initial. Use this for a single middle name, or for the first of several middle names (such as "Herbert" in George Herbert Walker Bush).
Name: Person	Middle Name 2[1]	Second middle name; for example, "Walker" in George Herbert Walker Bush.
Name: Person	Middle Name 3[1]	Third middle name; for example, "Louise" in Ethel May Roberta Louise Mertz.
Name: Person	Last Name[1]	Last name, or surname.
Name: Derived	Gender[1]	Probable gender: ▪ M= Male ▪ F= Female ▪ N= Neutral (either Male or Female) ▪ Blank=unknown.
Name: 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.
Name: Business	Firm Name[1]	Name of the company or organization, including divisions.
Name: Business	Firm Count[1]	Number of firms referenced in the record.
Address	Address[2]	Full address line, including: ▪ Primary Address ▪ Secondary Address
Address	Primary Address[2]	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.
Address: Primary Address	Street Number[2]	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 is 200.
Address: Primary Address	Pre Directional[2]	Street directional indicator appearing before the street name; for example, in 100 N University Drive, the Pre Directional is 'N'.
Address: Primary Address	Street Name[2]	Name of street.
Address: Primary Address	Street Type[2]	Street identifier; for example, ST, AVE, RD, DR, or HWY.

Table 20–2 (Cont.) Name and Address Operator Output Components

Parent Node(s)	Output Component	Description
Address: Primary Address	Post Directional[2]	Street directional indicator appearing after the street name; for example, in 100 15th Ave. S., the Post Directional is 'S'.
Address	Secondary Address[2]	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: Secondary Address	Unit Designator[2]	Type of secondary address, such as APT or STE. For example, in a secondary address of Suite 2100, the Unit Designator is 'STE' (a standardization of "Suite").
Address: Secondary Address	Unit Number[2]	Number that identifies the secondary address, such as the apartment or suite number. For example, in a secondary address of Suite 2100, the Unit Number is '2100'.
Address	Last Line	Final address line, including: ■ City ■ State or province (if state contains a county instead of a state or province, it is not included) ■ Formatted postal code if address was fully assigned
Address: Last Line	Neighborhood	Neighborhood or barrio, common in South and Latin American addresses.
Address: Last Line	City	Name of city. The US city names may be converted to United States Postal Service preferred names.
Address: Last Line	State	Name of state or province.
Address: Last Line	Postal Code	Full postal code with spaces and other non-alphanumeric characters removed.
Address: Last Line	Postal Code Formatted	Formatted version of postal code that includes spaces and other non-alphanumeric characters, such as dashes.
Address: Last Line	Delivery Point	Applies to 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.
Address: 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.
Address: 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.
Address: Last Line	Country Name	The full country name.

Table 20–2 (Cont.) Name and Address Operator Output Components

Parent Node(s)	Output Component	Description
Address: Other Address Line	Box Name[2]	The name for a post office box address; for example, for "PO Box 95," the Box Name is 'PO BOX'.
Address: Other Address Line	Box Number[2]	The number for a post office box address; for example, for "PO Box 95," the Box Number is '95'.
Address: Other Address Line	Route Name[2]	Route name for a rural route address. For an address of "Route 5 Box 10," the Route Name is 'RTE' (a standardization of "Route").
Address: Other Address Line	Route Number[2]	Route number for a rural route address. For an address of "Route 5 Box 10," the Route Number is '5'.
Address: Other Address Line	Building Name	Building name, such as "Cannon Bridge House." Building names are common in the United Kingdom.
Address: Other Address Line	Complex	Building, campus, or other complex. For example, USS John F. Kennedy, Shadow Green Apartments, Cedarvale Gardens, Concordia College. You can use an the Instance field in the Output Components dialog to specify which complex should be returned in cases where an address has more than one complex.
Address: Other Address Line	Miscellaneous Address	Miscellaneous address information, such as a telephone number or an e-mail address. In records with multiple miscellaneous fields, you can extract several by specifying which instance to use in the Name and Address operator Output Components screen. For example, in a record that has both an e-mail address and a telephone number, you can extract both items of information by using Miscellaneous Address 1 and Miscellaneous Address 2.
Error Status: Name and Address	Is Good Group	Possible values: T or F. Indicates whether the name group, address group, or name and address group was processed successfully: ▪ For name groups, T = The name has been successfully parsed. ▪ For address groups, T = 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, T = 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.
Error Status: Name and Address	Is Parsed	Indicates whether the name or address was parsed: ▪ T= The name or address was parsed successfully. A name or address is considered to be successfully parsed even if some parsing warnings exist. ▪ F= The name or address may not be parsed. Check the status of parsing warning flags (such as Name Warning, or City Warning).

Table 20–2 (Cont.) Name and Address Operator Output Components

Parent Node(s)	Output Component	Description
Error Status: Name and Address	Parse Status	Postal matching software parse status code.
Error Status: Name and Address	Parse Status Description	Text description of the postal matching software parse status.
Error Status: Name Only	Is Good Name	Indicates whether the name was parsed successfully: ■ T= The name was parsed successfully. Note that a name is considered to be successfully parsed even if some parsing warnings exist. ■ F= The name may not be parsed.
Error Status: 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.
Error Checking: 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, this indicates that address was not found in the database. If no postal matching database is available for the country, this indicates that the address may not be parsed. This flag is easier to use if you have a mix of records from both postal matched and non postal matched countries.
Error Checking: 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 can mean that the address is not a legal address, and also that postal matching is not available for the country. This flag is true only if all of the flags listed in the following section are true. If postal matching is available, this flag is the best indicator of record quality.
Error Checking: Address Only: Is Found	City Found	Indicates whether the postal matcher found the city: ■ T= The postal matcher found the city. ■ F= The postal matcher did not find the city.
Error Checking: Address Only: Is Found	Street Name Found	Indicates whether the postal matcher found the street name: ■ T= The postal matcher found the street name. ■ F= The postal matcher did not find the street name.

Table 20–2 (Cont.) Name and Address Operator Output Components

Parent Node(s)	Output Component	Description
Error Checking: Address Only: Is Found	Street Number Found	Indicates whether the postal matcher found the street number within a valid range of numbers for the named street: ■ T= The postal matcher found the street number. ■ F= The postal matcher did not find the street number within a valid range of numbers for the named street.
Error Checking: Address Only: Is Found	Street Components Found	Indicates whether the postal matcher found the street components, such as the Pre Directional or Post Directional: ■ T= The postal matcher found the street components. ■ F= The postal matcher did not find the street components.
Error Checking: 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.
Error Checking: Address Only	City Warning	Indicates whether the parser found unusual or possibly erroneous data in a city: ■ T= The parser had difficulty parsing a city. ■ F= No difficulty parsing city.
Error Checking: Address Only	Street Warning	Indicates whether the parser found unusual or possibly erroneous data in a street address: ■ T= The parser had difficulty parsing a street address. ■ F= No difficulty parsing street address.
Error Checking: Address Only	Is Address Verifiable	Indicates whether postal matching is available for the country of the address, but does not indicate the outcome of the matching operation: ■ T= Postal matching is available for the country address. ■ F= Matching is not available for the country of the address. This indicates that a postal matching database is installed for the country indicated by the address but matching is not available for the address. No postal matching database is installed for the country indicated by the address.

Table 20–2 (Cont.) Name and Address Operator Output Components

Parent Node(s)	Output Component	Description
Error Checking: 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, indicating the component that was changed, must also be True. ■ F= No components of the address were changed, with the possible exception of standardization.
Error Checking: Address Only: Address Corrected	Postal Code Corrected	Indicates whether the postal code was corrected during matching. Correction can include the addition of a postal extension: ■ T= The postal code was corrected. ■ F= The postal code was not corrected.
Error Checking: Address Only: Address Corrected	City Corrected	Indicates whether the city name was corrected during matching. Postal code input is used to determine the city name preferred by the postal service. ■ T= The city name was corrected during matching. ■ F= The city name was not corrected.
Error Checking: Address Only: Address Corrected	Street Corrected	Indicates whether the street name was corrected during matching. Some correct street names may be changed to an alternate name preferred by the postal service. ■ T= The street name was corrected during matching. ■ F= The street name was not corrected.
Error Checking: Address Only: Address Corrected	Street Components Corrected	Indicates whether any of the street components, such as the Pre Directional or Post Directional, were corrected during matching: ■ T= Street components were corrected. ■ F= Street components were not corrected.
Error Checking: Address Only	Address Type	Type of address. The following are common examples; actual values vary with vendors of postal matching software: ■ 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. ■ B= Box ■ R= Rural Code ■ S= Street ■ M= Military ■ P= Post Office Box

Table 20–2 (Cont.) Name and Address Operator Output Components

Parent Node(s)	Output Component	Description
Error Checking: Address Only	Parsing Country	Country parser that was used for the final parse of the record.
Country Specific: United States	ZIP 5	The five-digit United States postal code. This applies to United States addresses only.
Country Specific: United States	ZIP 4	The four-digit suffix that is added to the five-digit United States postal code to further specify location. This applies to United States addresses only.
Country Specific: United States	Urbanization Name	Urban unit name used in Puerto Rico. This applies to United States (Puerto Rico) addresses only.
Country Specific: United States	LACS Flag	Indicates whether the address requires a LACS conversion. 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 addresses, but on occasion they may involve renaming or renumbering existing city-style addresses. ■ T= Address needs 9-1-1 conversion (from old rural route to new street address) and should be submitted to a LACS vendor ■ F= Address does not require conversion. This applies to United States addresses only.
Country Specific: United States	CART	Four-character USPS Carrier route. This applies to United States addresses only.
Country Specific: United States	DPBC Check Digit	Check digit for forming a delivery point bar code. This applies to United States addresses only.
Country Specific: United States: 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. This applies to United States addresses only.
Country Specific: United States: Geography	Minor Census District	Minor Census District. This applies to United States addresses only.
Country Specific: United States: Geography	Latitude	Latitude in degrees north of the equator: positive for north of the equator; negative for south (always positive for North America).
Country Specific: United States: Geography	Longitude	Longitude in degrees east of the Greenwich Meridian: positive for east of GM; negative for west (always negative for North America).
Country Specific: United States: Geography	FIPS County	The three-digit county code as defined by the Federal Information Processing Standard (FIPS). This applies to United States addresses only.
Country Specific: United States: 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. This applies to United States addresses only.

Table 20–2 (Cont.) Name and Address Operator Output Components

Parent Node(s)	Output Component	Description
Country Specific: 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. This applies to United States addresses only.
Country Specific: United Kingdom	Locality Code	This applies to United Kingdom addresses only. For example, the following address is assigned Locality Code 23591: Chobham Rd Knaphill Woking GU21 2TZ
Country Specific: United Kingdom	Locality Name	This applies to United Kingdom addresses only. For example, the following address is assigned Locality Name KNAPHILL: Chobham Rd Knaphill Woking GU21 2TZ
Country Specific: United Kingdom	County Name	This applies to United Kingdom addresses only.
Country Specific: Canada	Installation Type	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'.
Country Specific: Canada	Installation Name	Name of Canadian postal installation. For example, for the address, "PO Box 7010, Scarborough ON M1S 3C6," the Installation Name is 'AGINCOURT'.
Country Specific: Hong Kong	Delivery Office Code	This applies to Hong Kong addresses only. For example, the following address is assigned the Delivery Office Code WCH: Oracle 39/F The Lee Gardens 33 Hysan Ave Causeway Bay
Country Specific: Hong Kong	Delivery Beat Code	This applies to Hong Kong addresses only. For example, the following address is assigned the Delivery Beat Code S06: Oracle 39/F The Lee Gardens 33 Hysan Ave Causeway Bay
Country Specific: Hong Kong	Address 2	The second address line, assigned when both a street address and a building or floor address are present. This applies to Hong Kong only.

[1] In records where this component occurs multiple times, you can specify which instance to use in the Name and Address operator Output Attributes components dialog. For example, in records with two occurrences of Firm Name, you can extract both by adding two output attributes for Firm1 and Firm2, and then assigning a First instance to one, and then a Second instance to the other, from the Instances drop list in the components dialog.

[2] 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. You can specify this in the Name and Address operator Output Attributes components dialog.

Name and Address Operator Supported Countries

Table 20–3 lists supported countries for some vendors of name and address cleansing software. This list varies by vendor.

Table 20–3 Countries Supported by the Name and Address Operator

Name	Postal Code Matching Software Available for Use with the Name and Address Operator
Argentina	No
Australia	Yes
Belgium	No
Brazil	Yes
Canada	Yes
Chile	No
Colombia	No
Denmark	No
France	Yes
Germany	Yes
Hong Kong	Yes
India	No
Ireland	No
Italy	No
Mexico	Yes
Malaysia	No
Netherlands	Yes
New Zealand	No
Peru	No
Philippines	No
Portugal	Yes
Singapore	Yes
South Africa	No
Spain	Yes
Sweden	No
Switzerland	No
United Arab Emirates	No
United Kingdom	Great Britain Only
United States	Yes
Venezuela	No

Country Postal Certifications

The postal matching software used by Oracle Warehouse Builder Name and Address can be certified for various countries. The certification depends on the vendor of the postal matching software you use. Possible certifications include:

- United States - CASS certification with US Postal Service
- Canada - SERP certification with Canada Post
- Australia - AMAS certification with Australia Post

United States Postal Service CASS Certification

The Coding Accuracy Support System (CASS) is a process developed by the United States Postal Service (USPS) in cooperation with the mailing industry. The process 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. Oracle9i Pure Name and Address is CASS-certified. The CASS report is a text file specified by the USPS and produced by Oracle9i Pure Name and Address. To meet USPS requirements, the mailer must submit a CASS report in its original form to the USPS.

Canada Post SERP Certification

Canada Post has 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.

Customers who utilize Incentive Lettermail, Addressed Admail and Publications Mail must meet the Address Accuracy Program requirements. Customers can obtain a Statement of Accuracy by comparing their database to Canada Post's address data.

Australia Post AMAS Certification

The Address Matching Approval System (AMAS®) is a software approval program that has been 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 the barcoding of mail.

AMAS allows companies to develop address matching software which:

- Prepares addresses for barcode creation
- Ensures quality addressing
- Enables qualification for discounts on PreSort letter lodgements

PreSort Letter Service prices are conditional upon customers using AMAS Approved Software with Delivery Point Identifiers (DPIDs) being current against the latest version of the PAF.

A declaration that your mail was prepared appropriately must be made when using the Presort Lodgement Document, available from Post Offices.

You can only produce one postal report for each mapping. The postal report created is for the primary country specified in the Name and Address operator definition. Ideally, all mapped addresses should be located in the primary country selected in the Definitions page of the Name and Address Wizard. For more information, see "Definitions" on page 8-58.

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 under the `ORACLE_HOME` of your Oracle Warehouse Builder Server Side Install at `owb/bin/admin`. The following code illustrates several important properties with their default settings.

```
TraceLevel=0
SocketTimeout=120
ClientThreads=16
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 parse 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.
- **ClientThreads:** This parameter is reserved for future development. Keep the default value of 16.
- **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. You must stop and restart the server manually if the auto shutdown property is `FALSE` and the server is running when you edit the file or perform the table maintenance.

To manually stop the Name and Addresss Server:

- In Windows, run *`ORACLE_HOME`*`/owb/bin/win32/NAStop.bat`.
- In UNIX, run *`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 `ORACLE_HOME/owb/bin/win32/NAStart.bat`.
- In UNIX, run `ORACLE_HOME/owb/bin/unix/NAStart.sh`.

Best Practices for Using the Name and Address Operator in a Mapping

Use the following best practice tips to define and use the Name and Address Operator in your mapping.

Correcting Errors in Name and Address Data

Oracle Warehouse Builder Name and Address is built on name and address software and data supplied by third-party software vendors who specialize in name and address cleansing. This section discusses additional considerations that pertain to these products, as well as general parsing issues.

Name and Address parsing and correction can provide great improvements in data quality, but can degrade quality if not applied carefully. Name and Address parsing, postal matching, or any cleansing of incorrect data is error-prone because the problem domain of dirty data is not bounded.

Name and Address parsing, like any other type of parsing, depends on identification of keywords and patterns containing those keywords. Unlike computer languages, which contain a small set of keywords, free form Name and Address data of any region is very difficult to parse because the keyword set is so large and the set is never 100% complete. For example, one name and address cleansing software provider's pattern table for USA contains over 80,000 defined keywords and over 5,000 patterns. 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 record lines contain common patterns of numbers, single letters, and alphanumeric strings, parsing can often be performed based on just the alphanumeric patterns. In more difficult cases, alphanumeric patterns may be ambiguous, with possible interpretation as either an address line or name line. In other cases, a particular pattern may not be found. With computer languages, compilation errors are reported and an executable image is not produced; with Name and Address parsing errors, parsing status codes are set and Name and Address elements may not be parsed.

Using Status Codes

Because failure of the parsing engine at some point is highly probable, perform all Name and Address processing with error processing in mind. Use status codes to control the data mapping. Since the criteria for quality vary between applications, numerous flags are available to help you determine the quality of a particular record. For countries with postal matching support, the Is Good Group flag is the best measure of quality because it verifies that an address is a valid entry in a postal database. For CASS or SERP certified mailings, this flag may be the best criterion for acceptance or rejection of a record.

If you do not perform postal reporting, an address does not have to be found in a postal database to be acceptable. For example, it may not be possible to locate street intersection addresses or addresses using building names may not be locatable in a postal database, but they may still be deliverable. If the parse status is good, some benefit will be derived from standardization of address elements. If the Is Good Group flag indicates postal matching failure, several parser warning/error flags are also available to 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. If parser warnings are present, it may be desirable to check those records manually. If Is Parsed indicates parsing failure, preservation of your original data is necessary to prevent data loss.

Using Discrete Input Roles

A wide range of input roles is available to match the granularity of input data. Line based input roles are available for line oriented input, and discrete roles are available for atomic attributes such as First Name. Whenever possible, use discrete roles because they give the parser more information about the data content, and result in better parsing. Some of the discrete input roles overlap less discrete roles; for example, the combination of city, state, and postal code redefines the Last Line role. Such redefining assignments should be avoided. Where they occur, the more discrete roles take precedence because they are more specific.

Map discrete Name and Address attributes whose original value you want to retain directly to the target (map them around the Name and Address operator). This is very common for first, middle, and last names because it is not a good practice to change a first name to a standardized first name (for example, changing Peggy to Margaret). This direct mapping prevents losing name data if a personal name is parsed as a firm name. To obtain a standardized first or middle name for matching and merging (data deduplication), you can map the same attributes into the Name and Address operator. However, only the recoded outputs, such as title, gender, or standardized name, are used as output from the operator. There is no advantage to using first, middle, or last name output because it mirrors the input. This recommendation only applies to discrete data where the first, middle, and last name is already known.

Separating Good Records from Bad Records with the Splitter Operator

Consider using the splitter operator to map good records to one target and bad records to another target. If the percentage of bad records is very low, or if the data set is small, you may consider manual correction of bad records. You can flag each record manually corrected record for later mapping to the target schema. If the IS_GOOD_GRP flag is F and the records are mapped to a special target for manual correction, you can map additional components that indicate the success level of the postal match. Components are available to indicate matching success at the city, street name, street number range, or street component (for example, the pre-directional, post-directional, or street type) level.

21 Using SAP R/3 Data in Warehouse Builder

The Warehouse Builder SAP Integrator enables you to import metadata object definitions from SAP Application data sources into the Warehouse Builder repository. This chapter describes how to use SAP objects in a mapping, generate PL/SQL and ABAP code for the mappings, and deploy them to a target. This chapter also describes how to extract and load SAP data into your target.

This chapter contains the following topics:

About the Warehouse Builder SAP Integrator

The Warehouse Builder SAP Integrator enables you to connect to SAP application source systems and import the SAP source definitions into a project in the Warehouse Builder repository.

You can then generate ABAP or PL/SQL code to extract, transform, and load data from SAP R/3 3.x and R/3 4.x systems to your target system.

About SAP Business Areas

SAP application systems logically group database and metadata objects under different business areas. In SAP, a business area is an organizational unit in an enterprise that groups product and market areas. For example, the Financial Accounting (FI) business area 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 into Warehouse Builder, you can use a graphical navigation tree in the Business Component Hierarchy dialog to search the business area structure in the SAP source application. This navigation tree enables you to select SAP metadata objects from the SAP application server.

SAP Table Types

The SAP integrator enables you to import metadata for SAP Business Areas or any of their related ABAP Dictionary objects.

With the SAP integrator, you can import definitions and generate deployment code for the following SAP table types:

- **Transparent:** A transparent table is first defined in the ABAP Dictionary and then created in the database. You can also use transparent tables independently of the R/3 System. You can generate either PL/SQL or ABAP code for transparent tables.
- **Cluster:** A cluster table is an ABAP Dictionary table type. It contains information pertaining to any group of database tables and it is not created in the SAP database. Because cluster tables are data dictionary tables and not database tables, you can only generate ABAP code.
- **Pooled:** The data from several tables is stored together as a table pool in the database. Pooled tables exist in the ABAP Dictionary and are not known to the database. You can only generate ABAP code for pooled tables.

Required Files For Windows

The Warehouse Builder SAP Integrator requires a dynamic link library file named `librfc32.dll` in order to use remote function calls on the client machine. This file is available on the SAP Application Installation CD. You need to copy this file to the following Warehouse Builder directory on your client system:

```
X:\ORACLE_HOME\bin\admin
```

where "`X:`" is the drive containing the Warehouse Builder client and "`\ORACLE_HOME`" is the Oracle home path name for Warehouse Builder.

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. Check the version or build number of your `.dll` file from your NT Explorer window.

The following version is currently supported in Warehouse Builder:

File Version: 4640,5,123,2956

Build: Wednesday, August 09 23:46:33 2000

File Size: 1,945,138 bytes

Product Version: 46D,123

You can locate this version of the `.dll` file on the Installation CD.

Required Files For Unix

The Warehouse Builder SAP Integrator requires a dynamic link library file named `librfccm.so` in order to use remote function calls on the client machine. This file is available on the SAP Application Installation CD. You need to copy this file to the following Warehouse Builder directory on your client system:

```
X:\ORACLE_HOME\bin\admin
```

where "`X:`" is the drive containing the Warehouse Builder client and "`\ORACLE_HOME`" is the Oracle home path name for Warehouse Builder.

You also need to add `X:\ORACLE_HOME\bin\admin` to the Unix environment variable path: LD_LIBRARY_PATH.

Defining SAP Metadata Objects

The New Module Wizard enables you to define an SAP source module. Warehouse Builder uses this definition to store imported metadata from your source SAP application.

This section contains the following topics:

- Creating SAP Module Definitions on page 21-3
- Importing SAP Metadata Definitions on page 21-7
- Updating SAP Source Modules on page 21-13

For related information, see:

- Configuring Connections for Database Sources on page 4-3
- Chapter 4, "Importing Data Definitions"
- SAP Application 3.0 documentation

Creating SAP Module Definitions

Using the New Module Wizard, you can choose SAP R/3 version 3.x or SAP R/3 version 4.x system types as your source. After you select the application version, you need to set the connection information between the Warehouse Builder repository and the SAP application server. You can edit this information using the module property sheet.

Note: To create SAP Module definitions, you must first obtain the connection information to your SAP Application server from your system administrator.

When you set the connection information, you can choose the following connection types:

- Remote Function Call (RFC)

  This is the default connection type. A remote function call locates a function module running in a system different from that of the caller. The remote function can also be called from within the same system (as a remote call), but usually the caller and the called are located in different systems. This method requires specific IP Address information for the SAP application server.

- SAP Remote Function Call (SAPRFC.INI)

  SAP can use its own initialization file to track the IP Address information for you. The SAPRFC.INI enables remote calls between two SAP Systems (R/3 or R/4), or between an SAP System and a non-SAP System. This method is useful when you know the SAP-specific connection information and want to automate the IP connection information.

Note: To use the SAPRFC.INI connection type, the file SAPRFC.INI must be installed in this directory:

```
X:\ORACLE_HOME\wbapp
```

where `X:\ORACLE_HOME` is the Warehouse Builder Oracle home path. This file is available in the SAP Application client installation CD. Consult your system administrator for more information.

The New Module Wizard creates the module for you based upon the metadata contained in the SAP application server.

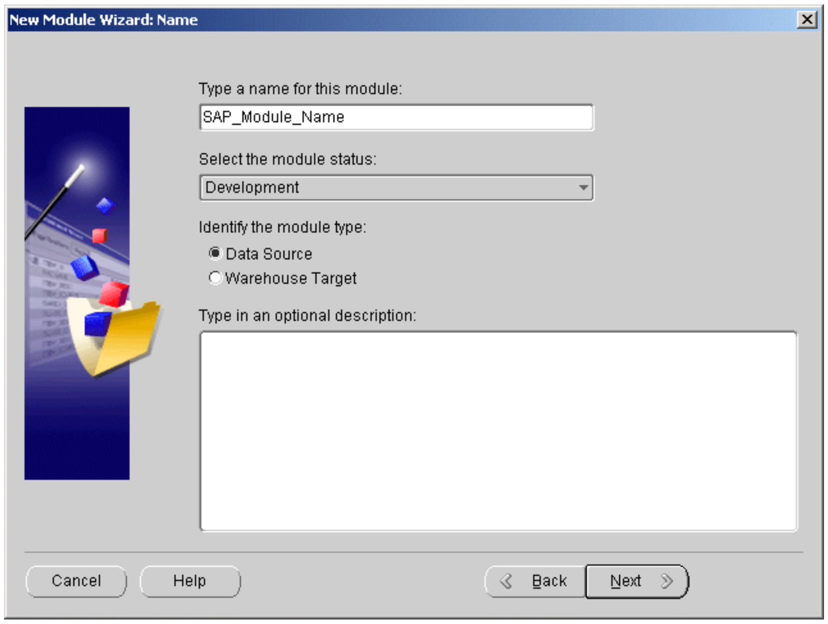
To create an SAP source module:

1. From the Warehouse Builder console navigation tree, expand the Applications node.
2. Right-click the SAP node and select **Create SAP R3 Source Module.**

   Warehouse Builder displays the Welcome page for the New Module Wizard.
3. Click **Next.**

   The wizard displays the Name page as shown in Figure 21–1.

Figure 21–1 New Module Import Wizard Name Page


New Module Wizard: Name

Type a name for this module: SAP_Module_Name

Select the module status: Development

Identify the module type:
(●) Data Source
( ) Warehouse Target

Type in an optional description:

4. Provide the following information in the Name page:

   Name of the module: Type a unique name for the module between 1 and 30 alphanumeric characters. Spaces are not allowed.

   Status of the module: Select a status for the module from the drop-down list: Development, Quality Assurance, Production.

   Selecting one of these options can help you document the warehouse design version.

   Description: Type a description of the module you are creating (Optional).

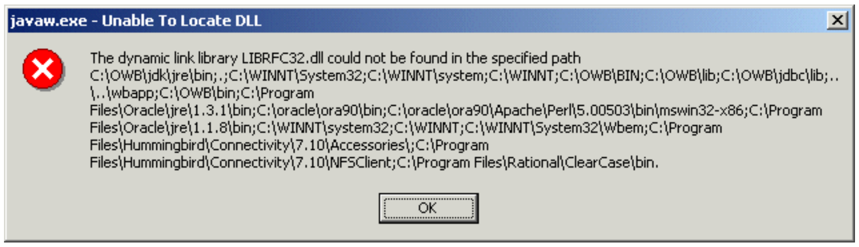
5. Click **Next**.

   The wizard displays the Data Source Information Page.

6. Select the correct version of your SAP application from the drop-down list: SAP R/3 3.x or SAP R/3 4.x.

   If the `librfc32.dll` file is missing, an error message displays, as shown in Figure 21–2.

Figure 21–2 Error Message for Missing librfc32.dll File



javaw.exe - Unable To Locate DLL

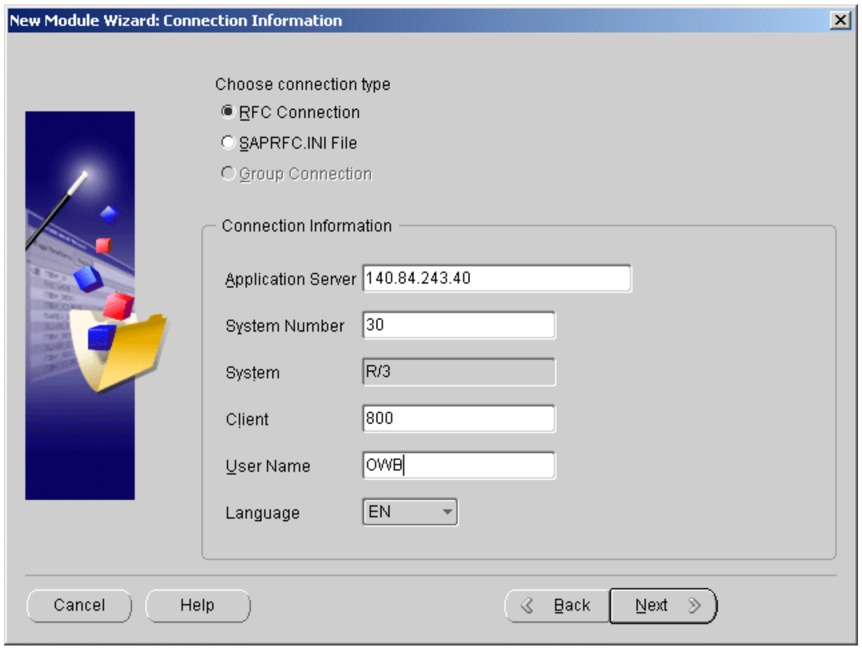
```
The dynamic link library LIBRFC32.dll could not be found in the specified path
C:\OWB\jdk\jre\bin;.;C:\WINNT\System32;C:\WINNT\system;C:\WINNT;C:\OWB\BIN;C:\OWB\lib;C:\OWB\jdbc\lib;..
\..\wbapp;C:\OWB\bin;C:\Program
Files\Oracle\jre\1.3.1\bin;C:\oracle\ora90\bin;C:\oracle\ora90\Apache\Perl\5.00503\bin\mswin32-x86;C:\Program
Files\Oracle\jre\1.1.8\bin;C:\WINNT\system32;C:\WINNT;C:\WINNT\System32\Wbem;C:\Program
Files\Hummingbird\Connectivity\7.10\Accessories\;C:\Program
Files\Hummingbird\Connectivity\7.10\NFSClient;C:\Program Files\Rational\ClearCase\bin.
```

   You must load the `librfc32.dll` file before you can proceed. For more information, see "Required Files For Windows" on page 21-2 and "Required Files For Unix" on page 21-2.

7. Click **Next.**

   The wizard displays the Connection Information page for an RFC Connection, as shown in Figure 21–3.

Figure 21–3 New Module Wizard Connection Information Page



New Module Wizard: Connection Information

Choose connection type

- (●) RFC Connection
- ( ) SAPRFC.INI File
- ( ) Group Connection

Connection Information

Application Server	140.84.243.40
System Number	30
System	R/3
Client	800
User Name	OWB
Language	EN

   Select one of the following connection types:

   Remote Function Call (RFC) is the default connection type. For more information, see Remote Function Call (RFC) on page 21-3.

SAP Remote Function Call (SAPRFC.INI). For more information, see SAP Remote Function Call (SAPRFC.INI) on page 21-3.

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

   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, as shown in Figure 21–4:

   Application Server: Type the alias name or the IP address of the SAP application server.

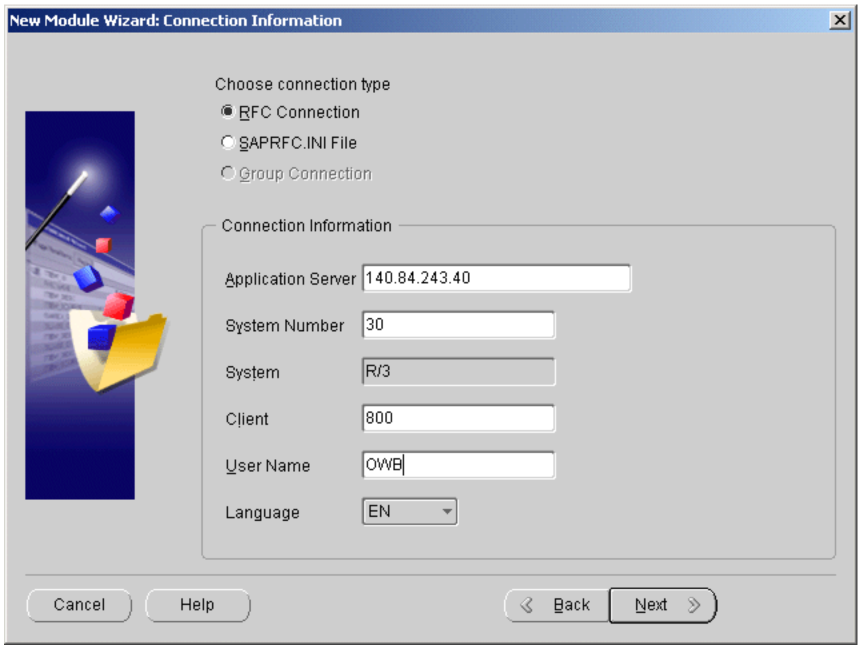
   System Number: Type the SAP system number for SAP user interface login. This number is required in the SAP application configuration and is supplied by the SAP system administrator.

   Client: Type the SAP client number. This number is required in the SAP application configuration and is supplied by the SAP system administrator.

   User Name: Type the user name for the SAP user interface. This name is required in the SAP application configuration and 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.

Figure 21–4 New Module Wizard Connection Information Page (Default RFC)


New Module Wizard: Connection Information

Choose connection type
(● RFC Connection) ○ SAPRFC.INI File ○ Group Connection

Connection Information

Application Server: 140.84.243.40
System Number: 30
System: R/3
Client: 800
User Name: OWB
Language: EN

SAPRFC.INI File connection type requires the following connection information:

RFC Destination: Type the alias for the SAP connection information.

Client: Type the SAP client number.

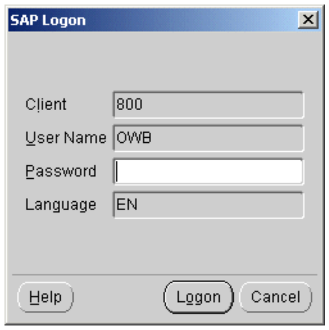
User Name: Type the SAP user name for the SAP user interface.

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.

9. Click **Next.**

   The wizard displays the Logon dialog for the SAP application, as shown in Figure 21–5.

Figure 21–5 SAP Logon Dialog


SAP Logon

Client: 800
User Name: OWB
Password:
Language: EN

10. Type the user name and password for SAP user interface and click **Logon.**

    If the SAP application version number you provide on the Data Source Information page does not match the application version for the source system, an error message displays. Click **Back** to return to the Data Source Information page and correct the application version.

    If no error message appears, the wizard logs onto the SAP application and displays the Finish page.

    To proceed directly to the Import Metadata Wizard, check the box at the bottom of the page. You can also choose to import metadata at a later time and leave the box unchecked.

11. Click **Finish.**

    The wizard creates the new SAP source module and inserts its name under the Applications node in the project navigation tree.

Importing SAP Metadata Definitions

After creating the SAP source module, you can import metadata definitions from SAP tables using the Import Metadata Wizard. This wizard enables you to filter the SAP objects you want to import, verify those objects, and re-import them. You can import metadata for transparent tables, cluster tables, or pool tables.

This section contains the following topics:

- Opening the Import Metadata Wizard on page 21-7
- Filtering SAP Metadata on page 21-8
- Re-importing SAP Objects on page 21-13

Opening the Import Metadata Wizard

If you choose to proceed to the Import Metadata Wizard wizard directly from the New Module Wizard Finish page, then the wizard opens its welcome page. If choose to import metadata at a later time, then you must open the wizard from the SAP module that you created.

To open the Import Metadata Wizard:

1. From the Warehouse Builder navigation tree, expand the **Applications** node and then the **SAP** node.
2. Right-click the SAP source module into which you want to import metadata and select **Import** from the pop-up menu.

   Warehouse Builder displays the welcome page for the Import Metadata Wizard.
3. Click **Next.**

Filtering SAP Metadata

The Import Metadata Wizard includes a filter page that enables you to select the metadata. This page offers two filtering methods:

- Business Component

  This method opens the Business Component Hierarchy tree. Browse the SAP business areas to locate the metadata you want to import. You can view a list of tables contained in the Business Area and the names of the tables in the SAP application.
- Text String Matching

  Search for tables by typing text string information in fields provided in the Filter Information page. This is a more direct search method if you are familiar with the contents of your SAP application database.

To filter SAP metadata by Business Component:

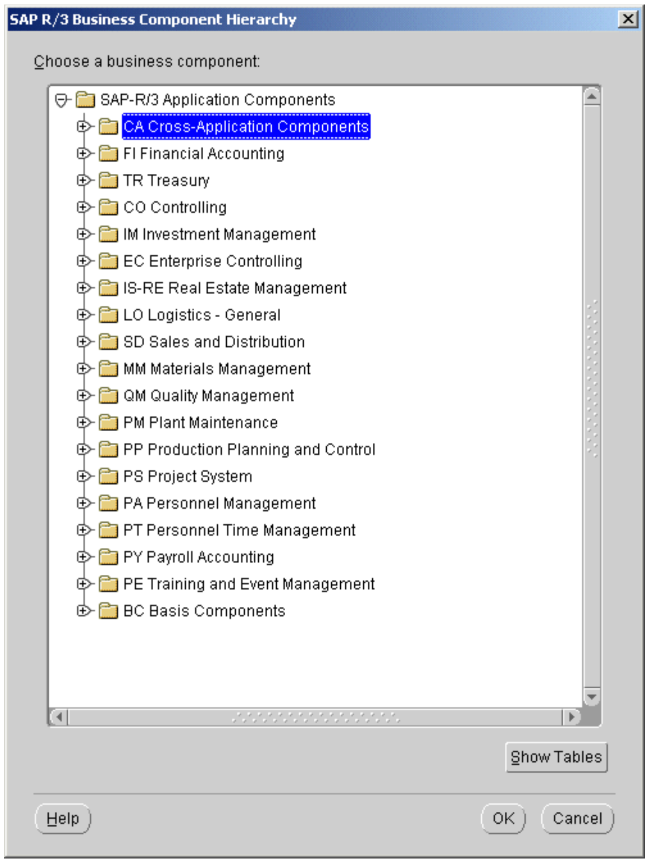
1. Select Business Component and click **Browse** to display the SAP R/3 Business Component Hierarchy dialog.

   It may take two to ten minutes to open this dialog depending upon the network location of the SAP application server, the type of LAN used, or the size of the SAP application database.

   The Import Metadata Wizard displays the Loading business component tree dialog.
2. Click **OK.**

   The wizard displays the Business Component Hierarchy dialog as shown in Figure 21–6.

Figure 21–6 Business Component Hierarchy Dialog

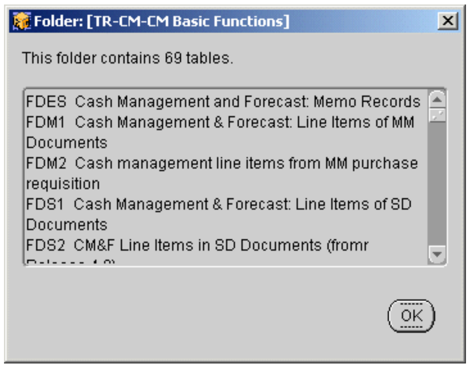

SAP R/3 Business Component Hierarchy
Choose a business component:
SAP-R/3 Application Components
CA Cross-Application Components
FI Financial Accounting
TR Treasury
CO Controlling
IM Investment Management
EC Enterprise Controlling
IS-RE Real Estate Management
LO Logistics - General
SD Sales and Distribution
MM Materials Management
QM Quality Management
PM Plant Maintenance
PP Production Planning and Control
PS Project System
PA Personnel Management
PT Personnel Time Management
PY Payroll Accounting
PE Training and Event Management
BC Basis Components
Show Tables
Help
OK
Cancel

Use this dialog to select the SAP Business Areas containing the metadata objects that you want to import.

3. Select a folder and click **Show Tables** to view the tables available in a business component.

   The Import Wizard displays a list of tables in the selected business component in the Folder dialog as shown in Figure 21–7.

Figure 21–7 Folder Dialog



This folder contains 69 tables.

```
FDES  Cash Management and Forecast: Memo Records
FDM1  Cash Management & Forecast: Line Items of MM
Documents
FDM2  Cash management line items from MM purchase
requisition
FDS1  Cash Management & Forecast: Line Items of SD
Documents
FDS2  CM&F Line Items in SD Documents (fromr
```

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

Some Business Components can contain more than 1000 tables. Importing such a large amount of metadata can take from one to three hours or more, depending upon the network connection speed and the processing power of the source and target systems.

4. Click **OK.**

   The wizard displays the Filter Information page with the SAP Business Area displayed in the Business Component field.

5. Select **Text String where object** and then choose the Name matches entry field or the Description matches entry field to type a string and obtain matching tables from the SAP data source.

   - Although the Name matches field is not case sensitive, the Description matches field is case sensitive.
   - You must type a text string in the selected Text String entry field. It cannot be left empty.
   - Create a filter for object selection by using the wildcard characters (%) for zero or more matching characters, and (_) for a single matching character.

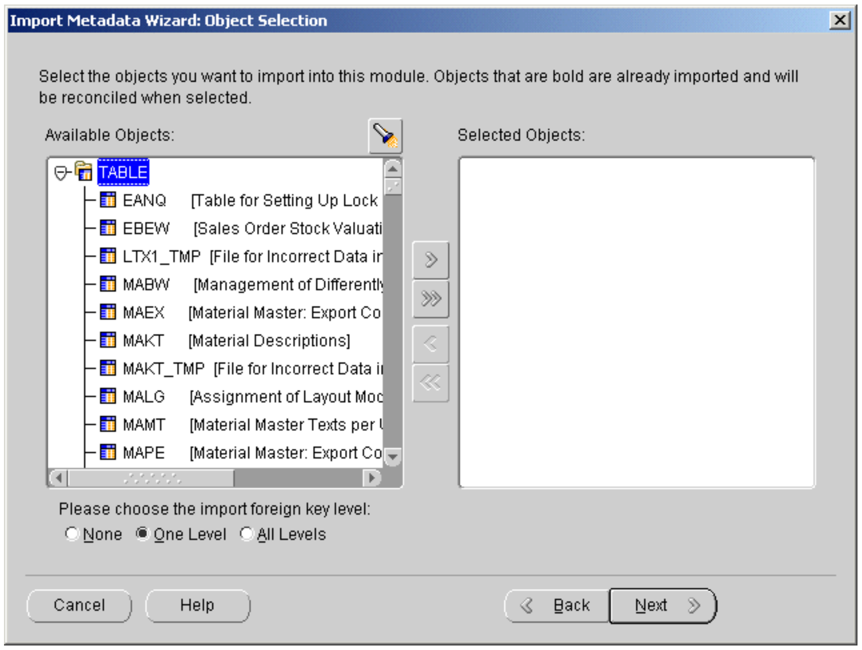
     For example, if you want to search the business area named A0 for tables whose names contain the word CURRENCY, then type `A0%CURRENCY%`. If you want to refine the search to include only tables named CURRENCY and followed by a single digit, then type `A0%CURRENCY_`.

6. Specify the number of tables you want to import in the Maximum number of objects displayed field.

7. Click **Next.**

   The wizard displays the Object Selection page with a description for each table, as shown in Figure 21–8.

Figure 21–8 Import Metadata Wizard Object Selection Page



Import Metadata Wizard: Object Selection

Select the objects you want to import into this module. Objects that are bold are already imported and will be reconciled when selected.

Available Objects:

TABLE
- EANQ [Table for Setting Up Lock
- EBEW [Sales Order Stock Valuati
- LTX1_TMP [File for Incorrect Data ir
- MABW [Management of Differently
- MAEX [Material Master: Export Co
- MAKT [Material Descriptions]
- MAKT_TMP [File for Incorrect Data i
- MALG [Assignment of Layout Moc
- MAMT [Material Master Texts per l
- MAPE [Material Master: Export Co

Selected Objects:

Please choose the import foreign key level:

None [✓ One Level] All Levels

The Import Wizard also enables you to choose whether you want to import tables with foreign key relationships for each table that you choose to import.

8. Select tables from the **Available Objects** list and move them to the **Selected Objects** list using the arrow buttons.

   The SAP Integrator only imports definitions for tables. The tables appear in the Selected Objects list in the Object Selection page**.**

9. Select the Import Foreign Key Level:

   None: Import only the objects in the Selected Objects list.

   One Level: Import the objects in the Selected Objects list and any tables linked to it directly through a foreign key relationship.

   All Levels: Import the objects in the Selected Objects list and all tables linked to it through foreign key relationships.

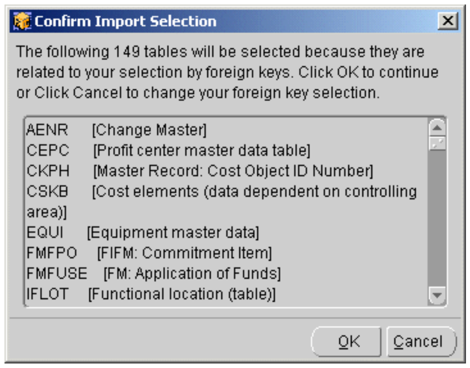
   The foreign key level you select is the same for all tables selected for importing.

   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 for your situation.

10. Click **Next.**

    If the radio button on the Object Selection page for the foreign key level is set to One Level or All Levels, the Confirm Import Selection dialog appears as shown in Figure 21–9.

Figure 21–9 Confirm Import Selection Dialog



The following 149 tables will be selected because they are related to your selection by foreign keys. Click OK to continue or Click Cancel to change your foreign key selection.

```
AENR     [Change Master]
CEPC     [Profit center master data table]
CKPH     [Master Record: Cost Object ID Number]
CSKB     [Cost elements (data dependent on controlling
area)]
EQUI     [Equipment master data]
FMFPO    [FIFM: Commitment Item]
FMFUSE   [FM: Application of Funds]
IFLOT    [Functional location (table)]
```

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

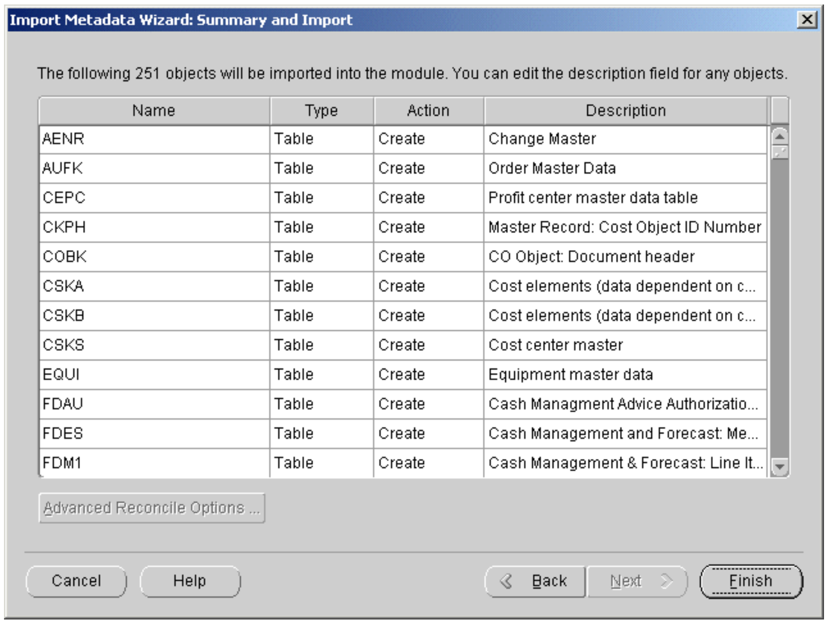
11. Click **OK.**

    The selected objects appear in the right pane of the Object Selection page.

12. Click **Next.**

    The wizard imports 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 as shown in Figure 21–10.

Figure 21–10 Import Metadata Wizard Summary and Import Page



Import Metadata Wizard: Summary and Import

The following 251 objects will be imported into the module. You can edit the description field for any objects.

Name	Type	Action	Description
AENR	Table	Create	Change Master
AUFK	Table	Create	Order Master Data
CEPC	Table	Create	Profit center master data table
CKPH	Table	Create	Master Record: Cost Object ID Number
COBK	Table	Create	CO Object: Document header
CSKA	Table	Create	Cost elements (data dependent on c...
CSKB	Table	Create	Cost elements (data dependent on c...
CSKS	Table	Create	Cost center master
EQUI	Table	Create	Equipment master data
FDAU	Table	Create	Cash Managment Advice Authorizatio...
FDES	Table	Create	Cash Management and Forecast: Me...
FDM1	Table	Create	Cash Management & Forecast: Line It...

Review the information on the Summary and Import page.

You can edit the descriptions for each table by selecting the description field and typing a new description.

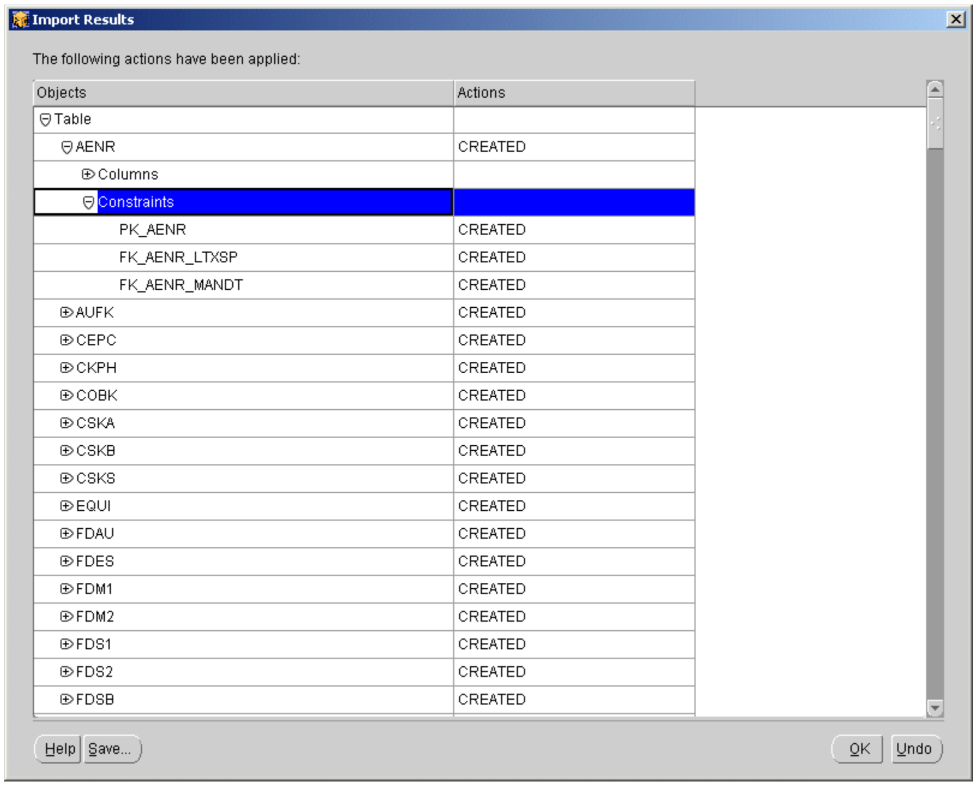
13. Click **Finish.**

The SAP integrator reads the table definitions from the SAP application server and creates the metadata objects in the Warehouse Builder repository.

The time it takes to import the SAP metadata to the Warehouse Builder repository depends on the size and number of tables and the connection between the SAP application server and the repository. Importing 500 or more objects could take one to three hours or more, especially if you are connecting servers in separate LANs.

When the Import completes, the Import Results dialog displays as shown in Figure 21–11.

Figure 21–11 Import Results Dialog



The following actions have been applied:

Objects	Actions
⊖Table	
⊖AENR	CREATED
⊕Columns	
⊖Constraints	
PK_AENR	CREATED
FK_AENR_LTXSP	CREATED
FK_AENR_MANDT	CREATED
⊕AUFK	CREATED
⊕CEPC	CREATED
⊕CKPH	CREATED
⊕COBK	CREATED
⊕CSKA	CREATED
⊕CSKB	CREATED
⊕CSKS	CREATED
⊕EQUI	CREATED
⊕FDAU	CREATED
⊕FDES	CREATED
⊕FDM1	CREATED
⊕FDM2	CREATED
⊕FDS1	CREATED
⊕FDS2	CREATED
⊕FDSB	CREATED

14. Click **OK.**

Re-importing SAP Objects

To re-import SAP objects, you 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. Re-imported table names appear in bold in the summary and import page. The wizard then activates the Advanced Reconcile Options button so that you can control the re-import options.

Updating SAP Source Modules

You must update existing SAP source module definitions whenever you upgrade SAP application versions, migrate SAP servers, and change network connection configurations. You also need to check this information when you re-import metadata.

You can update an SAP module by editing its properties using the Module Properties dialog.

To update SAP object definition:

1. From the Warehouse Builder navigation tree, expand the Applications node and then the SAP node.
2. Right-click the SAP source object and select **Properties** from the pop-up list.

   The Module Properties dialog displays.
3. Select the appropriate tab to edit the SAP object properties.

   Name: Use this tab to modify the unique identification of the module. Use the Name tab to:

Edit the name of the SAP object. The name assigns a unique value to an object and contains between 1 and 30 alphanumeric characters. The 30 character limit applies only to Physical Name Mode.

If necessary, change the status of the SAP object. Select Development, Quality Assurance, or Production.

Edit the description of the SAP object. This field can contain up to 4000 characters. For example, you can use this field to note the purpose of the module and how it relates to the information required by the end-users of the project.

Source: This tab is read-only. Use this page to view the following information:

Application Type: Indicates the application type corresponding to the data source. For example, SAP R/3 4.x.

System Type: Indicates the type of database system (SAP Application) used to host the source data.

Integrator used to access the data source: Displays the Warehouse Builder integrator that was used to connect to the source data.

Connection: Edit the information on this tab only if the network connection information for the SAP application server has changed.

Defining the ETL Process for SAP Objects

After you define the SAP source module and import the metadata, you can define the ETL mappings to extract and load the data from your SAP source to the target. The Warehouse Builder SAP Integrator features a special mapping tool for SAP objects. Warehouse Builder enables you to configure mappings to generate ABAP or PL/SQL code to deploy your metadata.

This section contains the following topics:

- Defining Mappings Containing SAP Objects on page 21-14
- Configuring Code Generation for SAP Objects on page 21-18
- Generating SAP Definitions on page 21-21

Defining Mappings Containing SAP Objects

You can use the Mapping Editor to define mappings for SAP sources. While SAP mappings are similar to other types of mappings, there are some important differences:

- The SAP Integrator in Warehouse Builder enables you to use the Copy and Map Wizard to define a target operator. This wizard is only available for SAP transparent tables.
- Only the Table, Cube, Dimension, Filter, and Joiner mapping operators are available for SAP objects.

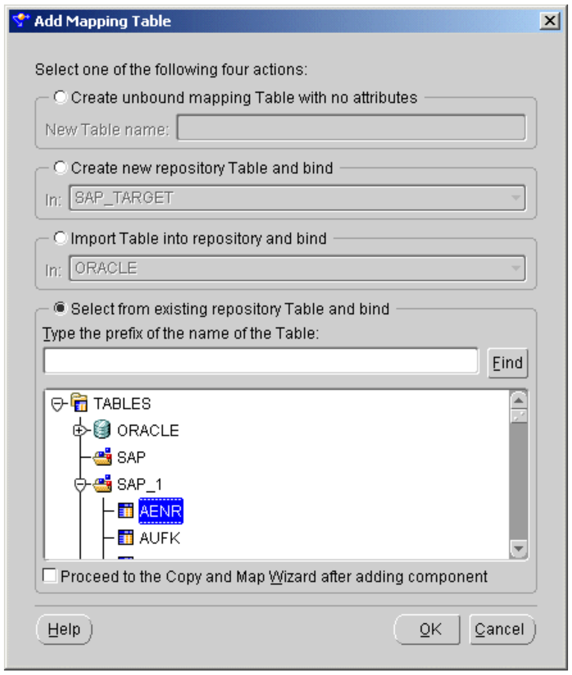
Adding SAP Objects to a Mapping

To add an SAP object to a mapping:

1. From the toolbox, drop the **Mapping Table** icon onto the Mapping Editor canvas.

   The Add Mapping Table dialog displays as shown in Figure 21–12.

Figure 21–12 Add Mapping Table Dialog


Add Mapping Table

Select one of the following four actions:

( ) Create unbound mapping Table with no attributes

New Table name:

( ) Create new repository Table and bind

In: SAP_TARGET

( ) Import Table into repository and bind

In: ORACLE

(●) Select from existing repository Table and bind

Type the prefix of the name of the Table:

[ ] Proceed to the Copy and Map Wizard after adding component

2. Choose **Select from existing repository tables and bind.**

   The field at the bottom of the dialog displays a list of SAP tables whose definitions were previously imported into the SAP source module.

3. Select a source table name and click **OK.**

   The editor places a mapping table on the mapping canvas to represent the SAP table.

   You can choose to use the Copy and Map wizard to define a mapping target. Or you can define it as you would with any other type of mapping operator.

Using the Copy and Map Wizard To Define a Target

When you select an SAP operator as a source, the SAP integrator enables an option to **use the Copy and Map wizard** to define the target object to which you want to map from an SAP source. This wizard enables you to create a target object whose attributes are identical to those of the SAP source. The Copy and Map wizard enables you to perform three functions:

- Copy the selected source object
- Create a target object based on the source object definitions
- Map the source object to the new target object.

You can use the Copy and Map wizard only when you select a single table as a source. If you select more than one table, then you must define the mapping target manually.

To use the Copy and Map Wizard:

1. In the **Add Mapping Table** dialog, select the SAP table you want to use as the source.

2. Check the option to **Proceed to the Copy and Map wizard after adding component.**

3. Click **OK.**

   The **Copy and Map Wizard welcome** page displays.

4. Click **Next.**

   The wizard displays the **Target Name** page.

5. Type the following information:

   Physical Name: A unique physical name for the target object you are creating. The name can contain 1 to 30 characters, no spaces are allowed.

   Object Type: Only tables that map from SAP source tables can be created. The choice in this field is preselected and is read-only.

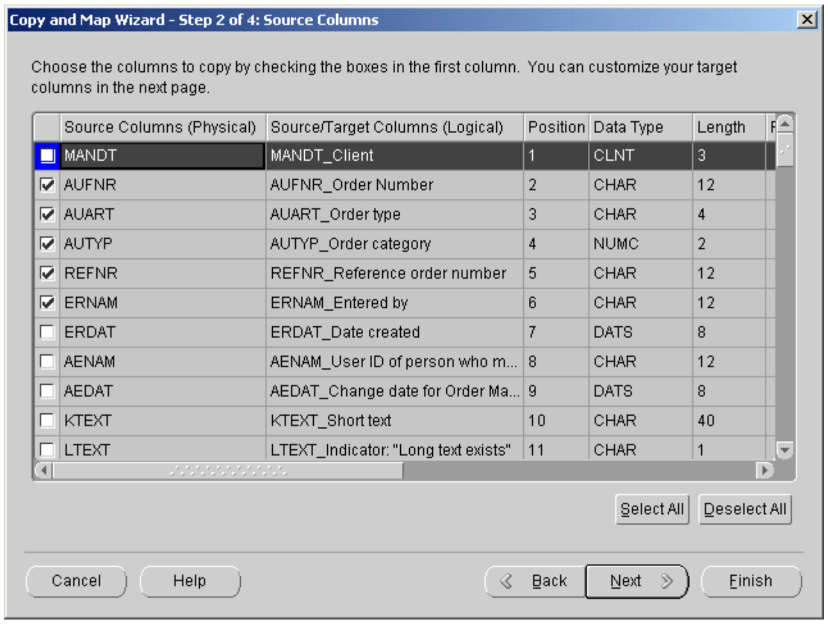
   Application: The target application that contains the target object. The choice in this field is preselected and is read-only.

   Description: Optional field used to describe the target object you are creating. This field can contain file information up to 2 MB in size.

6. Click **Next.**

   The **Source Columns** page displays as shown in Figure 21–13.

Figure 21–13 Source Columns Page



Choose the columns to copy by checking the boxes in the first column. You can customize your target columns in the next page.

	Source Columns (Physical)	Source/Target Columns (Logical)	Position	Data Type	Length
☐	MANDT	MANDT_Client	1	CLNT	3
✓	AUFNR	AUFNR_Order Number	2	CHAR	12
✓	AUART	AUART_Order type	3	CHAR	4
✓	AUTYP	AUTYP_Order category	4	NUMC	2
✓	REFNR	REFNR_Reference order number	5	CHAR	12
✓	ERNAM	ERNAM_Entered by	6	CHAR	12
☐	ERDAT	ERDAT_Date created	7	DATS	8
☐	AENAM	AENAM_User ID of person who m...	8	CHAR	12
☐	AEDAT	AEDAT_Change date for Order Ma...	9	DATS	8
☐	KTEXT	KTEXT_Short text	10	CHAR	40
☐	LTEXT	LTEXT_Indicator: "Long text exists"	11	CHAR	1

By default, this page displays and selects all columns in the source table as defined in the SAP application. You can deselect columns in one of the following ways:

- Click the check box next to the column.
- Click **Deselect All** to deselect all the columns within the table.

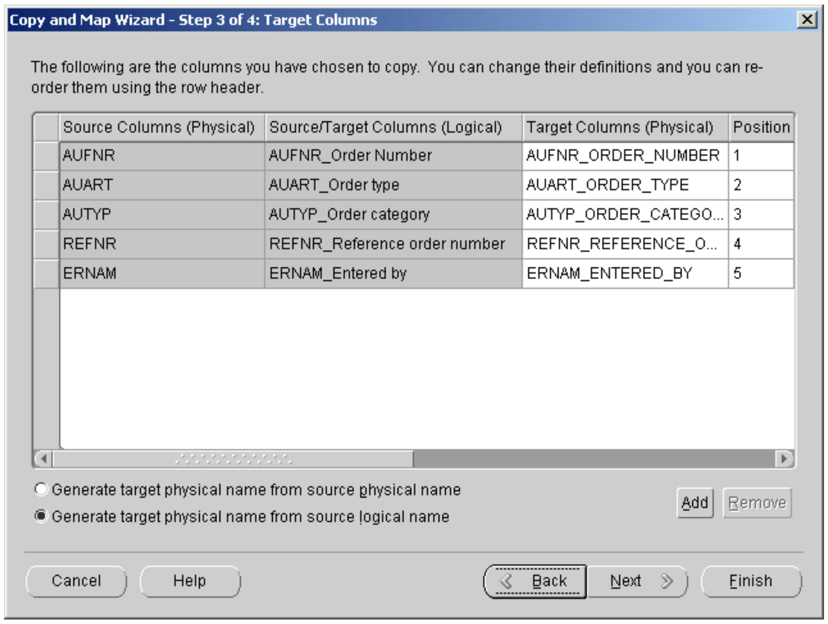
You must select at least one column to copy. You can select columns in one of the following ways:

- Click the check box next to the column.
- Click **Select All** to copy all the columns within the table.

7. Click **Next.**

The **Target Columns** page displays as shown in Figure 21–14.

Figure 21–14 Target Columns Page



Copy and Map Wizard - Step 3 of 4: Target Columns

The following are the columns you have chosen to copy. You can change their definitions and you can re-order them using the row header.

Source Columns (Physical)	Source/Target Columns (Logical)	Target Columns (Physical)	Position
AUFNR	AUFNR_Order Number	AUFNR_ORDER_NUMBER	1
AUART	AUART_Order type	AUART_ORDER_TYPE	2
AUTYP	AUTYP_Order category	AUTYP_ORDER_CATEGO...	3
REFNR	REFNR_Reference order number	REFNR_REFERENCE_O...	4
ERNAM	ERNAM_Entered by	ERNAM_ENTERED_BY	5

○ Generate target physical name from source physical name
● Generate target physical name from source logical name

Use this page to customize the selected columns or add new ones to your target table. The first two columns within the table display the physical and business names of the columns. You can modify the physical names of a column. The default physical names are copied from the source physical names. You can edit the following fields:

– **Target Columns (Physical):** To change the physical name of a target column, click the value and edit the text. To add a new column, click **Add** and type a new in the name field. If you add a new column, you must name it.

– **Position:** Change the position of the columns by dragging the row header up or down.

– **Data Type:** Change the data type for each column. It is automatically translated to an Oracle data type.

– **Not Null:** (optional) A check indicates that the column cannot contain a NULL value. Use this option for unique and primary keys for loads that require constraint checking.

- Click **Add** to add a new column within your target object. The **Remove** option is only enabled for columns you create. If you do not want to copy a column defined in the source object, click **Back** to return to the Source Columns page and deselect the column.
- Click **Generate target physical name from source physical name** to copy the physical target column names from the physical source column names.
- Click **Generate target physical name from source business name** to copy the physical target column names from the logical source column names. The business names can exceed the Oracle physical name limit of 30 characters.

8. Click **OK** to rename the column automatically or click **Cancel** to rename it yourself.

If you rename a target column, then the new name is preserved regardless of how you choose to generate the target physical name. The renamed target column supersedes the generated target column name. For example, the source physical name of a table is MANDT and its business name is MANDT_Client. If you change the default target physical name from MANDT to MANDT_Client_Name, then this name is preserved.

9. Click **Next.**

   The **Summary and Copy** page displays. Verify the information on this page.

10. Click **Finish.**

    The Mapping Editor displays the source table mapped to the newly created target table. The bound target table displays under the TABLES node in the Warehouse Builder navigation tree.

Configuring Code Generation for SAP Objects

Configuring a mapping containing an SAP source is the similar to configuring a mapping containing any other source:

- Use the Operator properties window to set the loading properties.
- Use the Configuration properties window to define the code generation properties.
- If you intend to generate ABAP code, you set directory and initialization file settings in the Configuration properties window.

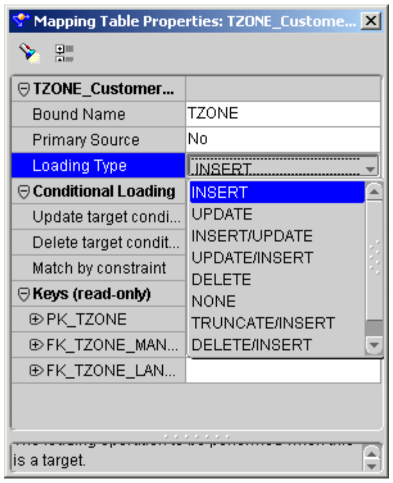
Setting the Loading Type

To set the loading type for an SAP operator:

1. From the Mapping Editor, right-click the SAP source operator and select **Operator Properties** from the popup menu.

   The Operator properties window displays as shown in Figure 21–15.

Figure 21–15 Mapping Table Operator Properties Window for an SAP Target



Mapping Table Properties: TZONE_Custome...

TZONE_Customer...	
Bound Name	TZONE
Primary Source	No
Loading Type	INSERT
Conditional Loading	
Update target condi...	
Delete target condit...	
Match by constraint	
Keys (read-only)	
PK_TZONE	
FK_TZONE_MAN...	
FK_TZONE_LAN...	

is a target.

2. Select a loading type from the Loading Type drop-down list. If you specify ABAP code as the step type for the mapping, Warehouse Builder generates SQL*Loader code as indicated in Table 21–1.

Table 21–1 Loading Types in ABAP Code

Loading Type	SQL* Loader Code Generated in ABAP Code
INSERT	APPEND
CHECK/INSERT	INSERT
TRUNCATE/INSERT	TRUNCATE
DELETE/INSERT	REPLACE
All other types	APPEND

3. Close the Operator Properties window to save the setting.

Setting the Step Type Parameter

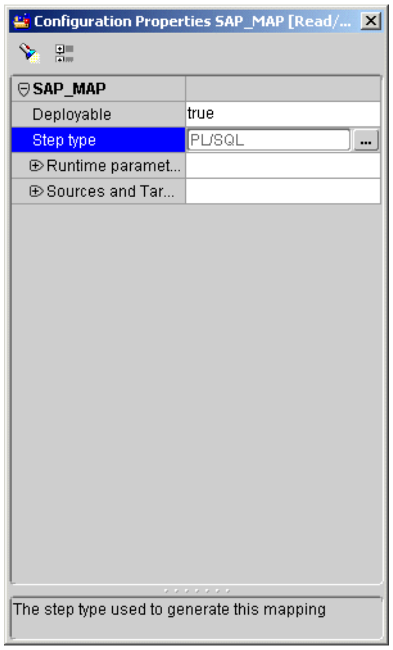
This parameter enables you to choose the type of code you want to generate for your SAP mappings. If your source includes clustered or pooled tables, then you must select ABAP as the generated code.

To choose the step type:

1. Right-click the mapping and select **Configure** from the popup menu.

   The Configuration Properties window displays as shown in Figure 21–16.

Figure 21–16 Step Type in the Configuration Properties Window



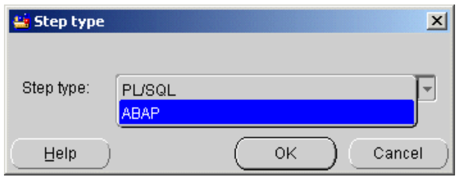
Deployable	true
Step type	PL/SQL

The step type used to generate this mapping

2. Click the **Step Type** field and click the **...** button.

   The Step Type dialog displays as shown in Figure 21–17.

Figure 21–17 Step Type Dialog



3. From the drop-down list, select the type of code you want to generate: ABAP or PL/SQL scripts (available for transparent tables only).

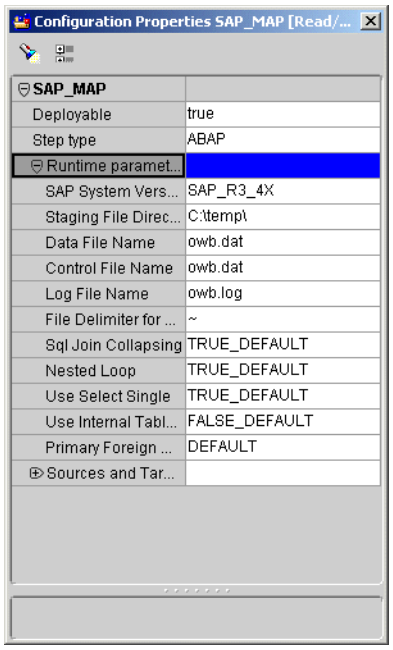
4. Click **OK.**

   The Configuration Properties window displays.

Setting the Runtime Parameters

If you set the step type to ABAP, then you can expand the Runtime Parameters node on the Configuration Properties window to display settings specific to ABAP code Generation, as shown in Figure 21–18. These settings come with preset properties that optimize code generation and should not be changed. If you alter these settings, you may slow the code generation process.

Figure 21–18 Runtime Parameters in the Configuration Properties Window



SAP_MAP	
Deployable	true
Step type	ABAP
Runtime paramet...	
SAP System Vers...	SAP_R3_4X
Staging File Direc...	C:\temp\
Data File Name	owb.dat
Control File Name	owb.dat
Log File Name	owb.log
File Delimiter for ...	~
Sql Join Collapsing	TRUE_DEFAULT
Nested Loop	TRUE_DEFAULT
Use Select Single	TRUE_DEFAULT
Use Internal Tabl...	FALSE_DEFAULT
Primary Foreign ...	DEFAULT
Sources and Tar...	

The following lists the runtime parameters available for SAP mappings:

- **SAP System Version:** Specifies the SAP system version number to which you want to deploy the ABAP code.
- **Staging File Directory:** Specifies the location of the directory where the data generated by ABAP code resides.
- **Data File Name:** Specifies the name of the data file created during code generation.

- **Control File Name:** Specifies the name of the control file created during code generation.
- **Log File Name:** Specifies the log file name created during code generation. This file is useful for debugging purposes.
- **File Delimiter for Staging File:** Specifies the column separator in a SQL data file.
- **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.
- **Nested Loop:** Specifies a hint to generate nested loop code for a join, if possible
- **Use Single Select:** Indicates whether Warehouse Builder generates Select Single..., if possible.
- **Use Internal Table for Lookup only Table:** Specifies whether you want to generate code using internal table, if possible.

Setting the Generation Target Directories

Before loading SAP data into the repository, you must first check the Physical Configuration properties window to ensure that the directory parameters are set correctly.

The following list describes the parameters related to generating ABAP code to the Generation Target Directories:

- **ABAP Directory:** Sets the storage location for ABAP scripts. The default is `abap\`.
- **ABAP Extension:** Sets the file name extension for ABAP scripts. The default is `.abap`
- **ABAP Run Parameter File:** Sets the run parameter file suffix for the parameter script in an ABAP job. The default is `_run.ini`.
- **ABAP Spool Directory:** Sets the location where ABAP scripts are buffered during script generation processing. The default is `abap\log\`

Generating SAP Definitions

You can generate PL/SQL code for a mapping containing an SAP transparent table just as you generated code for any other PL/SQL mapping in Warehouse Builder. However, you must generate ABAP code for pooled and cluster tables.

Warehouse Builder validates and generates the scripts required to create and populate the SAP source object.

When you generate code, a single script is generated for each physical object you want to create. For example, there is one script for each index you are creating. This is useful if you need to re-deploy a single object at a later time without re-deploying the entire warehouse.

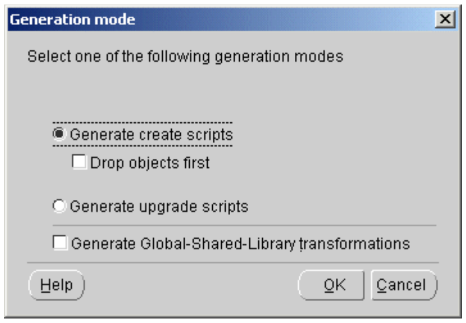
For related information, see Chapter 13, "Deploying Target Systems"

To generate the scripts for SAP mappings:

1. From the Warehouse Module Editor menu, choose **Module** then **Generate.**

   The Generation Mode dialog displays as shown in Figure 21–19.

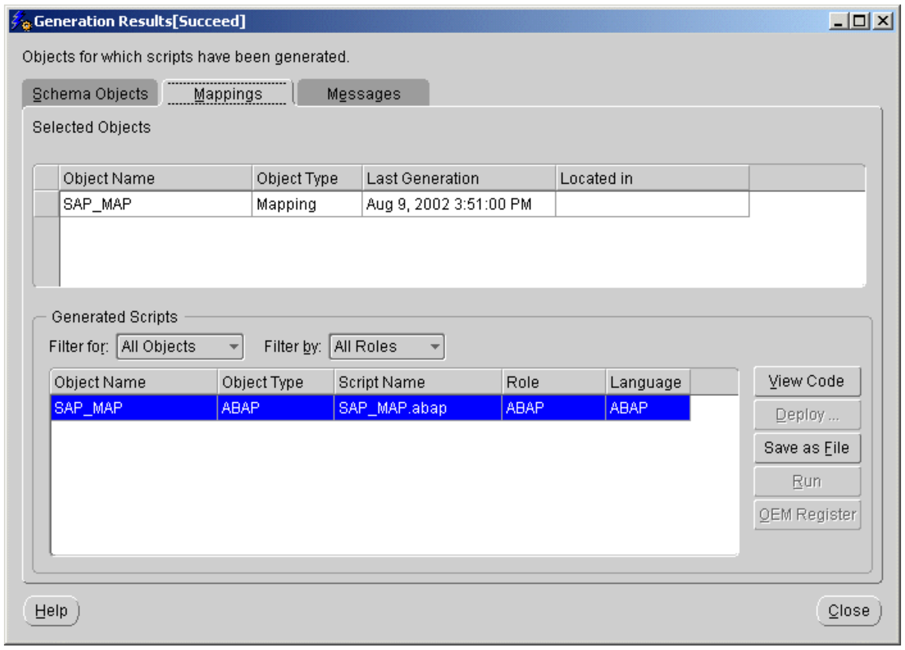
Figure 21–19 Generation Mode Dialog



2. Select a generation mode and click **OK.**

   The Generation Results dialog displays as shown in Figure 21–20.

Figure 21–20 Generation Results Dialog



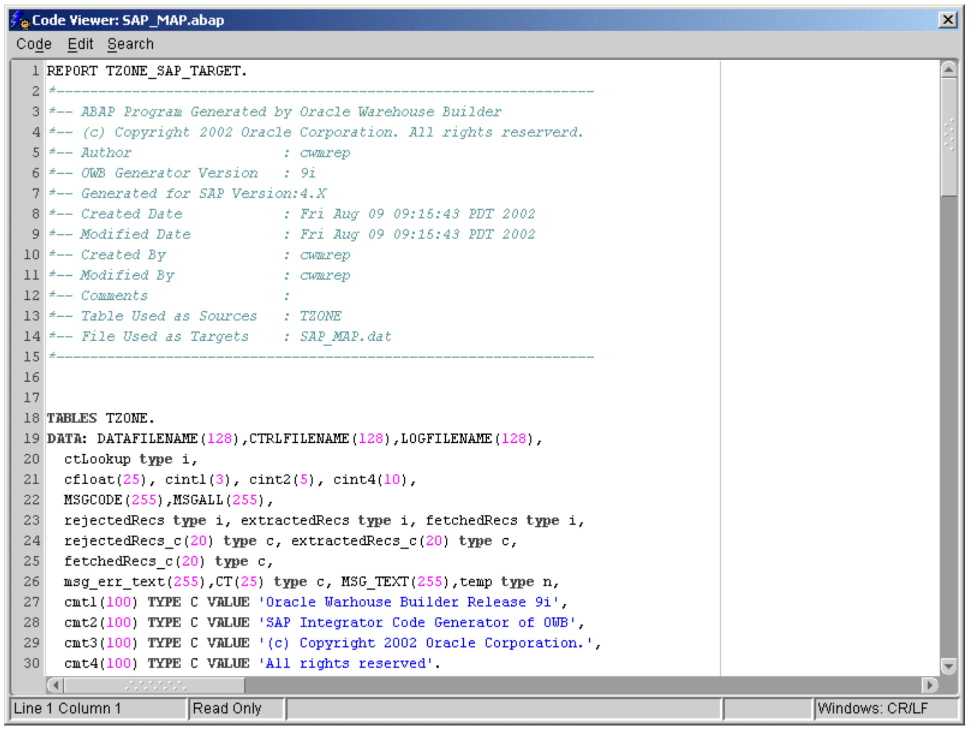
Object Name	Object Type	Last Generation	Located in
SAP_MAP	Mapping	Aug 9, 2002 3:51:00 PM	

Object Name	Object Type	Script Name	Role	Language
SAP_MAP	ABAP	SAP_MAP.abap	ABAP	ABAP

3. Click **View Code.**

   The generated code displays in the Code Viewer as shown in Figure 21–21.

Figure 21–21 Generated Code Displayed in the Code Viewer



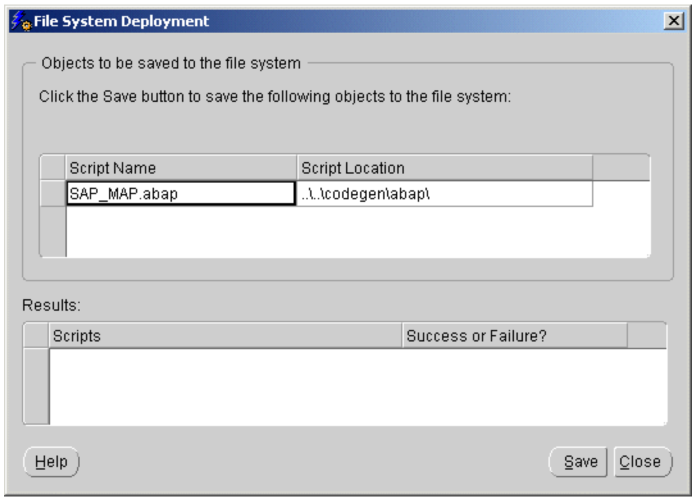
```
REPORT TZONE_SAP_TARGET.
*----------------------------------------------------------------
*-- ABAP Program Generated by Oracle Warehouse Builder
*-- (c) Copyright 2002 Oracle Corporation. All rights reserverd.
*-- Author                  : cwmrep
*-- OWB Generator Version   : 9i
*-- Generated for SAP Version:4.X
*-- Created Date            : Fri Aug 09 09:15:43 PDT 2002
*-- Modified Date           : Fri Aug 09 09:15:43 PDT 2002
*-- Created By              : cwmrep
*-- Modified By             : cwmrep
*-- Comments                :
*-- Table Used as Sources   : TZONE
*-- File Used as Targets    : SAP_MAP.dat
*----------------------------------------------------------------


TABLES TZONE.
DATA: DATAFILENAME(128),CTRLFILENAME(128),LOGFILENAME(128),
  ctLookup type i,
  cfloat(25), cint1(3), cint2(5), cint4(10),
  MSGCODE(255),MSGALL(255),
  rejectedRecs type i, extractedRecs type i, fetchedRecs type i,
  rejectedRecs_c(20) type c, extractedRecs_c(20) type c,
  fetchedRecs_c(20) type c,
  msg_err_text(255),CT(25) type c, MSG_TEXT(255),temp type n,
  cmt1(100) TYPE C VALUE 'Oracle Warhouse Builder Release 9i',
  cmt2(100) TYPE C VALUE 'SAP Integrator Code Generator of OWB',
  cmt3(100) TYPE C VALUE '(c) Copyright 2002 Oracle Corporation.',
  cmt4(100) TYPE C VALUE 'All rights reserved'.
```

You can edit, print, or save the file using the code editor. Close the window to return to the Generation Results dialog.

4. From the Generation Results dialog, click **Save as File** to save the ABAP code to your hard drive.

   The File System Deployment dialog displays as shown in Figure 21–22.

Figure 21–22 File System Deployment Dialog



Click the Save button to save the following objects to the file system:

Script Name	Script Location
SAP_MAP.abap	..\..\codegen\abap\

Scripts	Success or Failure?

5. Click **Save** to save the generated scripts to a file system. You can save the ABAP code with any file extension. You can use the suffix .abap (for example, MAP1.abap) or any other naming convention.

Loading SAP Data into the Repository

When you generate an ABAP code for an SAP mapping, Warehouse Builder creates an ABAP program that loads the data. You must run this program from the SAP user interface. The program uploads the generated code and executes it on your SAP system. You can then load the data into your Warehouse Builder staging area before using SQL*Loader to upload the data into your warehouse tables.

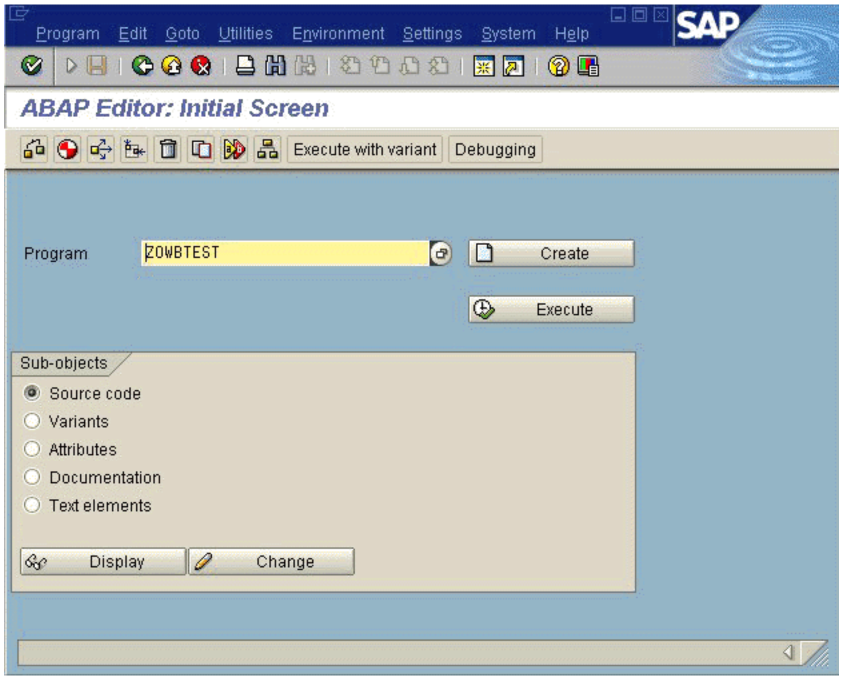
To upload and execute the ABAP code on your SAP system using the SAP user interface:

1. Open the SAP user interface and specify op-code `SE38`.
2. Create a program to execute the ABAP code (for example, ZOWBTEST1). For detailed instructions on creating a program, refer to your SAP documentation. If you already have a program created for testing purposes, you can use it to execute the ABAP code.

   The default selection is set to Source Code.

   Figure 21–23 shows the SAP ABAP editor.

Figure 21–23 SAP ABAP Editor

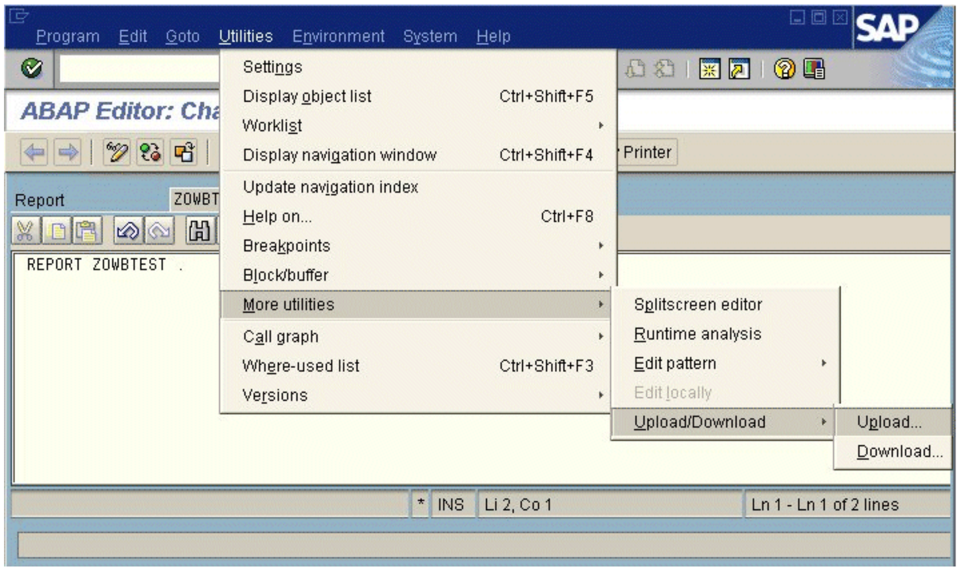


Program: ZOWBTEST

Sub-objects:
- [✓ Source code]
- Variants
- Attributes
- Documentation
- Text elements

3. Click **Change.**

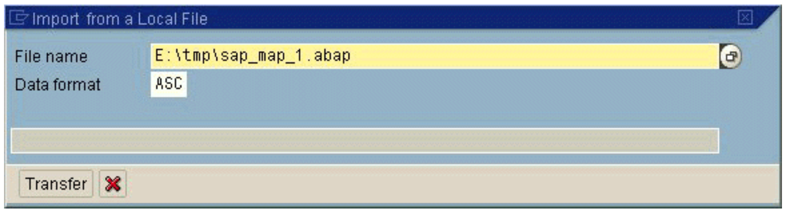
Figure 21–24 SAP ABAP Editor Upload Command



4. From the ABAP Editor screen, choose **Utilities,** then **Upload/Download,** and then **Upload,** as shown in Figure 21–24.

   The Import from a Local File dialog displays as shown in Figure 21–25.

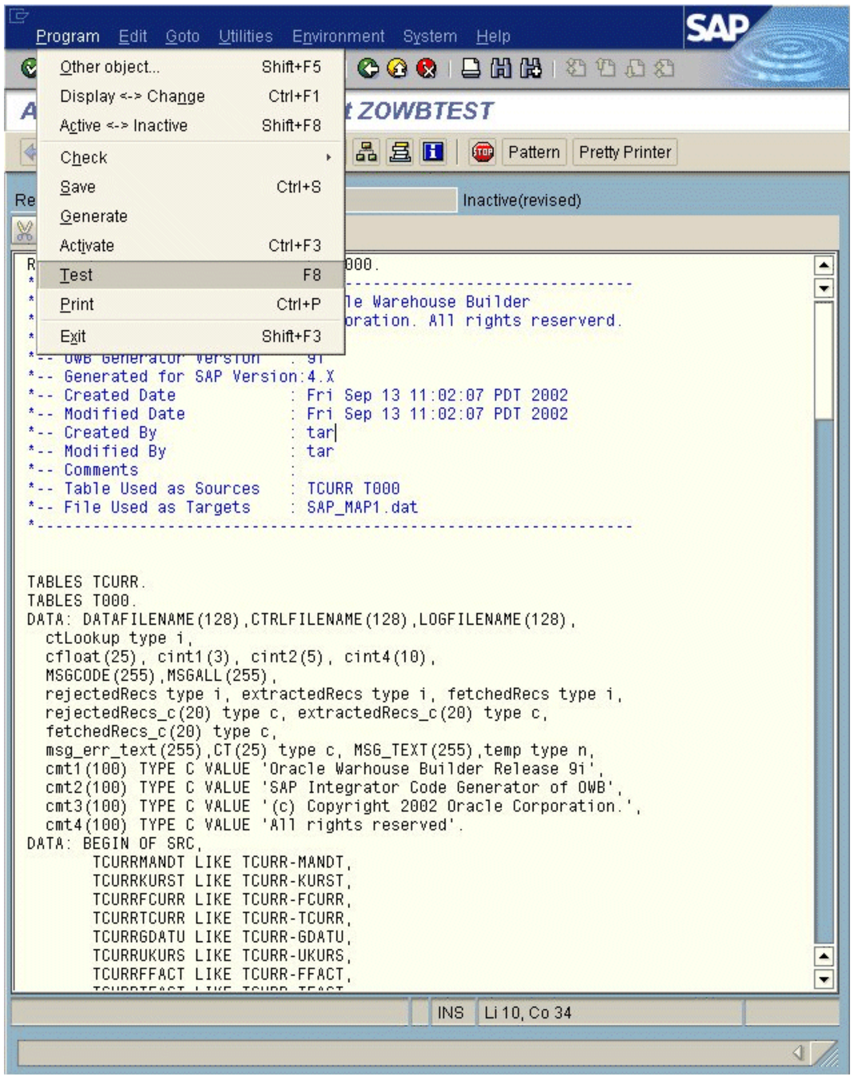
Figure 21–25 SAP Import From a Local File Dialog



File name E:\tmp\sap_map_1.abap
Data format ASC

5. Specify the location of the ABAP code generated by Warehouse Builder.
6. Click **Transfer.**

Figure 21–26 SAP ABAP Code Execution



A t ZOWBTEST

```
R                                000.
*                                --------------------------------
*                                le Warehouse Builder
*                                oration. All rights reserverd.
*
*-- OWB Generator Version     : 9i
*-- Generated for SAP Version:4.X
*-- Created Date              : Fri Sep 13 11:02:07 PDT 2002
*-- Modified Date             : Fri Sep 13 11:02:07 PDT 2002
*-- Created By                : tar
*-- Modified By               : tar
*-- Comments                  :
*-- Table Used as Sources     : TCURR T000
*-- File Used as Targets      : SAP_MAP1.dat
*-----------------------------------------------------------------


TABLES TCURR.
TABLES T000.
DATA: DATAFILENAME(128),CTRLFILENAME(128),LOGFILENAME(128),
  ctLookup type i,
  cfloat(25), cint1(3), cint2(5), cint4(10),
  MSGCODE(255),MSGALL(255),
  rejectedRecs type i, extractedRecs type i, fetchedRecs type i,
  rejectedRecs_c(20) type c, extractedRecs_c(20) type c,
  fetchedRecs_c(20) type c,
  msg_err_text(255),CT(25) type c, MSG_TEXT(255),temp type n,
  cmt1(100) TYPE C VALUE 'Oracle Warhouse Builder Release 9i',
  cmt2(100) TYPE C VALUE 'SAP Integrator Code Generator of OWB',
  cmt3(100) TYPE C VALUE '(c) Copyright 2002 Oracle Corporation.',
  cmt4(100) TYPE C VALUE 'All rights reserved'.
DATA: BEGIN OF SRC,
       TCURRMANDT LIKE TCURR-MANDT,
       TCURRKURST LIKE TCURR-KURST,
       TCURRFCURR LIKE TCURR-FCURR,
       TCURRTCURR LIKE TCURR-TCURR,
       TCURRGDATU LIKE TCURR-GDATU,
       TCURRUKURS LIKE TCURR-UKURS,
       TCURRFFACT LIKE TCURR-FFACT,
       TCURRTFACT LIKE TCURR-TFACT
```

7. Press F8 to execute the ABAP code. Or you can also choose **Program** and then **Check** before selecting **Program** and then **Execute** to run the code.

   The ABAP code generated in Warehouse Builder is executed in the SAP application server.

8. Use FTP to fetch data from the SAP application server and send it to the Warehouse Builder staging area.

9. Use SQL*Loader to upload data into your warehouse tables. The following is an example of a command line:

   ```
   SQLLDR USERID=scott/tiger CONTROL=abap_datactlfile.dat LOG=yourlogfile.log
   ```

Deploying PL/SQL Scripts for Transparent Tables

Deployment of PL/SQL scripts for SAP transparent tables is the same as deployment of PL/SQL scripts for Oracle database sources. The PL/SQL scripts run in your Oracle data warehouse and perform remote queries to extract table data from the SAP application.

For related information, see Chapter 13, "Deploying Target Systems"

22

Integrating Warehouse Builder Metadata with Other BI Products

Warehouse Builder provides several utilities for sharing its metadata with other Business Intelligence products. This chapter first shows you how to create collections that store definitions for a group of Warehouse Builder objects you can export to other systems using the Warehouse Builder Transfer Wizard. Next, this chapter outlines the steps to enable Business Intelligence (BI) integration and Online Analytical Processing (OLAP) using Warehouse Builder metadata. This chapter includes the following topics:

- Overview of Warehouse Builder Integration Features on page 22-1
- Defining Collections on page 22-2
- Business Intelligence Integration Using the Transfer Wizard on page 22-4
- Online Analytical Processing (OLAP) with Warehouse Builder on page 22-11
- Enabling OLAP with Warehouse Builder on page 22-12
- Warehouse Builder Bridges: Transfer Parameters and Considerations on page 22-28

Overview of Warehouse Builder Integration Features

The following sections in this chapter address Business Intelligence (BI) integration:

- **Warehouse Builder Transfer Wizard:** Use to synchronize, integrate, and use metadata stored in a variety of BI tools. Exchange metadata with CWM compliant applications, Oracle Discoverer, Oracle Express, and Oracle Database OLAP Server. Instructions for using the Transfer Wizard are discussed in this chapter. See "Business Intelligence Integration Using the Transfer Wizard" on page 22-4.
- **Online Analytical Integration with Warehouse Builder:** Warehouse Builder is the single tool that enables you to design, deploy, and load multidimensional OLAP objects from different data sources to your Oracle Database *database. After the data is loaded, you can use tools and applications to run complex analytical queries that answer your business questions. Using* Warehouse Builder*, you can now create and manage both your relational and analytical data stores, from the same metadata. See* "Online Analytical Processing (OLAP) with Warehouse Builder" on page 22-11.

Defining Collections

Collections are areas in Warehouse Builder that store the metadata you want to export to other tools and systems. You can use the Warehouse Builder Transfer Wizard to deploy a collection. Use collections to perform the following tasks:

- Organize a large logical warehouse.
- Validate and generate a group of objects in a collection.
- Export metadata to other tools using the Warehouse Builder Transfer Wizard.

You can use the Warehouse Builder Transfer Wizard to export collections from Warehouse Builder to Oracle Discoverer, OLAP Server, and CWM. For more information on exporting collections to other tools, see "Exporting Metadata from Warehouse Builder" on page 22-9.

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

You can define more than one collection within a project and an object can be referenced by more than one collection. For example, each user that accesses a project can create their own collection of frequently used objects. Each user can 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 or greyed out in the collection. To remove greyed out shortcuts from a collection, right-click the shortcut and select delete.

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

Creating a Collection

Use the New Collection Wizard to define a collection.

To define a new collection:

1. Select and expand a project node on the navigation tree.
2. Right-click the **Collections** node and select **Create Collection.**

   Warehouse Builder displays the Welcome page for the New Collections Wizard.
3. Click **Next.**

   The wizard displays the Name page.
4. Type in a name for the collection and an optional description.
5. Click **Next.**

   The wizard displays the Contents page .
6. Select and expand the project node in the left panel.

   The wizard displays a list of objects you can add to the collection.
7. Select objects from **Available Objects** in the left panel.

   Use the **Ctrl** key to select multiple objects. You can select objects at the object level or the module level. For example, under the Files node, you can 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.

8. Click the left to right arrow.

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

9. Click **Next.**

   The wizard displays the Finish page.

10. Use the Finish page to view the objects you selected for the collection.

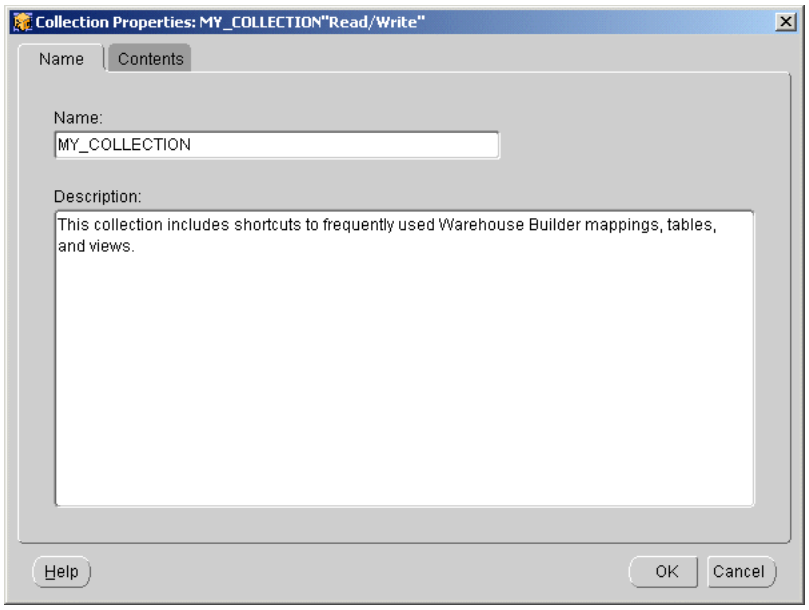
    Select **Back** to make changes to your selections.

    When you select **Finish,** Warehouse Builder creates the collection and adds it to the navigation tree. All the objects under the collection are designated with a shortcut symbol in their associated icons.

Editing a Collection

To edit a collection, right-click the collection from the navigation tree and select Properties. Warehouse Builder displays the properties window as shown in Figure 22–1.

Figure 22–1 Properties Window for a Collection


Collection Properties: MY_COLLECTION"Read/Write"

Name: MY_COLLECTION

Description: This collection includes shortcuts to frequently used Warehouse Builder mappings, tables, and views.

You can edit the following tabs in the properties window:

- Name
- Contents

Specify the following on the Name Tab:

- **Name:** You can rename the collection. Type in a name for the collection that is 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.

- **Description:** You can edit the optional text description with a maximum of 4000 characters.

Specify the following on the Contents Tab:

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

To complete the Contents page:

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

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

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

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

3. Click the left to right arrow.

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

Business Intelligence Integration Using the Transfer Wizard

The Warehouse Builder Transfer Wizard enables you to synchronize, integrate, and use metadata stored in multiple sources and formats. The Warehouse Builder Transfer Wizard enables you to import metadata from and export metadata to various tools.

The Transfer Wizard performs two major tasks:

1. Exporting selected metadata from the Warehouse Builder repository using a bridge to a variety of targets. The target can be:

   - CWM (Common Warehouse Metamodel) compliant applications version 1.0
   - Oracle Discoverer versions 4*i* and 9*i*
   - Oracle Express
   - Oracle Database OLAP Server

   The Oracle Discoverer and Oracle Express bridges are only available if you are running Oracle Warehouse Builder client on a Windows platform.

2. Importing selected metadata from source tools into the Warehouse Builder repository using a bridge. The source can be:

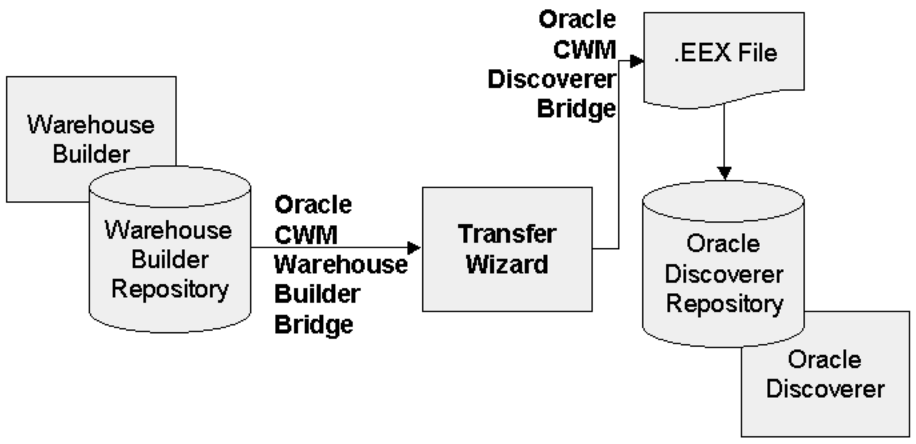
   - Oracle Database OLAP Server
   - OMG CWM (Common Warehouse Metamodel) compliant applications version 1.0
   - Computer Associates ERwin (3.5.1)
   - Powersoft PowerDesigner (version 6)

   The ERwin bridge is only available if you are running Oracle Warehouse Builder client on a Windows platform.

For CWM applications, the Transfer Wizard creates an intermediate XML file conforming to the XML Metadata Interchange (XMI) standard. This process is transparent when you use the Transfer Wizard. You provide the source and target parameters and the Transfer Wizard performs the exporting, conversion, and downloading tasks.

Figure 22–2 shows an example of the transfer process for exporting metadata from Warehouse Builder into Discoverer.

Figure 22–2 Warehouse Builder Transfer Wizard Model


Warehouse Builder
Warehouse Builder Repository
Oracle CWM Warehouse Builder Bridge
Transfer Wizard
Oracle CWM Discoverer Bridge
.EEX File
Oracle Discoverer Repository
Oracle Discoverer

Integrating with the Meta Integration Model Bridges (MIMB)

Warehouse Builder enables you to integrate with Meta Integration Model Bridges (MIMB) that translate metadata from a proprietary metadata file or repository to the standard CWM format that can be imported into Warehouse Builder using the Transfer Wizard. After integrating with MIMB, you can also import metadata from CA ERwin, Sybase PowerDesigner, and many other sources through these bridges.

MIMB integration enables you to import metadata into Warehouse Builder from the following sources:

- OMG CWM 1.0: Database Schema using JDBC 1.0/2.0
- OMG CWM 1.0: Meta Integration Repository 3.0
- OMG CWM 1.0: Meta Integration Works 3.0
- OMG CWM 1.0: XML DTD 1.0 (W3C)
- OMG CWM 1.0: XML DTD for HL7 3.0
- Acta Works 5.x
- Adaptive Repository
- ArgoUML
- Business Objects Data Integrator
- Business Objects Designer 5.1.3 to 5.1.5
- CA COOL
- CA ERwin
- CA ParadigmPlus
- Hyperion Application Builder

- IBM
- Informatica PowerMart 5.1
- Merant App Master Designer 4.0
- Miscrodoft Visio
- Oracle Designer
- Popkin System Architect
- ProActivity 3.x & 4.0
- Rational Rose
- Select SE 7.0
- Silverrun
- Sybase Power Designer
- Unisys Rose
- Visible IE Advantage 6.1
- Various XML/XMI versions

Follow these steps to integrate Warehouse Builder with MIMB.

Download the Meta Integration Model Bridge

After you download the MIMB on your system, Warehouse Builder automatically recognizes the installation and displays the new import options on the Warehouse Builder Transfer Wizard: Metadata Source and Target Identification page.

To download MIMB:

1. Download the Model Bridge (personal) product from the following Web site: *http://www.metaintegration.net/Products/Downloads/*
2. Install the MIMB by running the setup on your system.
3. During installation, choose **Typical with Java Extensions** as the installation type from the Setup Type page.

   If the set up program is not able to find a JDK on your machine, you must provide the JNI library directory path name. Your path environment variable must contain the metaintegration directory. If not, you need to add it to the path: *c:\program files\metaintegration\win32.*
4. Enable MIMB through your Warehouse Builder client by starting the Warehouse Builder Transfer Wizard (Import). For more information on importing metadata using the Warehouse Builder Transfer Wizard, see "Importing Metadata into Warehouse Builder" on page 22-6.

Importing Metadata into Warehouse Builder

This section contains instructions for using the Warehouse Builder Transfer Wizard to import metadata into Warehouse Builder:

To launch the Transfer Wizard for an import:

1. From the **Project** menu, select **MetaData Import,** and then **Bridge.**

   The Oracle Transfer Wizard Welcome window displays, identifying the steps you perform while using the Transfer Wizard.

If you want to display version information about the Transfer Wizard, click **About Oracle WB Transfer Tool**. For version information about the individual bridges, press the **Bridge Versions** button from the About Oracle WB Transfer Tool dialog.

2. Click **Next.**

   The Metadata Source and Target Identification window displays as shown in Figure 22–3

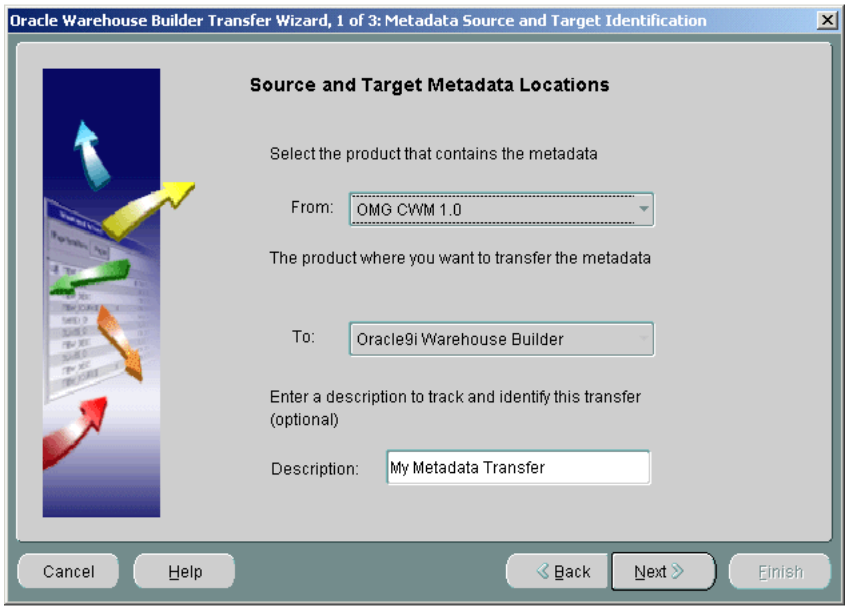
3. In the **From** field, identify your metadata source (Oracle9*i* OLAP, OMG CWM 1.0, CA ERwin, PowerDesigner). You can import metadata from many other sources if you choose to integrate with the Meta Integration Model Bridge (MIMB). For details, see "Integrating with the Meta Integration Model Bridges (MIMB)" on page 22-5.

   If you integrate with MIMB, an Info button displays next to the From field. Choose a source type and click **Info** to view details about using the bridge for that source type.

4. Optionally, enter a **Description** of the metadata to be transferred.

   This description displays in the progress bar during the transfer process.

Figure 22–3 Data Source and Target Identification Window


Oracle Warehouse Builder Transfer Wizard, 1 of 3: Metadata Source and Target Identification

Source and Target Metadata Locations

Select the product that contains the metadata

From: OMG CWM 1.0

The product where you want to transfer the metadata

To: Oracle9i Warehouse Builder

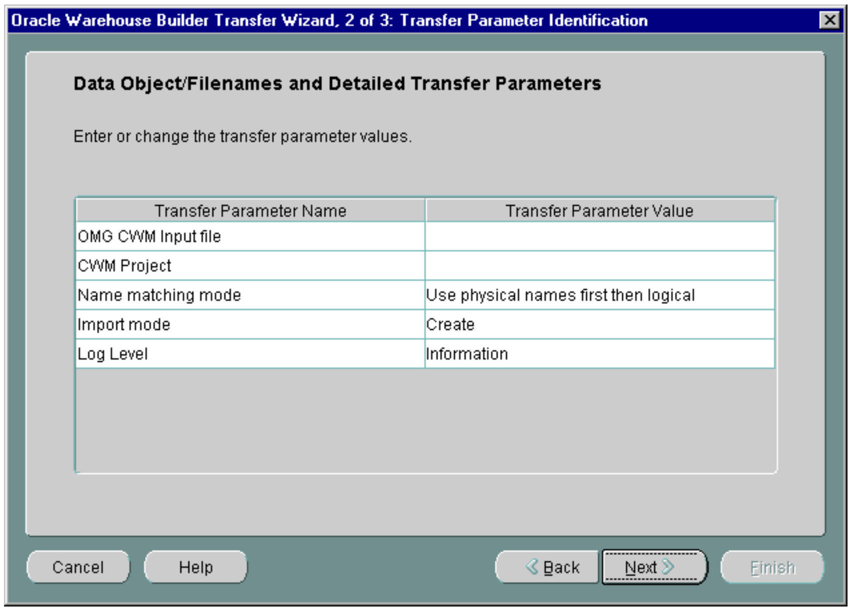
Enter a description to track and identify this transfer (optional)

Description: My Metadata Transfer

5. Click **Next.**

   The Transfer Parameter Identification window displays as shown in Figure 22–4.

Figure 22–4 Transfer Parameter Identification Window


Oracle Warehouse Builder Transfer Wizard, 2 of 3: Transfer Parameter Identification

Data Object/Filenames and Detailed Transfer Parameters

Enter or change the transfer parameter values.

Transfer Parameter Name	Transfer Parameter Value
OMG CWM Input file	
CWM Project	
Name matching mode	Use physical names first then logical
Import mode	Create
Log Level	Information

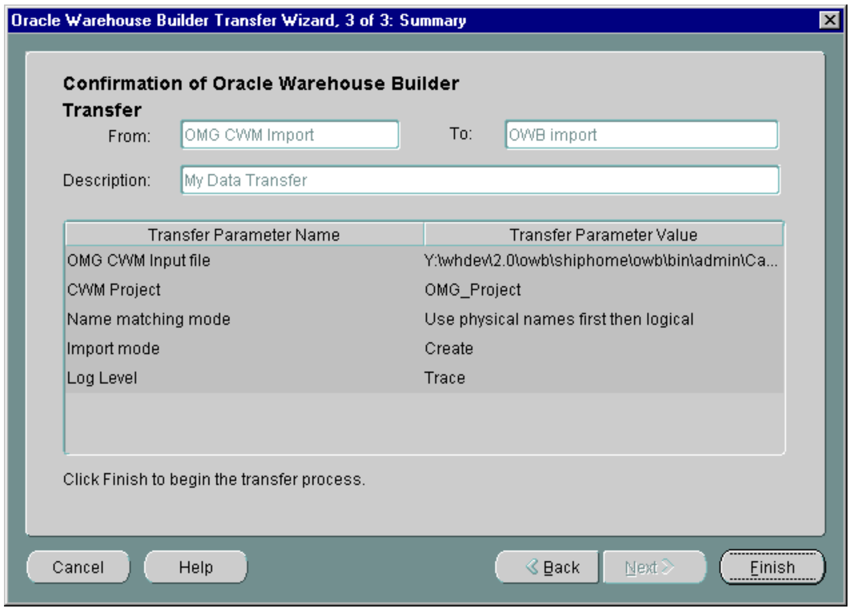
Cancel
Help
Back
Next
Finish

The Transfer Parameters window displays a different list of parameters based upon the metadata source you selected. For details, see "Transfer Parameters" on page 22-28.

If you are running the Meta Integration Model Bridge (MIMB), the descriptions of the parameters display in the box under on the wizard page. For details, see "Integrating with the Meta Integration Model Bridges (MIMB)" on page 22-5.

6. Click **Next.** The Summary window displays as shown in Figure 22–4.

Figure 22–5 Summary Window



Confirmation of Oracle Warehouse Builder Transfer

From: OMG CWM Import To: OWB import

Description: My Data Transfer

Transfer Parameter Name	Transfer Parameter Value
OMG CWM Input file	Y:\whdev\2.0\owb\shiphome\owb\bin\admin\Ca...
CWM Project	OMG_Project
Name matching mode	Use physical names first then logical
Import mode	Create
Log Level	Trace

Click Finish to begin the transfer process.

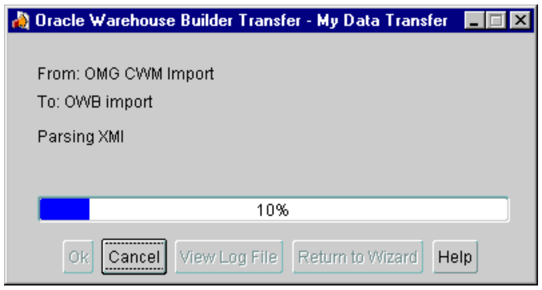
7. Review your entries.

If any are incorrect, click **Back** to return to the previous screen and make the necessary changes.

8. Click **Finish** on the Confirmation window.

   The transfer window displays with a status bar as shown in Figure 22–6.

Figure 22–6 Data Transfer Progress Panel



From: OMG CWM Import
To: OWB import
Parsing XMI
10%

The title of the transfer window is the description you provided. 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.

9. Do one of the following:

   - If the transfer completes successfully (100% displays), click **OK.**
   - If a transfer failure message displays, click **View Log File** and review the log. (You can also view the log of a successful transfer.)

     To save the log for reference, click **Save As** to open the Save dialog. Select the folder where you want to store the log and click **Save.**
   - If you determine the cause of the failure from the Information Log, note the information requiring update. Close the log by clicking **OK.** On the transfer window, click **Return to Wizard** and update the incorrect information on the Transfer Parameters window. 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 transfer window, click **Return to Wizard** and change the Log Level to **Trace** on the Transfer Parameters window. Then transfer the data again.

   If the transfer is successful, the Transfer Wizard creates an output file and stores it in the location you specified.

Exporting Metadata from Warehouse Builder

The Warehouse Builder Transfer Wizard enables you to export metadata to the following types of targets:

- A file that conforms to the OMG CWM standard
- Oracle Discoverer versions 4*i* and 9*i*
- Oracle Express
- Oracle Database OLAP

To export metadata to any of these targets, you must first define collections in Warehouse Builder. See "Defining Collections" on page 22-2.

To export metadata using the Warehouse Builder Transfer Wizard:

1. From the **Project** menu, select **MetaData Export,** and then **Bridge.**

   The Oracle Transfer Wizard Welcome window displays, identifying the steps you perform while using the Transfer Wizard.

2. Click **Next.**

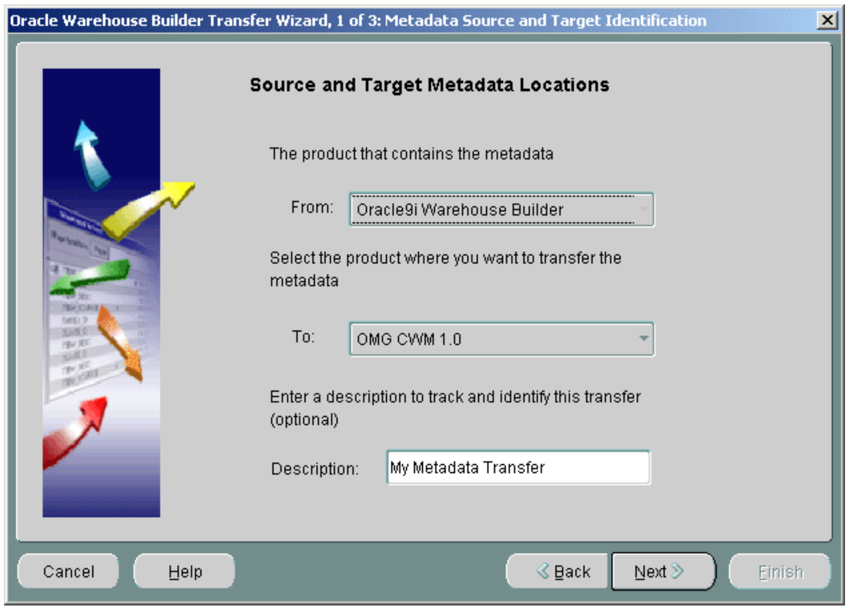
   The Metadata Source and Target Identification window displays as shown in Figure 22–7.

3. In the **To** field, select your target (OMG, Express, Discoverer, or the OLAP server) to identify the target for your export.

4. Optionally, enter a **Description** of the metadata to be transferred.

   This description displays in the progress bar during the transfer process.

Figure 22–7 Source and Target Information Window


Oracle Warehouse Builder Transfer Wizard, 1 of 3: Metadata Source and Target Identification
Source and Target Metadata Locations
Cancel
Help
Back
Next
Finish

The product that contains the metadata

From: Oracle9i Warehouse Builder

Select the product where you want to transfer the metadata

To: OMG CWM 1.0

Enter a description to track and identify this transfer (optional)

Description: My Metadata Transfer

5. Click **Next.**

   The Transfer Parameter Identification window displays.

   The Transfer Parameters window table lists parameters that you must enter or select. This window displays a different set of parameters depending on the target you selected in the previous step. For details, see "Transfer Parameters" on page 22-28.

6. Click **Next.**

   The Confirmation of Warehouse Builder Transfer window displays.

7. Review your entries.

   If any are incorrect, click **Back** to return to the previous screen and make the necessary changes.

8. Click **Finish.**

   The transfer window displays with a status bar.

The title of the transfer window is the description you assigned to the transfer on the Choose Data Source and Target Types window. 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.

9. Do one of the following:

   - If the transfer completes successfully, click **OK.**
   - If a transfer failure message displays, click **View Log File** and review the log. You can also view the log of a successful transfer.

     To save the log for reference, click **Save As** to open the Save dialog. Select the folder where you want to store the log and click **Save.**
   - If you determine the cause of the failure from the Information Log, note the data requiring update. Close the log by clicking **OK.** On the transfer window, click **Return to Wizard** and update the erroneous data on the Transfer Parameters window. 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 transfer window, click **Return to Wizard** and change the Log Level to **Trace** on the Transfer Parameters window. Then transfer the data again.

During a successful transfer, the Transfer Wizard creates the output file and stores it in the location you specified.

Online Analytical Processing (OLAP) with Warehouse Builder

Business organizations typically have complex analytic, forecast, and planning requirements. Analytic Business Intelligence (BI) applications provide a solution by answering critical business questions using the data available in your database. These applications run analytical queries that include time series analysis, inter-row calculations, access to aggregated historical and current data, and forecasts, some of which cannot be performed using SQL queries.

The OLAP option of the Oracle Database database enables you to perform all analytical tasks in the same database system that supports your Business Intelligence applications. For more information, see the *OLAP User's Guide.*

Warehouse Builder is the single tool that enables you to design, deploy, and load multidimensional OLAP objects from different data sources to your Oracle Database database. After the data is loaded, you can use tools and applications to run complex analytical queries that answer your business questions.

Using Warehouse Builder, *you can now create and manage both your relational and analytical data stores from the same metadata.*

About the Analytic Workspace

The analytic workspace (AW) is a container within the Oracle database that stores multidimensional data objects and procedures written in the OLAP DML. AWs are central to analytic processing. You can create multiple Analytic Workspaces within a single database. They are owned by a relational schema and can be shared among multiple users. You can use Warehouse Builder to populate AWs with data from another AW, a flat file, or from relational tables.

Enabling OLAP with Warehouse Builder

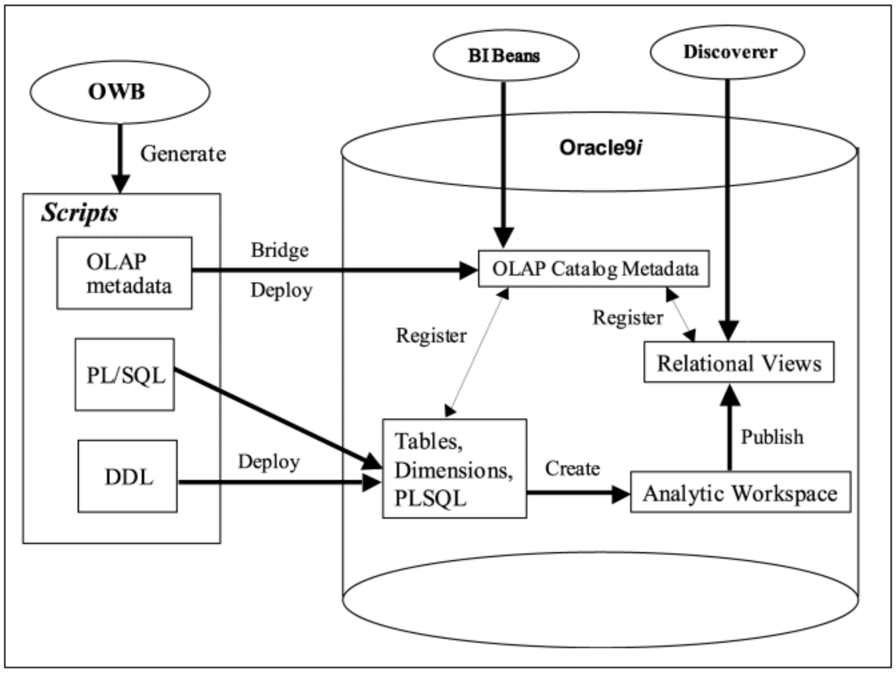
Warehouse Builder enables you to prepare your data stores for Oracle Database OLAP processing. Using Warehouse Builder, you can design, deploy, and load online analytical processing (OLAP) objects that provide complex analytic power to your data warehouse.

To run an analytical query without using Warehouse Builder, you need to manually perform these tasks in a MOLAP database:

1. Define Objects in the Analytic Workspace using PL/SQL or OLAP DDL commands.
2. Define the loading of the Analytic Workspace.
3. Create scripts for dependencies.
4. Map OLAP metadata to relational views.
5. Create materialized views.

Warehouse Builder enables you to perform all these activities to create an OLAP environment from a relational star schema, as shown in Figure 22–8. First, you can use the Warehouse Builder design environment to create your OLAP metadata. You can define multidimensional objects, such as dimensions and cubes, necessary to create the AW in your database and enable analytical processing.

Figure 22–8 Using Warehouse Builder to Create a ROLAP Environment



Using the Warehouse Builder Transfer Bridge, you can deploy this metadata to your database to create the AW, the relational views on top of it, and OLAP metadata based on these views. Next, you can define mappings or process flows in Warehouse Builder to load the data from different data sources to the AW. After you load the OLAP data in the AW, you are ready to perform complex analytical queries on it using BI tools such as Oracle Discoverer and BI Beans.

This section shows you how to perform the following tasks to design and create a Relational Online Analytical Processing (ROLAP) environment using Warehouse Builder:

- Create the relational warehouse.
- Create the OLAP metadata.
- Deploy the OLAP metadata.
- Load the data into the Analytic Workspace.

Creating the OLAP Metadata

You can use the Warehouse Builder wizards and editors to define your dimensions, hierarchies, attributes, and cubes. For detailed instructions on creating dimensions and cubes, see "Creating a Dimension Definition" on page 3-41 and "Creating Cube Definitions" on page 3-48. To create OLAP compliant metadata, use the guidelines when described in the following sections.

Defining OLAP Dimensions in Warehouse Builder

When you create a dimension, use the extensions listed in Table 22–1 while defining the implementing column for the dimension level attributes. Table 22–1 lists the level attribute names that are generated as OLAP compliant level descriptors when you deploy the dimension to the Oracle Database OLAP catalog using the Warehouse Builder Transfer Wizard.

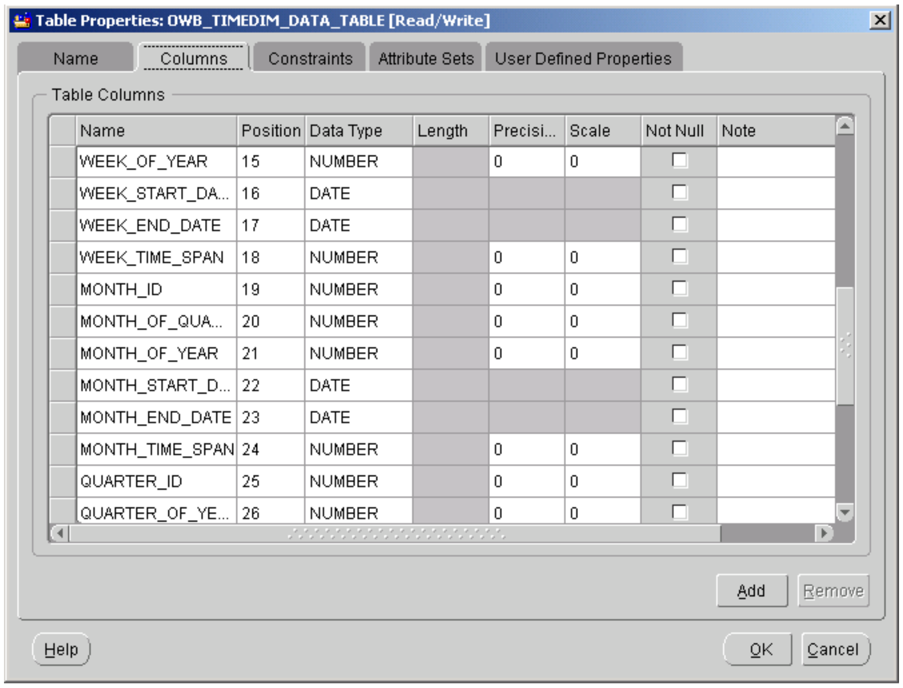
The generated Warehouse Builder level attribute named ID (created when a level is created) should not be used for the dimension attribute. Because no database dimension level attribute is created for this column, make sure that this level attribute does not follow this naming practice.

Table 22–1 Dimension Attribute Suffixes

Physical Level Attribute Name Suffixes in Warehouse Builder	Dimension Attribute Created
_NAME or NAME	Short_Description or Long_Description
_END_DATE or END_DATE	End_Date
_TIME_SPAN or TIME_SPAN	Time_Span
_PRIOR_PERIOD or PRIOR_PERIOD	Prior_Period
_YEAR_AGO_PERIOD or YEAR_AGO_PERIOD	Year_Ago_Period

Warehouse Builder creates a dimension attribute for each dimension level attribute that contains an implementing column with a known extension, as shown in Figure 22–9. For example, if an end date is specified for each level, Warehouse Builder also creates a dimension attribute for each level. If a fiscal end date and a calendar end date are specified for each level, then Warehouse Builder creates two dimension attributes using the names of the level attributes. For more information on creating dimensions in Warehouse Builder, see "Creating a Dimension Definition" on page 3-41.

Figure 22–9 Implementation of the Dimension Attributes in Columns



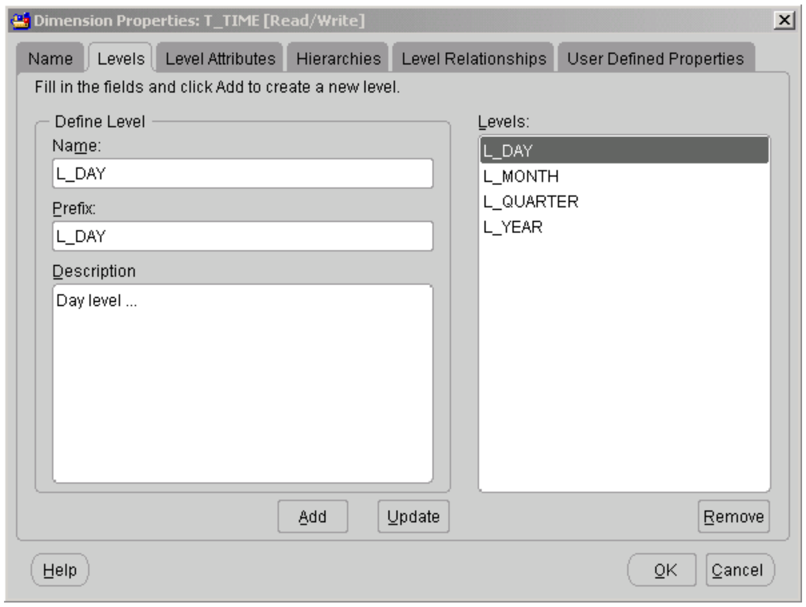
Name	Position	Data Type	Length	Precisi...	Scale	Not Null	Note
WEEK_OF_YEAR	15	NUMBER		0	0	☐	
WEEK_START_DA...	16	DATE				☐	
WEEK_END_DATE	17	DATE				☐	
WEEK_TIME_SPAN	18	NUMBER		0	0	☐	
MONTH_ID	19	NUMBER		0	0	☐	
MONTH_OF_QUA...	20	NUMBER		0	0	☐	
MONTH_OF_YEAR	21	NUMBER		0	0	☐	
MONTH_START_D...	22	DATE				☐	
MONTH_END_DATE	23	DATE				☐	
MONTH_TIME_SPAN	24	NUMBER		0	0	☐	
QUARTER_ID	25	NUMBER		0	0	☐	
QUARTER_OF_YE...	26	NUMBER		0	0	☐	

Time Dimensions An example of a time dimension is provided for you with your Warehouse Builder installation in the form of a Metadata Loader (`.MDL`) file. Navigate to *`OWB_Home_Directory`*`\owb\misc\time` directory. The `readme.txt` file in that directory explains how to use the prepackaged `.MDL` and `.SQL` files.

To create a time dimension in Warehouse Builder, you must follow these guidelines:

- Create the dimension name with the extension `_TIME` in upper case, for example, `T_TIME`. This enables the Transfer Bridge to create a descriptor of the type Time Dimension for the dimension. Figure 22–10 shows an example of a time dimension created using these rules.

Figure 22–10 Best Practice Time Dimension



Fill in the fields and click Add to create a new level.

Define Level

Name: L_DAY

Prefix: L_DAY

Description: Day level ...

Levels:
L_DAY
L_MONTH
L_QUARTER
L_YEAR

- Create time dimension levels with extensions listed in Table 22–2.

  Note that a Week-level suffix (`_WEEK`) is not included. A week cannot roll up neatly into a calendar month, quarter, or year because weeks span these time periods. If you use fiscal instead of calendar time periods for your time dimension, then weeks are neatly folded into the other time periods, and you can work in a Week level.

Table 22–2 Time Dimension Attribute Suffixes

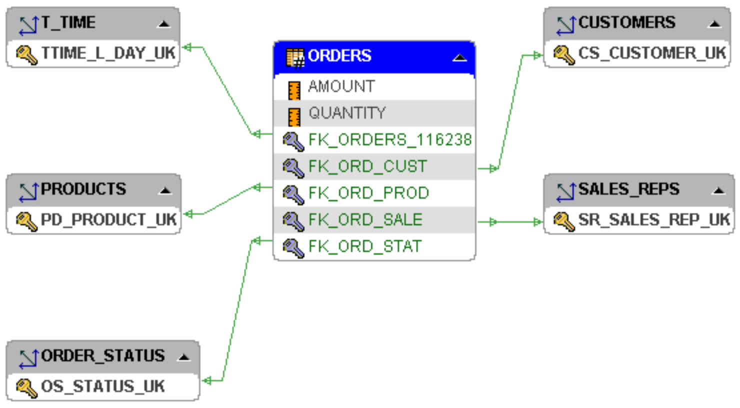
Physical Level Name in Warehouse Builder	Time Dimension Levels Created in Oracle Database
_DAY	Day Level Type
_MONTH	Month Level Type
_QUARTER	Quarter Level Type
_YEAR	Year Level Type

Defining Cubes in Warehouse Builder

The cube is a key object in OLAP processing that represents data in an organized and intuitive way. Warehouse Builder enables you to define a cube using the New Cube Wizard. For more information on creating cubes, see "Creating Cube Definitions" on page 3-48.

Figure 22–11 shows a sample cube created in Warehouse Builder.

Figure 22–11 A Cube Created in Warehouse Builder



Defining Collections in Warehouse Builder

Collections are areas in Warehouse Builder that store the metadata you want to deploy, for example, to your OLAP database. You can use the Warehouse Builder Transfer Wizard to deploy a collection and create the OLAP metadata that populates the OLAP catalog and creates the AW.

You must first define a collection in Warehouse Builder containing all the OLAP compatible dimensions and cubes.

Deploying OLAP Metadata Using the Warehouse Builder Transfer Wizard

After you define your metadata, you are ready to deploy the definitions to your Oracle Database OLAP database. To create your OLAP environment, follow these steps in Warehouse Builder:

Generate and Deploy DDL Scripts

Generate and deploy the DDL scripts (for dimensions, tables, and cubes) to create the relational storage objects in your OLAP database. You can later run a mapping or a process flow to load these structures with data. See "Loading the Analytic Workspace". For more information on generation and deployment, see Chapter 13, "Deploying Target Systems".

Deploy Metadata Using the Warehouse Builder Transfer Wizard

Run the Warehouse Builder Transfer Wizard to deploy metadata to the Oracle Database OLAP server. The Transfer Wizard populates the OLAP catalog, creates the Analytic Workspace objects within and the relational views on top of the Analytic Workspace. It also creates materialized views, if required.

To deploy OLAP metadata:

1. Start the Warehouse Builder Transfer Wizard from the **Project** menu by selecting **MetaData Export,** then **Bridge.**

   The Warehouse Builder Transfer Wizard Welcome page displays.

2. Click **Next.**

3. Provide the following information on the Metadata Source and Target Identification page:

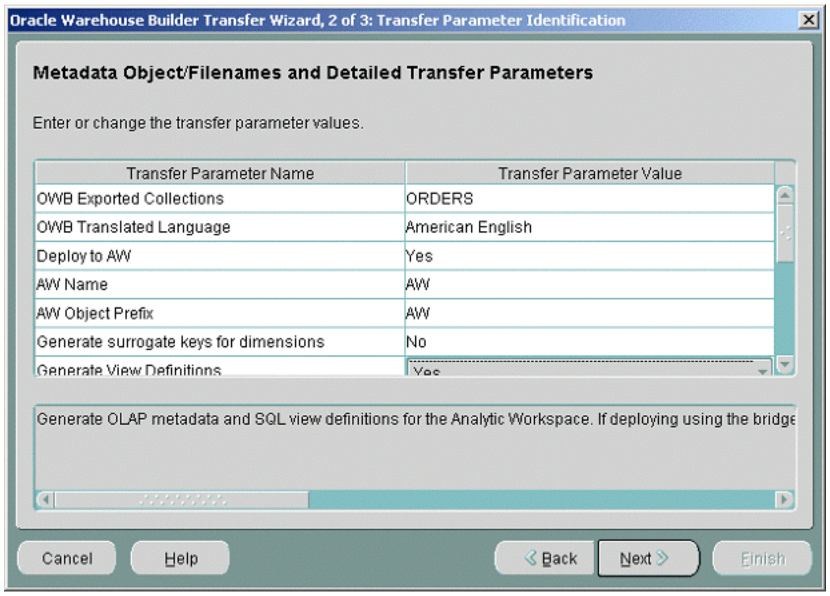
From: Oracle Warehouse Builder Export

To: Oracle9*i* OLAP Server

Description: Optional

Figure 22–12 shows the Warehouse Builder Transfer Wizard: Step 1 of 3.

Figure 22–12 Warehouse Builder Transfer Wizard: Step 1 of 3



Oracle Warehouse Builder Transfer Wizard, 2 of 3: Transfer Parameter Identification

Metadata Object/Filenames and Detailed Transfer Parameters

Enter or change the transfer parameter values.

Transfer Parameter Name	Transfer Parameter Value
OWB Exported Collections	ORDERS
OWB Translated Language	American English
Deploy to AW	Yes
AW Name	AW
AW Object Prefix	AW
Generate surrogate keys for dimensions	No
Generate View Definitions	Yes

Generate OLAP metadata and SQL view definitions for the Analytic Workspace. If deploying using the bridge

4. Click **Next.**
5. Provide the following information on the Transfer Parameter Identification page:

   OWB Exported Collections: Specify the collection containing the multidimensional metadata you want to export.

   Deploy to AW: Specify whether you want to deploy the metadata to the Analytic Workspace or not.

   AW Name: Provide the name of the Analytic Workspace where you want to deploy the metadata.

   AW Object Prefix: Prefix for the dimensions and cubes in the AW.

   Load Cube Data: Loads the cube data into the AW. If this option is not selected, only the cube loading program will be loaded.

   Generate Surrogate Keys for Dimensions: Generate surrogate keys for dimensions if your dimension values are not unique across all levels. This option can be used to generate surrogate keys.

   Generate View Definitions: Specify whether you want to generate view definitions for the Analytic Workspace. This option will also generate OLAP metadata for the cube based on the SQL view for the Analytic Workspace.

   Generate Materialized Views: Indicate whether you want to generate materialized views for the relational implementation of the cube.

   Generated View Directory: Specify the name of a directory to store the generated view scripts. If you choose the Deploy PL/SQL in Database option, this directory is located on the client side. Otherwise this is a server side directory object.

Deploy PL/SQL in Database: Specify if you want to deploy PL/SQL to the OLAP database. This option deploys the OLAP metadata and, if selected, the materialized views and views in the Analytic Workspace.

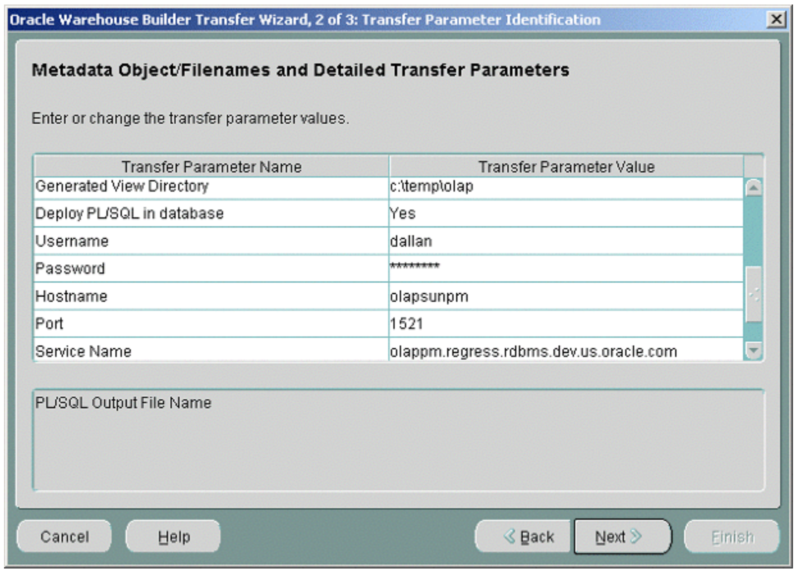
OLAP Instance Information: Specify the host name, Port, Service Name, Username, and Password to access the OLAP target instance.

PL/SQL Output File: Specify a location for the PL/SQL output file.

Log Level: Specify the level of detail you want to obtain from your log file after the transfer is complete.

Figure 22–13 shows the Warehouse Builder Transfer Wizard: Step 2 of 3.

Figure 22–13 Warehouse Builder Transfer Wizard: Step 2 of 3


Oracle Warehouse Builder Transfer Wizard, 2 of 3: Transfer Parameter Identification
Metadata Object/Filenames and Detailed Transfer Parameters
Enter or change the transfer parameter values.
PL/SQL Output File Name
Cancel
Help
Back
Next
Finish

Transfer Parameter Name	Transfer Parameter Value
Generated View Directory	c:\temp\olap
Deploy PL/SQL in database	Yes
Username	dallan
Password	********
Hostname	olapsunpm
Port	1521
Service Name	olappm.regress.rdbms.dev.us.oracle.com

6. Click **Next.**

   The Transfer Wizard Summary page displays.

7. Review the information you provided in the previous steps. Click **Finish** to start the deployment.

   The Transfer Bridge generates a script to perform the following:

   - Create OLAP metadata for cubes in the collection.
   - Optionally create metadata in an AW based on the ROLAP metadata.
   - Optionally generate a script to create SQL views and OLAP metadata based on the AW. For details on these views, refer to the OLAP *User's Guide.*
   - Optionally generate a script to create Materialized Views based on the relational data for performance of BI Beans and OLAPI. For details on these views, refer to the OLAP *User's Guide*.

Debugging the Export to OLAP

When you use the Deploy PL/SQL in Database option for the bridge, the diagnostics are provided in a log file. If you execute the generate PL/SQL manually, then the following helps you in debugging.

To debug the OLAP export, you can save the generated PL/SQL code to a file and execute it using SQL*Plus. To see additional error details, the server output must be switched on in SQL*Plus. For example, before executing the script in SQL*Plus, you must execute the following:

```
SET SERVEROUTPUT ON SIZE 99999
```

When there are errors, you will see detailed error messages outlining the types of errors. A common error for the OLAP bridge is when you fail to deploy dependent objects, such as dimensions, to the relational part of the server. For example, if you run the bridge before you deploy the relational dimension, the following error displays as shown in Table 22–3.

Table 22–3 Error Messages When the Dimension Has Not Been Deployed

Error Details	Description
ERROR: dimension_not_found	Error type from OLAP
Object Type: DIMENSION	The object type concerned.
Object Owner: RTS60	The schema name.
Object Name: DD	The object name.
Secondary Name	Second class object name (column name).
Tertiary Name	Third class object name (level attribute name).

Loading the Analytic Workspace

After you deploy the metadata to your OLAP database system, you can load your relational source data into the analytic workspace using one of the following methods:

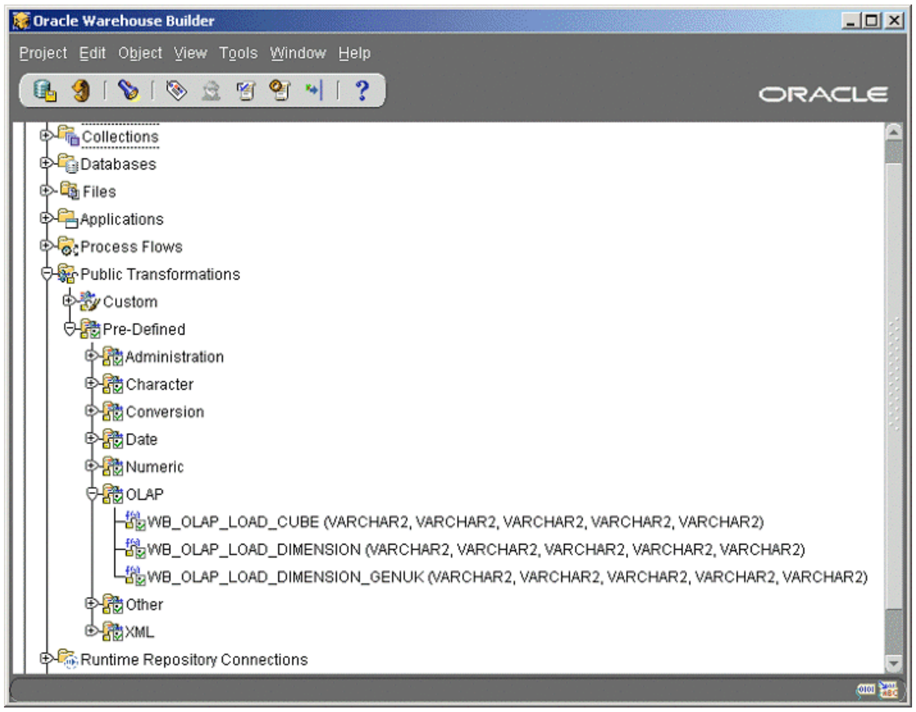
- Design a mapping with a post-mapping process that uses a PL/SQL routine provided with Warehouse Builder.
- Design a process flow containing a PL/SQL routine provided with Warehouse Builder.

Warehouse Builder provides PL/SQL packages that enable you to create and load (or refresh) relational data into the analytic workspace. These packages form a wrapper on top of the DBMS-OLAP procedures provided in Oracle Database Release 2. For detailed information, see the OLAP User's Guider for documentation on `DBMS_AWM` routines. This package provides routines for performing incremental loads and customized aggregations behavior.

To locate the PL/SQL packages in Warehouse Builder:

1. From the Warehouse Builder tree, expand the PUBLIC TRANSFORMATIONS node.
2. Expand the Pre-Defined node and then expand the OLAP node as shown in Figure 22–14.

Figure 22–14 PL/SQL Packages in Warehouse Builder



```
Collections
Databases
Files
Applications
Process Flows
Public Transformations
  Custom
  Pre-Defined
    Administration
    Character
    Conversion
    Date
    Numeric
    OLAP
      WB_OLAP_LOAD_CUBE (VARCHAR2, VARCHAR2, VARCHAR2, VARCHAR2, VARCHAR2)
      WB_OLAP_LOAD_DIMENSION (VARCHAR2, VARCHAR2, VARCHAR2, VARCHAR2, VARCHAR2)
      WB_OLAP_LOAD_DIMENSION_GENUK (VARCHAR2, VARCHAR2, VARCHAR2, VARCHAR2, VARCHAR2)
    Other
    XML
Runtime Repository Connections
```

Warehouse Builder displays the wrapper packages: WB_OLAP_LOAD_CUBE, WB_OLAP_LOAD_DIMENSION, and WB_OLAP_LOAD_DIMENSION_GENUK.

Creating a Mapping to Load an Analytic Workspace

Create a mapping that joins different dimension tables, transforms them, and loads them into the dimension or cube. To load the data into the Analytic Workspace, you must define a Post-Mapping Process with a Constant containing the load values.

To add a Post-Mapping Process to a mapping to create and load the Analytic Workspace:

1. Drag and drop a Post-Mapping Process operator from the Toolbox onto the Mapping Editor canvas.

   The Add Mapping Transformation dialog displays.

2. Choose **Select from existing repository Transformation and bind.**
3. Locate the OLAP packages from the Warehouse Builder navigation tree. See the steps listed in the previous section for more information on locating the packages.
4. Click **OK.**
5. The Post-Mapping Process transformation displays on the canvas.

   If you select the WB_OLAP_LOAD_CUBE procedure, you see the following attributes:

   - **OLAP_AW_OWNER:** Owner of the Analytic Workspace to be created or maintained.
   - **OLAP_AW_NAME:** Name of the Analytic Workspace to be created or maintained
   - **OLAP_CUBE_OWNER:** Owner of the cube.

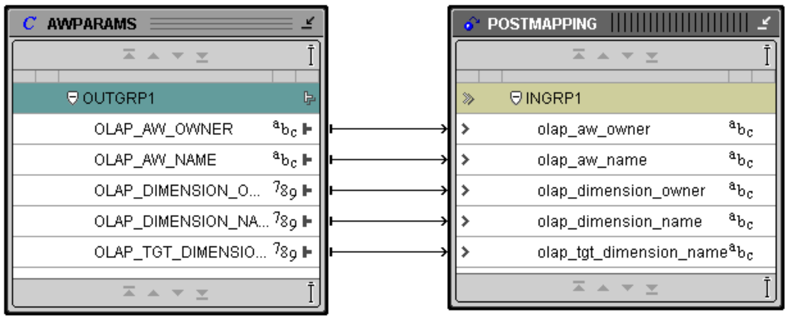
- **OLAP_CUBE_NAME:** Name of the cube.
- **OLAP_TGT_CUBE_NAME:** Name of the cube in the target.

If you select the WB_OLAP_LOAD_DIMENSION or WB_OLAP_LOAD_DIMENSION_GENUK procedure, you see the following attributes:

- **OLAP_AW_OWNER:** Owner of the Analytic Workspace to be created or maintained.
- **OLAP_AW_NAME:** Name of the Analytic Workspace to be created or maintained.
- **OLAP_DIMENSION_OWNER:** Owner of the dimension.
- **OLAP_DIMENSION_NAME:** Name of the dimension.
- **OLAP_TGT_DIMENSION_NAME:** Name of the dimension in the target system.

6. Drag and drop a Constant operator from the Toolbox onto the Mapping Editor canvas.
7. Right-click the Constant operator and select **Edit** from the pop-up menu.
8. From the Constant Editor, select the Output tab.
9. Define the same attributes for the Constant as defined for the Post-Mapping Process.
10. Map the attributes from the Constant operator to the attributes in the Post-Mapping Process as shown in Figure 22–15.

Figure 22–15 Defining a Post-Mapping Process to Load an AW

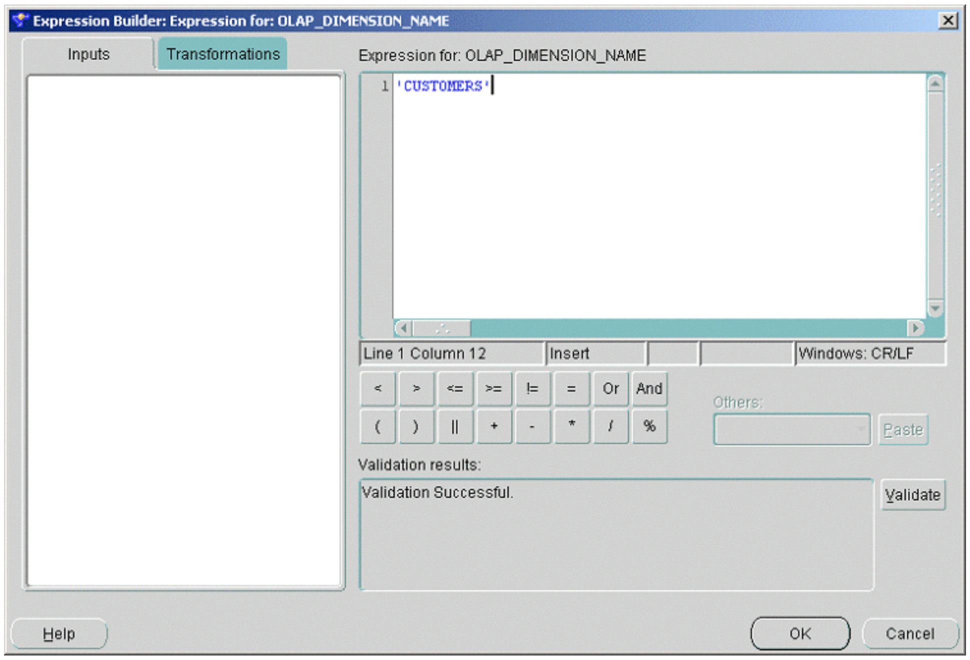


11. From the Constant operator, right-click each attribute and select **Attribute Properties.**

    The Attribute Properties dialog displays.
12. Click in the Expression field and click the **...** button.

    The Expression Builder displays.
13. Type the value for the attribute in the Expression Builder. For example, for the attribute OLAP_DIMENSION_NAME as shown in Figure 22–16, type 'CUSTOMERS', which is the name of the dimension to be deployed.

Figure 22–16 Expression Builder


Expression Builder: Expression for: OLAP_DIMENSION_NAME

Expression for: OLAP_DIMENSION_NAME

```
'CUSTOMERS'
```

Line 1 Column 12 | Insert | Windows: CR/LF

Validation results:

Validation Successful.

14. Continue this process until you assign expressions to all the attributes in the Constant operator.

    You are now ready to validate the mapping and to generate PL/SQL code. After you generate the PL/SQL, you can run the mapping to load the cubes and dimensions in your OLAP instance. Warehouse Builder loads a fully resolved cube, if you choose, in the Analytic Workspace.

Creating a Process Flow to Load an Analytic Workspace

You can also create a process flow to load your OLAP dimensions and cubes into the AW. Design a process flow containing the PL/SQL routines provided with Warehouse Builder.

To locate the PL/SQL packages in Warehouse Builder:

1. Create a process flow. For more information, see Chapter 10, "Designing Process Flows".
2. Drag and drop the Transform activity onto the canvas.

   Warehouse Builder displays a dialog for you to select a transformation.
3. Expand the PUBLIC node.
4. Expand the Pre-Defined node and then expand the OLAP node.

   Warehouse Builder displays the wrapper procedures: WB_OLAP_LOAD_CUBE, WB_OLAP_LOAD_DIMENSION, and WB_OLAP_LOAD_GENUK.
5. Select one of these packages.
6. Define the parameters for the dimensions and cubes in the Activity View panel.

If you selected the WB_OLAP_LOAD_CUBE package, you will see the following parameters. Type the value for each of these parameters in the **Value** column. Optionally, type a description for each parameter in the Description column.

- **OLAP_AW_OWNER:** Owner of the Analytic Workspace to be created or maintained.
- **OLAP_AW_NAME:** Name of the Analytic Workspace to be created or maintained
- **OLAP_CUBE_OWNER:** Owner of the ROLAP cube.
- **OLAP_CUBE_NAME:** Name of the ROLAP cube.
- **OLAP_TGT_CUBE_NAME:** Name of the cube in the target Analytic Workspace.

If you selected the WB_OLAP_LOAD_DIMENSION procedure, you will see the following parameters. Type the value for each of these parameters in the **Value** column. Optionally, type a description for each parameter in the Description column.

- **OLAP_AW_OWNER:** Owner of the Analytic Workspace to be created or maintained.
- **OLAP_AW_NAME:** Name of the Analytic Workspace to be created or maintained.
- **OLAP_DIMENSION_OWNER:** Owner of the ROLAP dimension.
- **OLAP_DIMENSION_NAME:** Name of the ROLAP dimension.
- **OLAP_TGT_DIMENSION_NAME:** Name of the dimension in the target Analytic Workspace.

You are now ready to validate the process flow and to generate XPDL code. After you generate the code, you can run the process flow to load the cubes and dimensions in your OLAP instance. Warehouse Builder loads a cube in the Analytic Workspace and the OLAP routines create dynamic aggregator maps for resolving the cube.

If you need to perform any level of precalculated aggregation or incremental refreshes, see the *OLAP User's Guide* for details on operations in the DBMS_AWM package.

Debugging the Cloning in a Mapping or Process Flow

To access the information required to debug the cloning transformation, you can build a PL/SQL wrapper to customize the required logging.

To create a PL/SQL wrapper, create public transformations containing the same signatures as the ones provided by Warehouse Builder: WB_OLAP_LOAD_DIMENSION and WB_OLAP_LOAD_CUBE. The implementation of these transformations can use the OLAP routines to enable logging;

```
CWM2_OLAP_MANAGER.BEGIN_LOG(<directory_alias>, <filename>); CWM2_OLAP_MANAGER.END_
LOG;
```

These transformations can be used to start or end a log file. During these calls, you can invoke the appropriate Warehouse Builder routine to load the dimension or cube. For example, to trace the dimension load you can create a procedure with the following block of code using the directory object

```
'MY_OLAP_TRACE';
CWM2_OLAP_MANAGER.BEGIN_LOG('MY_OLAP_TRACE', olap_dimension_name || '.log'); OWB_
OLAP_LOAD_DIMENSION(olap_aw_owner, olap_aw_name, olap_dimension_owner, olap_
dimension_name, olap_tgt_dimension_name);
CWM2_OLAP_MANAGER.END_LOG;
```

This code provides a level of trace useful for debugging issues in the data load.

Notes on the Warehouse Builder OLAP Bridge (9.0.5)

The Warehouse Builder OLAP best practices integrate the design definitions in Warehouse Builder, the Warehouse Builder OLAP bridge, and the OLAP metadata it generates. The minimal requirement for an OLAP dimension is that the lowest level in the hierarchy have a level attribute nominated for the description. This can be achieved by using the best practices.

Execution Modes for the Warehouse Builder OLAP Bridge

There are two types of execution modes available for the Warehouse Builder OLAP bridges: Standard Mode and Debug Mode. These modes are discussed in the following sections.

Standard Mode: Deploying PLSQL from Bridge

If you deploy the PLSQL scripts directly from the bridge, it will::

- Load the ROLAP catalog metadata in the OLAP catalog.
- Clone the ROLAP cube in the AW (MOLAP solution - optional).
- Enable the MOLAP cube stored in AW with OLAPI by executing the OLAP generated scripts to create metadata in CWM2 (optional).
- Create materialized views for the ROLAP cube by executing the OLAP generated scripts to generate Materialized Views (optional).

Deploy ROLAP Selecting **No** for AW clone, OLAP API generation, and MV generation will generate only the ROLAP OLAP catalog metadata. Switch off these options to obtain the ROLAP model before cloning.

Clone AW Selecting **Yes** for AW clone will create a MOLAP clone of the ROLAP cube in an AW.

Enable OLAPI Specify a valid directory on the client for the directory parameter. The OLAP API enabler scripts will be generated on the client under this directory. The bridge will execute these scripts and a script will be generated for each dimension and cube in the collection. In the script, the physical name of the object will be suffixed with .sql and will reside in the directory on the client file system. A drop script will also be generated for each dimension and cube in the collection, with the physical name of the object suffixed with _drop.sql. This script will also reside in the directory on the client file system. This can be used for removing the SQL object types and views.

Materialized View Generation Specify a valid directory in the client for the directory parameter. The MV scripts will be generated on the client under this directory. The bridge will execute these scripts and a script will be generated for each dimension and cube in the collection. The script will suffix the physical name of the object with _ mv.sql and will be located in the directory on the client file system.

Debug mode: Deploying PLSQL manually

When you execute the generated PLSQL script from SQL*Plus, it will:

- Load the ROLAP catalog metadata in the OLAP catalog
- Clone the ROLAP cube in the AW (MOLAP solution - optional).

- Generate scripts to enable the MOLAP cube stored in the AW with OLAPI (optional).
- Generate scripts to define materialized views for the ROLAP cube (optional).

Deploy ROLAP If you select **No** for AW cloning, OLAP API generation, and MV generation, the bridge will generate only the ROLAP OLAP catalog metadata. This step must be completed before AW cloning. Switching off these options is useful for obtaining the right ROLAP model before cloning.

Clone AW Select **Yes** for AW clone to create a MOLAP clone of the ROLAP cube in an AW.

Enable OLAPI Specify a valid directory object for the directory parameter. The OLAP API enabler scripts will be generated on the server under this directory object. You are responsible for executing these scripts. A script will be generated for each dimension and cube in the collection, containing the physical name of the object suffixed with .sql. These scripts will be stored in the directory object location on the server file system. A drop script will also be generated for each dimension and cube in the collection. This drop script will contain the physical name of the object suffixed with _ drop.sql and will be stored in the directory object location on the server file system. This can be used for removing the SQL object types and views.

Materialized View Generation The user must specify a valid directory object for the directory parameter. The MV scripts will be generated on the server under the directory object. You are responsible for executing these scripts. A script will be generated for each dimension and cube in the collection, containing the physical name of the object suffixed with _mv.sql, located in the directory object location on the server file system.

Customizing the Bridge

You can customize the bridge according to the cloning of your AW. The customization can be done on a cube by cube basis using special tags inside the description for the cube in Warehouse Builder. These tags and their content will be removed from the final cube description in the OLAP catalog. The preload block can be used to define the segwidths of dimensions or to define composite dimensions in the AW. These structures can be used to improve performance from the AW. The postload block can be used to define calculated measures in the AW. The following two sets of tags can be used for customization:

```
<PRELOAD>
anonymous_plsql_block
</PRELOAD>
```

and

```
<POSTLOAD>
anonymous_plsql_block
</POSTLOAD>
```

The bridge generates an OLAP load specification with the name of the cube. The preload block can be used to augment this load specification (using the OLAP apis) to define a composite specification which can then be added to the load specification.

The following example uses the preload block that defines a new composite specification to create a composite dimension named CMP1 in the AW containing the dimensions CUSTOMERS and PRODUCTS with segwidth 555555.

```
     <PRELOAD>
     declare
       cnam varchar2(32);
       cub varchar2(30);
       cst varchar2(30);
     BEGIN
        cnam := 'ORDERS_CST';
        cub := 'ORDERS';
        cst := 'CMP1';
        begin
          dbms_awm.delete_awcomp_spec(cnam, USER, cub);
        EXCEPTION when others then null;
        end;

        dbms_awm.create_awcomp_spec(cnam, USER, cub);
        dbms_awm.add_awcomp_spec_member(cnam, USER, cub, 'T_TIME', 'DIMENSION',
USER, 'T_TIME');
        dbms_awm.set_awcomp_spec_member_seg(cnam, USER, cub, 'T_TIME', 7777777);
        dbms_awm.add_awcomp_spec_member(cnam, USER, cub, cst, 'COMPOSITE');
        dbms_awm.add_awcomp_spec_comp_member(cnam, USER, cub, cst, cst||'_
CUSTOMERS',
     'DIMENSION', USER, 'CUSTOMERS');
        dbms_awm.add_awcomp_spec_comp_member(cnam, USER, cub, cst, cst||'_
PRODUCTS', 'DIMENSION',
     USER, 'PRODUCTS');
        dbms_awm.set_awcomp_spec_member_seg(cnam, USER, cub, cst, 555555);
        dbms_awm.add_awcubeload_spec_comp(cub, USER, cub, cnam);
        EXCEPTION
          WHEN OTHERS THEN raise;
     END;
     </PRELOAD>
```

This will add the composite specification 'ORDERS_CST' to the load specification 'ORDERS' created by the bridge. This code is invoked before the OLAP refresh cube routine. See the *OLAP User Guide* for details on the OLAP apis and their functionality.

The postload block is executed after the cube has been created in the AW. It can be used for to define new calculated measures in the AW based on the objects created during the bridge run. The AW is then updated and committed at the end of the bridge run.

Cloning Behavior (Incremental Loading, Precalculated Aggregation)

The OLAP cloning routines used by Warehouse Builder and the ones exposed as transformations are the base routine for cloning ROLAP cubes in the AW and for refreshing the data. The refresh defined in these routines is a complete load of the dimension or fact table. The following sections discuss how to perform incremental loading and pre-calculated aggregations.

Incremental Loading

The default cloning and load behavior of the Warehouse Builder OLAP integration loads entire sets of data. To perform incremental loading of the AW, you need to use OLAP routines and define filters (SQL WHERE clauses) for refining the actual rows to load from the dimension or cube. The transformation ORDERS_LOAD_FILTER enables you to use the OLAP filter routines for loading only a select set of rows (where month_id > 33) from the ROLAP fact table. The follow example shows the use of the OLAP APIs.

```
procedure ORDERS_LOAD_FILTER
BEGIN
 dbms_awm.create_awcubeload_spec ('ORDERS_FIL', USER, 'ORDERS', 'LOAD_DATA');

 -- Define the where clause for the load specification
 dbms_awm.Add_AWCubeLoad_Spec_Filter('ORDERS_FIL',USER,'ORDERS',USER,'ORDERS','
month_id>33');
 dbms_awm.refresh_awcube (USER, 'AWS', 'AWORDERS', 'ORDERS_FIL');

    EXCEPTION
        WHEN OTHERS THEN
            NULL;  -- enter any application specific exception code here
END;
```

Pre-Calculated Aggregation

By default, during cloning the Warehouse Builder OLAP bridge creates runtime aggregation maps (AGGMAPs) in the AW that are used to fully resolve the cube. If you want to optimize the cube by performing any customized pre-calculated aggregations, then the OLAP APIs can be used to build such a definition. The custom aggregation can be executed as an activity in a process flow.

In the following example, the transformation ORDERS_BUILD uses the OLAP routines to pre-calculate to a certain level in the cube in the AW. The following example shows you how to use the OLAP APIs.

```
procedure ORDERS_BUILD
  aggspec VARCHAR2(32);
  AW_NAME VARCHAR2(32);
  CUBE_NAME VARCHAR2(32);
BEGIN
  aggspec := 'AGGORDERS';
  AW_NAME := 'TESTAW';
  CUBE_NAME := 'AWORDERS';

  dbms_awm.create_awcubeagg_spec(aggspec, USER, AW_NAME, CUBE_NAME);

  -- Only aggregate city and customer for the CUSTOMERS dimension
  dbms_awm.add_awcubeagg_spec_level(aggspec, USER, AW_NAME, CUBE_NAME,
'AWCUSTOMERS', 'CITY');
  dbms_awm.add_awcubeagg_spec_level(aggspec, USER, AW_NAME, CUBE_NAME,
'AWCUSTOMERS',
'CUSTOMER');

  -- Only aggregate day and month for the time dimension
  dbms_awm.add_awcubeagg_spec_level(aggspec, USER, AW_NAME, CUBE_NAME, 'AWT_TIME',
'DAY');
  dbms_awm.add_awcubeagg_spec_level(aggspec, USER, AW_NAME, CUBE_NAME, 'AWT_TIME',
'MONTH');

  dbms_awm.add_awcubeagg_spec_measure(aggspec, USER, AW_NAME, CUBE_NAME,
'AMOUNT');

  -- Now build the cube. This may take some time on large cubes.
  dbms_awm.aggregate_awcube(USER, AW_NAME, CUBE_NAME, aggspec);

   EXCEPTION
        WHEN OTHERS THEN
            NULL;  -- enter any application specific exception code here
END;
```

Warehouse Builder Bridges: Transfer Parameters and Considerations

This section includes the following topics:

- Transfer Parameters on page 22-28
- Transfer Considerations on page 22-34
- Importing Metadata from an Object Management Group CWM Standard System on page 22-34
- Importing Metadata from Computer Associates ERwin 3.5.1 on page 22-35
- Importing Metadata from Powersoft PowerDesigner 6.0 on page 22-35
- Importing Metadata from Oracle9i OLAP Server on page 22-36
- Exporting Metadata to Oracle Discoverer 4i and 9i on page 22-37
- Importing Transferred Data into Discoverer on page 22-40
- Exporting Metadata to an Object Management Group CWM Standard System on page 22-41
- Exporting Metadata to Oracle Express on page 22-42
- Exporting Metadata to Oracle9i OLAP Server on page 22-43

Transfer Parameters

The Warehouse Builder Transfer Wizard enables you to import metadata intoWarehouse Builder from data warehousing tools and file systems such as Object Management Group Common Warehouse Metamodel, Computer Associates ERwin, Powersoft PowerDesigner, and Oracle9*i* OLAP Server. It also enables you to export the metadata from Warehouse Builder to Oracle tools such as Oracle Discoverer, Oracle Express, and Oracle9*i* OLAP Server.

When you use the Warehouse Builder Transfer Wizard, you must provide transfer parameter information based upon the metadata source or target you select. Table 22–4 lists the transfer parameters for each data source or target type.

Table 22–4 Transfer Parameters for OLAP Server Import

Transfer Parameter Name	Description
Username	Username to access the OLAP server instance from where you want to import metadata.
Password	Password to access the OLAP server instance from where you want to import metadata.
Hostname	Host system for the OLAP server instance from where you want to import metadata.
Port	Port number for the OLAP server instance from where you want to import metadata.
Service Name	Service Name for the OLAP server instance from where you want to import metadata.
Measure Folder	Provide a name for the folder containing the measures that you want to import into Warehouse Builder. When you import the measures, Warehouse Builder creates a collection containing the cubes corresponding to these measures.

Table 22–4 (Cont.) Transfer Parameters for OLAP Server Import

Transfer Parameter Name	Description
Project	Type a name for the new project to be created in Warehouse Builder during the import. This parameter is based on your import mode. If you are replacing or updating the metadata, then you need to specify an existing project into which you want to import the metadata.
Default Module Type for Relational Schemas	Choose the type of module into which you want to import the metadata, source or warehouse. The transfer wizard will create the correct type of module and import the metadata into it.
Process OLAP Physical Representation	If you are importing metadata from a star schema, then you can choose Yes if you want to maintain its physical representation or No to only import its logical definitions. If you are importing a snowflake schema, its physical implementation is lost and only the logical definitions are imported into Warehouse Builder.

Table 22–5 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 **...** to browse for the file location.
CWM Project	Specify the file containing the OMG CWM model you want to import. Click **...** to browse for the file location.
Default Module Type for Relational Schemas	Choose the type of module into which you want to import the metadata, source or warehouse. The transfer wizard will create the correct module and import the metadata into it.
Process OLAP Physical Representation	If you are importing metadata from a star schema, then you can choose Yes if you want to maintain its physical representation or No to only import its logical definitions. If you are importing a snowflake schema, its physical implementation is lost and only the logical definitions are imported into Warehouse Builder.
Import Mode	Select an import mode: create, replace, update, or incremental update.
Log Level	Log level to be assigned to the transfer: Errors, Information, and Trace. **Error:** lists the errors, if any, generated by 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.

Table 22–6 Transfer Parameters for Importing from CA ERwin

Transfer Parameter Name	Description
Input Model File	Select the file that was exported from ERwin. Click ... to browse for the file location.
Warehouse Builder Project	Type a name for the newly imported Project that will be located under the Project node in Warehouse Builder. This parameter is based on your import mode. If you are replacing or updating the metadata, then you need to specify an existing project into which you want to import the metadata.

Table 22–6 (Cont.) Transfer Parameters for Importing from CA ERwin

Transfer Parameter Name	Description
Warehouse Builder Module Name	Choose the type of module into which you want to import the metadata, source or warehouse. The transfer wizard will create the correct module and import the metadata into it.
Default Module Type for Relational Schemas	Name matching mode you want to utilize during the import. Physical name mode is the default.
Process OLAP Physical Representation	If you are importing metadata from a star schema, then you can choose Yes if you want to maintain its physical representation or No to only import its logical definitions. If you are importing a snowflake schema, its physical implementation is lost and only the logical definitions are imported into Warehouse Builder.
Import Mode	Select an import mode: create, replace, update, incremental update.
Log Level	Log level to be assigned to the transfer: Errors, Information, and Trace. **Error:** list the errors, if any, generated by 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.

Table 22–7 Transfer Parameters for Importing from Powersoft PowerDesigner

Transfer Parameter Name	Description
Input Model File	Select the file that was exported from PowerDesigner. Click **...** to browse for the file location.
Warehouse Builder Project	Type a name for the newly imported Project that will be located under the Project node in Warehouse Builder. This parameter is based on your import mode. If you are replacing or updating the metadata, then you need to specify an existing project into which you want to import the metadata.
Warehouse Builder Module Name	Type the name of the Warehouse Builder module that will be created during import.
PDM Name Mapping	Choose whether you want to use the logical (NAME) name or physical (CODE) name for the Physical Data Model (CDM). Default is CODE.
Submodel Mapping	Name of the submodel you want to import into Warehouse Builder. The default choice (*) imports all the sub models within the model
Default Module Type for Relational Schemas	Name matching mode you want to utilize during the import. Physical name mode is the default.
Process OLAP Physical Representation	If you are importing metadata from a star schema, then you can choose Yes if you want to maintain its physical representation or No to only import its logical definitions. If you are importing a snowflake schema, its physical implementation is lost and only the logical definitions are imported into Warehouse Builder.
Import Mode	Select an import mode: create, replace, update, incremental update.

Table 22–7 (Cont.) Transfer Parameters for Importing from Powersoft PowerDesigner

Transfer Parameter Name	Description
Log Level	Log level to be assigned to the transfer: Errors, Information, and Trace. **Error:** list the errors, if any, generated by 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.

Table 22–8 Transfer Parameters for Exporting Metadata to OMG CWM

Transfer Parameter Name	Description
Warehouse Builder Exported Collections	Select the collections you want to export from Warehouse Builder. The default is All Collections. You can also select individual Collections from the drop-down list.
OMG CWM Output File	Type an output file for the exported Collection. Click the field and the **...** button to browse and select a file.
Log Level	Enter the log level to be assigned to the transfer: Errors, Information, Trace. **Errors:** produces just a list of errors, if any, generated by 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.
Warehouse Builder Translated Language	Select the translated language to export. The default will be the base MLS language in the repository. You can choose to select a specific language, or you can select All to export all supported languages.

Table 22–9 Transfer Parameters when Exporting Metadata to Discoverer

Transfer Parameter Name	Parameter Description
Warehouse Builder Exported Collections	Select the collections you want to export from Warehouse Builder. The default is All Collections. You can also select individual Collections from the drop-down list
Discoverer EUL Owner	Name of the owner of the Discoverer EUL.
Discoverer Schema Owner	Name of the owner of the Discoverer schema.
Dimensional Reuse	True or False option for dimensional reuse (defined as two or more foreign key constraints in a cube that point to the same dimension). Select **True** if you changed the logical names of foreign key columns and you want them to appear as separate folders in Discoverer.

Table 22–9 (Cont.) Transfer Parameters when Exporting Metadata to Discoverer

Transfer Parameter Name	Parameter Description
Discoverer Output File	Path and name of the Discoverer .EEX file (which will contain the metadata) that is generated by the Transfer Wizard to be imported into Discoverer. Enter the path or select it by clicking Browse in the Transfer Parameter Value column of the Discoverer Output File parameter. Create a name of your choosing for the .EEX file and enter it at the end of the path.
Log Level	The log displays data about successful and unsuccessful transfers. The Transfer Wizard provides two additional log levels, Errors and Trace, to assist you in debugging. **Errors:** produces just a list of errors, if any, generated by 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.
OWB Translated Language	Select the translated language to export. The default will be the base MLS language in the repository. You can choose to select a specific language, or you can select All to export all supported languages. An EEX file will be created for each language.

Table 22–10 Transfer Parameters when Exporting Metadata to Express

Transfer Parameter Name	Parameter Description
Warehouse Builder Exported Collections	Select the collections you want to export from Warehouse Builder. The default is All Collections. You can also select individual Collections from the drop-down list
Express User	User id for connecting to the Express Repository.
Express User Password	Password for the Express User.
Express Connect String	Connecting odbc string for Express, using the following syntax: `host_machine_name:port_number:database_sid`
Express Table Owner	Name of the Express Table Owner.
RAA Version Number	Relational Access Administrator version.
Log Level	The log displays data about successful and unsuccessful transfers. The Transfer Wizard provides two additional log levels, Errors and Trace, to assist you in debugging. **Errors:** produces just a list of errors, if any, generated by 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.

Table 22–10 (Cont.) Transfer Parameters when Exporting Metadata to Express

Transfer Parameter Name	Parameter Description
OWB Translated Language	Select the translated language to export. The default will be the base MLS language in the repository. You can choose to select a specific language, or you can select All to export all supported languages.

Table 22–11 Transfer Parameters when Exporting Metadata to Oracle9i OLAP server

Transfer Parameter Name	Parameter Description
Warehouse Builder Exported Collections	Select the collections you want to export from Warehouse Builder. The default is All Collections. You can also select individual Collections from the drop-down list
User name	The user name for the Oracle9*i* OLAP server.
Password	The password for the Oracle9*i* OLAP server.
Host name	The host name for the Oracle9*i* OLAP server.
Port	The port number for the Oracle9*i* OLAP server.
SID	The database SID for the Oracle9*i* OLAP server.
PL/SQL Output File	File name for the generated PL/SQL file. The PL/SQL file has one parameter which must be passed when executed, the parameter is the schema in which you are creating the OLAP objects into. So when executing the file 'sales_history.sql' into the schema 'SH', you can execute as 'sqlplus OLAPDBA/<passwd>@sales_history.sql SH'
Deploy PL/SQL in Database	Yes or No. Execute the generated PL/SQL in the database, or save the PL/SQL for scheduled execution.
Log Level	The log displays data about successful and unsuccessful transfers. The Transfer Wizard provides two additional log levels, Errors and Trace, to assist you in debugging. **Errors:** produces just a list of errors, if any, generated by 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.
OWB Translated Language	Select the translated language to export. The default will be the base MLS language in the repository. You can choose to select a specific language, or you can select All to export all supported languages.
Deploy to AW	Specify whether you want to deploy the metadata to the Analytic Workspace or not.
AW Name	Name of the Analytic Workspace to be created or maintained.
AW Object Prefix	Provide the name of the Analytic Workspace where you want to deploy the metadata.
Load Cube Data	Loads the cube data into the AW. If this option is not selected, only the cube loading program will be loaded.
Generate Surrogate Keys for Dimensions	Generate surrogate keys for dimensions if your dimension values are not unique across all levels. This option can be used to generate surrogate keys.

Table 22–11 (Cont.) Transfer Parameters when Exporting Metadata to Oracle9i OLAP

Transfer Parameter Name	Parameter Description
Generate View Definitions	Specify whether you want to generate view definitions for the Analytic Workspace. This option will also generate OLAP metadata for the cube based on the SQL view for the Analytic Workspace.
Generate Materialized Views	Indicate whether you want to generate materialized views for the relational implementation of the cube.
Generated View Directory	Specify the name of a directory to store the generated view scripts. If you choose the Deploy PL/SQL in Database option, this directory is located on the client side. Otherwise this is a server side directory object.

Transfer Considerations

Before you transfer metadata into the Warehouse Builder repository or out of the Warehouse Builder repository, you need to perform tasks within the source and target tools to ensure that the metadata can be transferred successfully.

This section provides instructions for transferring metadata from a source tool to the Warehouse Builder repository or from the Warehouse Builder repository to a target tool. It also lists the objects that are extracted from the source tool during the transfer and their corresponding Warehouse Builder objects.

The Warehouse Builder Transfer Wizard exports metadata from the Warehouse Builder repository as collections. Before you export metadata, you must create collections within the Warehouse Builder repository.

This section includes the following topics:

- Importing Metadata from an Object Management Group CWM Standard System
- Importing Metadata from Computer Associates ERwin 3.5.1
- Importing Metadata from Powersoft PowerDesigner 6.0
- Importing Metadata from Oracle9i OLAP Server
- Exporting Metadata to Oracle Discoverer 4i and 9i
- Exporting Metadata to an Object Management Group CWM Standard System
- Exporting Metadata to Oracle Express
- Exporting Metadata to Oracle9i OLAP Server

Importing Metadata from an Object Management Group CWM Standard System

Table 22–12 lists the object conversion from an Object Management Group (OMG) Common Warehouse Metamodel (CWM) file system to the Warehouse Builder repository.

Table 22–12 Object Management Group CWM Standard Conversions

Object in Warehouse Builder	Object Exported to OMG CWM
Project	Package
Module	Schema
Dimension	Dimension

Table 22–12 (Cont.) Object Management Group CWM Standard Conversions

Object in Warehouse Builder	Object Exported to OMG CWM
Level	Level
Level Attributes	Attribute
Hierarchy	Hierarchy
Cube	Cube
Cube Attributes	Measure
Table	Table
Column	Column
Foreign Key	Foreign Key
Unique Key	Unique Constraint/Primary Key
View	View
Column	Column

Importing Metadata from Computer Associates ERwin 3.5.1

While importing a file from ERwin into the Warehouse Builder repository, save the file containing the metadata for transfer as an .ERX file.

Table 22–13 lists the object conversion from ERwin to the Warehouse Builder repository.

Table 22–13 Object Conversion for Import from ERwin into Warehouse Builder

Object in ERwin	Imported into Warehouse Builder
Table	Table
Column	Columns
Foreign Key	Foreign Keys
Primary Key	Unique Keys
View	View
Column	Columns

Importing Metadata from Powersoft PowerDesigner 6.0

While importing a file from PowerDesigner into the Warehouse Builder repository, save the file containing the metadata for transfer as a .PDM file.

Table 22–14 lists the object conversion from PowerDesigner to the Warehouse Builder repository.

Table 22–14 Object Conversion for Import from PowerDesigner into Warehouse Builder

Object in PowerDesigner	Imported into Warehouse Builder
Table	Table
Column	Columns
Index	Foreign Keys
Index	Unique Keys

Table 22–14 (Cont.) Object Conversion for Import from PowerDesigner into Warehouse

Object in PowerDesigner	Imported into Warehouse Builder
View	View
Column	Columns

Importing Metadata from Oracle9*i* OLAP Server

Table 22–15 lists the object conversion from Oracle9*i* OLAP server to the Warehouse Builder repository.

Table 22–15 Object Conversion for Import from Oracle9i OLAP server into Warehouse Builder

OLAP Object in Oracle9*i*	Imported into Warehouse Builder
Schema	Module
Cube	Cube
Measure	Measure
Dimension	Dimension
Level	Level
Hierarchy	Hierarchy
Attribute	LevelAttribute
MeasureFolder	Collection

There are certain structures that Warehouse Builder does not support that are possible in the Oracle9*i* OLAP server. For example, Warehouse Builder does not support snowflake modelling and cubes based upon views. If you have OLAP definitions that use these structures, you can import the logical definition of these objects, but not their physical representation.

Notes on the OLAP Import

This section describes guidelines for importing cubes and dimensions from an OLAP server.

The level identifier columns that have been generated from Warehouse Builder with a prefix are imported with that prefix. Because these columns are not modelled as relational dimension level attributes, there are no names in the database without the prefix. Therefore, when a you import dimension, the level identifying columns which have Warehouse Builder 'level attributes' defined will have also have the prefix in their names.

You can manually remove these prefixes. For example, if you define a dimension CUSTOMERS in Warehouse Builder with a level COUNTRY and level prefix of CT, then this level has a level attribute identifying column called COUNTRY_ID. The generated code maps COUNTRY to CT_COUNTRY_ID and the prefix is incorporated in the column name. When you import it, the level attribute defined in Warehouse Builder is named CT_COUNTRY_ID.

If set to **True,** the bridge parameter Process OLAP Physical Representation imports the physical structure of the relational dimension for star schemas. If the cube is a snowflake, then Warehouse Builder imports the logical definition of the cube and discards the physical implementation is discarded regardless of the setting. If a cube refers to dimensions across schemas, then those dimensions are ignored.

Exporting Metadata to Oracle Discoverer 4*i* and 9*i*

When you export a file from the Warehouse Builder repository to a Discoverer target using the Warehouse Builder Transfer Wizard, you can refer to the following terminology matrix to check how objects in Warehouse Builder repository are mapped in Discoverer. Table 22–16 lists the object conversion from Warehouse Builder repository to Discoverer.

Table 22–16 Object Conversion from Warehouse Builder to Discoverer

Object in Warehouse Builder	Object Exported to Discoverer
Dimension	Folder
Level	Hierarchy Node
Level Attributes	Item
Hierarchies	Hierarchy
Cube	Folder
Cube Attributes	Item
Table	Folder
Columns	Item
Unique Keys	Key
View	Folder (View)
Columns	Item
Attributes flagged as Item Classes	Item Classes
Collection	Business Area

Before you transfer metadata to Discoverer, you need to perform some additional tasks within Warehouse Builder to ensure that the metadata transfers successfully and displays appropriately in Discoverer. These tasks include:

- Configuring Warehouse Builder for dimensional reuse.
- Creating collections in a Warehouse Builder warehouse module so that the transferred metadata displays as business areas within Discoverer.
- Hiding some Warehouse Builder table columns so that they are grayed in Discover Administration Edition and unavailable in Discoverer User Edition.
- Specifying attributes to use as Item Classes in Discoverer.
- Using names to indicate which attributes to place in the hierarchy nodes within Discoverer.

The following sections provide instructions for performing these tasks.

Configuring Warehouse Builder for Dimensional Reuse

Within Warehouse Builder, a cube can contain two or more foreign key constraints pointing to a single dimension, such as ordered date and shipped date for a calendar dimension. This is referred to as dimensional reuse. Within Discoverer, each of these constraints needs its own folder so the dimension roles display properly. Discoverer requires these dimension roles to facilitate query building. In the calendar example, one folder should display for ordered date and another one for shipped date.

Note: Follow these steps only if you are going to set the Dimensional Reuse parameter value to TRUE in the Warehouse Builder Transfer Wizard.

To enable dimensional roles to display correctly in Discoverer, you need to create dummy tables in Warehouse Builder. This dummy table becomes the dimension role and the columns in the table become the columns that the cube dimensions use.

The recommended naming convention for the dummy table is

```
<dimension reference>##<role reference>
```

where the dimension and role references help you remember what role on which dimension this table represents. The logical name of the table is the role name. To further identify the dummy table, mark it as Deployable false.

The dummy table to real dimension association relies on any one of the contained columns. The dummy column to cube association is determined by making the column's physical name the same as the physical name of the foreign key constraint in the cube.

The following example illustrates how you can set up dimensional roles for exporting to Discoverer:

Defining Dimensions and Cubes in Warehouse Builder

1. Create dimensions–DAY, PRODUCT and OPERATOR–in the normal way.
2. Create cubes–SALES and FLIGHTS–in the normal way.
3. Create the following constraints:
   - SALES to DAY (for ORDER_DATE) Constraint Physical Name is SALES_DAY_FK
   - SALES to DAY (for SHIP_DATE) Constraint Physical Name is SALES_DAY2_FK
   - SALES to PRODUCT (for PURCHASE) Constraint Physical Name is SALES_PROD_FK
   - FLIGHTS to DAY (for ORDER_DATE) Constraint Physical Name is FLIGHTS_DAY_FK
   - FLIGHTS to PRODUCT (for TICKET) Constraint Physical Name is FLIGHTS_PROD_FK
   - FLIGHTS to OPERATOR Constraint Physical Name is FLIGHTS_OPERATOR_FK

Defining the Dummy Tables

Create the dummy tables as follows using the Table editor:

1. For the DAY (ORDER_DATE) role, define the table as follows:
   - Physical Name DAY##ORD (logical name ORDER_DATE)
   - Column Physical Name SALES_DAY_FK
   - Column Physical Name FLIGHTS_DAY_FK
2. For the DAY (SHIP_DATE) role, define the table as follows;

- Physical Name DAY##SHIP (logical name SHIP_DATE)
- Column Physical Name SALES_DAY2_FK

3. For the PRODUCT (PURCHASE) role, define the table as follows:
   - Physical Name PROD##PUR (logical name PURCHASE)
   - Column Physical Name SALES_PROD_FK
4. For the PRODUCT (TICKET) role, define the table as follows:
   - Physical Name PROD##TIC (logical name TICKET)
   - Column Physical Name FLIGHTS_PROD_FK
5. Right-click the dummy table, select **Configure** from the pop-up menu, and set the Deployable value to **False.**

In order to use a specific level attribute in a hierarchy, the physical name of the level attribute must be one of the following:

- <level name>_NAME
- NAME

For example: a product dimension has one hierarchy p_hierarchy that consists of 3 levels with the following attributes (physical names):

PRODUCT_TYPE:

- T_ID (unique key (level))
- T_NAME
- T_DESCRIPTION

PRODUCT_CATEGORY

- C_ID (unique key (level))
- C_NAME
- C_DESCRIPTION

PRODUCT

- P_ID (unique key (level); physical primary key)
- P_NAME
- P_DESCRIPTION

The default bridge behavior is to build a hierarchy based on T_ID, C_ID and P_ID. In order to (for example) use the names instead of the IDs, define the following levels with physical attribute names:

PRODUCT_TYPE

- T_ID (unique key (level))
- PRODUCT_TYPE_NAME (or NAME)
- T_DESCRIPTION

PRODUCT_CATEGORY

- C_ID (unique key (level))
- PRODUCT_CATEGORY_NAME (or NAME)
- C_DESCRIPTION

PRODUCT

- P_ID (unique key (level); physical primary key)
- PRODUCT_NAME (or NAME)
- P_DESCRIPTION

The bridge will in this case use the (...)NAME attributes as hierarchy node attributes.

Hiding Data Prior to Transfer

You can hide specific Warehouse Builder table columns (such as obsolete or unused columns) so they are grayed in the Discoverer Administration Edition and unavailable in the Discoverer User Edition. To hide these columns, you create a custom entity attribute set in Warehouse Builder.

To hide table columns in Warehouse Builder:

1. From the main Warehouse Builder navigator tree, double-click the warehouse target module in which you want to hide table columns prior to transfer to Discoverer.

   The Warehouse Module Editor displays.
2. Expand the Warehouse Module Editor navigation tree.
3. Right-click the cube or dimension table in which you want to hide columns and select **Properties** from the pop-up menu.

   The Properties window for that table displays.
4. Select the **Attribute Sets** tab on the Properties window.
5. Create a new Attribute Set for the export by clicking **Add** and setting the Attribute Type to be BRIDGE_TYPE.
6. Select the attributes to include within this set and click **Advanced.**

   The Advanced Attribute Set Properties dialog displays.
7. Choose the columns to hide in Discoverer by checking the corresponding box under the **Hidden** field.
8. You can also use this dialog to define Item Class and set default aggregation and default position of table columns within Discoverer.

Importing Transferred Data into Discoverer

After transferring the selected metadata using the Transfer Wizard, you import the .EEX file into Discoverer.

For detailed information on accessing the projects imported from Warehouse Builder in Discoverer 4*i* and 9*i*, refer to the *Oracle Discoverer Administration Guide.*

Dimensional Reuse Naming Conventions in Discoverer

When you transfer metadata from Warehouse Builder to Discoverer using the Transfer Wizard and you select TRUE for the Dimensional Reuse parameter, a flag is set so dimensional roles display correctly in Discoverer. For more information, see "Configuring Warehouse Builder for Dimensional Reuse" on page 22-37.

Dimension roles are required to make query building easier in Discoverer. When two tables have multiple joins between them, Discoverer needs to know which join to use the first time the two folders representing the tables are referred to in a query.

Subsequently, it always uses that join in further parts of the query. If the query requires different joins at different times, the only way to support it is to provide another folder based on the same table. This multi-join scenario is common in dimensional models.

The support for dimension roles provides Discoverer with a separate folder for each role of the dimension. For example, if a DAY dimension is joined to the SALES cube, once for ORDER_DATE and once for SHIP_DATE, two dimension role folders are created in Discoverer. If the same role is required for a different cube, the original role is reused. For example, if a FLIGHTS cube was also using the ORDER_DATE role, it would reference the same folder as the one referenced by the SALES cube.

The Discoverer navigation tree denotes dimensional reuse (two or more foreign key constraints pointing to the same dimension in Warehouse Builder) in two ways:

- Warehouse Builder dimension folder names may appear in the following format:

```
DIMENSION LOGICAL NAME : RoleName
```

- When no role is defined, a default role is generated in Discoverer:

```
DIMENSION LOGICAL NAME : Default
```

Examples (following from the example in "Configuring Warehouse Builder for Dimensional Reuse") of the appearance of the folders in Discoverer:

DAY (hidden)

DAY : ORDER_DATE

DAY : SHIP_DATE

FLIGHTS

OPERATOR (hidden)

OPERATOR : Default (see preceding default role)

PRODUCT (hidden)

PRODUCT : PURCHASE

PRODUCT : TICKET

SALES

Exporting Metadata to an Object Management Group CWM Standard System

When you export metadata from the Warehouse Builder repository to an OMG CWM standard system, the Warehouse Builder Transfer Wizard converts it to a file that conforms to the OMG CWM standard. Table 22–17 lists the object conversion from the Warehouse Builder repository to the OMG CWM standard.

Table 22–17 Object Conversion from Warehouse Builder to OMG CWM

Object in Warehouse Builder	Object Exported to OMG CWM
Project	Package
Module	Schema
Dimension	Dimension
Level	Level
Level Attributes	Attribute
Hierarchy	Hierarchy

Table 22–17 (Cont.) Object Conversion from Warehouse Builder to OMG CWM

Object in Warehouse Builder	Object Exported to OMG CWM
Cube	Cube
Cube Attributes	Measure
Table	Table
Column	Column
Foreign Key	Foreign Key
Unique Key	Unique Constraint/Primary Key
View	View
Column	Column

Exporting Metadata to Oracle Express

After you export metadata from Warehouse Builder to Oracle Express, use the following steps to access the metadata within Express:

1. Open RAA.
2. Enter the user name and password for your RAA repository.
3. Open the project that you just transferred. After you run the bridge, the project will appear in the drop-down list.
4. Once the project is open, select **Express Database Maintenance** and enter the user name, password, and service name for your RAA repository.
5. Choose **Create Maintenance Procedure,** then provide the procedure and select general maintenance.

   The RDBMS login displays automatically.
6. Select **Generate Qualified Selects at Runtime.**
7. Select defaults for dimension processing.
8. Name your Express database and log file. These will be saved on the OES server machine under d:\orant\database\express_info.
9. Choose **OES** and then **OES Batch Manager** to monitor this job (Jobs -> Monitor).

   You have now created your raw Express database.

OSA Configuration

Before you look at the data, follow these configuration steps:

1. Open the OSA Application Manager.
2. Select **Database Setup** and then **Create.**

   You can now create a .dsc file. Provide a name for it that reflects the project you transferred. Choose edit and select remote thin-client. For the configuration file, go to the directory where you saved the database and log file and choose the .rdc file that relates to the project that you just transferred.
3. Select **Communication Setup,** then **Define,** and then **Create.**
4. Name your setup and choose thin-client, remote system. This is the server machine. For RPC, choose TCP/IP.

5. You can now open OSA and choose **New,** and then **Reports** or **Graphs** to view your data. You'll need to first select the correct dimensions, hierarchies, and measures based on your Warehouse Builder repository.

Exporting Metadata to Oracle9*i* OLAP Server

To deploy Oracle9*i* OLAP objects, you must first deploy warehouse objects from Warehouse Builder to the database server. To export additional OLAP metadata use a Metadata Export Bridge.

Table 22–18 lists the object conversion from the Warehouse Builder repository objects to OLAP objects in Oracle9*i*.

Table 22–18 Object Conversion for Export from Warehouse Builder into Oracle9i OLAP server

Object into Warehouse Builder	Exported OLAP Object
Cube	Cube
Measure	Measure
Dimension	Dimension
Level	Level
Hierarchy	Hierarchy
LevelAttribute	Attribute
Collection	MeasureFolder

Warehouse Builder generates a PL/SQL file after a metadata export to the Oracle9*i* OLAP Server. You can deploy this file to your server using SQL*Plus. For example, to execute the file sales_history.sql into the schema SH, you can execute the following:

```
sqlplus OLAPDBA/<passwd> @sales_history.sql SH
```

This defines the OLAP cube, additional dimension metadata and measure folders in the SH schema. The warehouse database objects must be deployed before running this script, otherwise the dependencies are not satisfied.

Part VI

Appendix

This part contains the following appendixes:

A
Keyboard Shortcuts

This appendix lists the keyboard options for accessing Warehouse Builder commands. For conventions that are supported by most applications designed for Windows 95 and Windows NT, refer to the Microsoft Windows Keyboard Guide. This appendix includes the following topics:

General Windows Keys

Table A–1 lists the keyboard sequences and descriptions for general window keys.

Table A–1 General Windows Keys

Keyboard Sequence	Description
F1	Launches the Help facility.
Esc	Close current window or menu and move Focus to parent (to whatever it was set before activating current window or menu). Currently Cancel command button enabled.
Delete	Deletes the selected items.
Tab	Moves Focus to next control.
Shift + Tab	Moves Focus to previous control.
Spacebar	Activates current push button when Focus is on Button; toggles checkbox to yes/no state.
Enter	Activates current push button when Focus is on Button.

Tree View Control Keys

Table A–2 lists the keyboard sequences and descriptions for tree view control keys.

Table A–2 Tree View Control Keys

Keyboard Sequence	Description
Right arrow	Expands tree on window canvas, when Focus set on objects with hierarchical relationships.
Left arrow	Collapse tree on window canvas, when Focus set on objects with hierarchical relationships.
Up/Down arrows	Selects the previous or next object.
Home	Moves Focus to the first control on window canvas.
End	Moves Focus to the last control on window canvas.

Accelerator Keys Set for Warehouse Builder

Accelerator keys are keyboard shortcuts for actions that are frequently performed. Accelerator keys enable you to bypass the menu by using a specific combination of keystrokes that perform the same function as a corresponding menu item. Table A–3 lists the keyboard sequences and descriptions for accelerator keys.

Table A–3 Accelerator Keys Set for Warehouse Builder

Keyboard Sequence	Description
Ctrl + N	Creates an object by invoking the New Wizards.
Ctrl + X	Cut text.
Ctrl + O	Invokes Object Editor.
Ctrl + C	Copy text.
Ctrl + R	Invokes Object properties.
Ctrl + V	Paste text.
Ctrl + F	Opens Object Find dialog.
Shift + Right Arrow	Highlight text.
Ctrl + S	Opens Commit Confirmation dialog.
Ctrl + P	Opens Print dialog.
Ctrl +Tab	Moves from grid to the next available control, chooses the shortcut page tab where available.

Menu Commands and Access Keys

Menu titles and menu items have underlined access keys. Press **Alt** with the access key to activate the control or menu anywhere within the active window. If an item does not have an underlined character, use up or down arrows to move the focus to the menu item and press **Enter**. Access keys can sometimes be used without the **Alt** key for choosing controls or menu items. Use access keys without **Alt** to select items from an open menu. Table A–4 lists the keyboard sequences and descriptions for menu commands and access keys.

Table A–4 Menu Commands and Access Keys

Keyboard Sequence	Description
Alt	Activates the menu bar of the active window. The first menu name on the left is selected.

Table A–4 (Cont.) Menu Commands and Access Keys

Keyboard Sequence	Description
Alt + any printing character	Activates the menu with the underlined character (access key) on the main menu bar.
Any printing character	Activates the menu with the underlined character (access key) on an open menu.
Left/Right arrows	Move the Focus between menus on the menu bar in the direction of the arrow. If the original menu was open, the target menu will be opened and the focus on the first item.
Up/Down arrows	Open the selected menu. Select previous and next command on the open menu.
Enter	Opens the selected menu when focus is on the menu title, activates a menu item when focus is on a menu item.
Alt + F4	Closes active window, returns Focus to the setting before activating current window or menu.
Shift + F10	Opens the shortcut menu for the active object (Focus is on object, item).

Mapping Editor Keyboard Operations

You can navigate the Mapping Editor canvas using the Tab key and the keyboard arrow keys.

The Tab key enables you to navigate to the next object on the canvas and select it. When no objects are selected, the system selects the node that is closest to the upper-left corner of the Mapping Editor canvas. The order of navigation is determined by the position in which the objects appear on the canvas. The system follows a Z navigation path.

Within a selected attribute group, you can navigate between attributes using the up and down keys.

When positioned on an input attribute, the left arrow key enables you to navigate to the incoming mapping line. There is only one incoming mapping line for each attribute. The right arrow key is not active.

When you select an object and then press the **Delete** key, the selected object is deleted from the canvas. You can delete the following objects:

- Operators. Warehouse Builder prompts you to confirm the delete before the delete occurs.
- Mapping lines from or to an attribute. You cannot delete mapping lines that start or end at an attribute group or header.

Auto-Mapping Mnemonic Key Assignments

To provide easy access to the elements of the Auto-Mapping Dialog, the following mnemonic keys have been assigned. Table A–5 lists the keyboard sequences and descriptions for mnemonic keys.

Table A–5 Auto-Mapping Mnemonic Key Assignments

Keyboard Sequence	Description
Alt + o	OK.

Table A–5 (Cont.) Auto-Mapping Mnemonic Key Assignments

Keyboard Sequence	Description
Alt + c	Cancel.
Alt + y	Copy source attributes to target group and match.
Alt + p	Match by position of source and target attributes.
Alt + n	Match by name.
Alt + i	Ignore case differences.
Alt + s	Ignore special characters.
Alt + e	Ignore source prefix.
Alt + u	Ignore source suffix.
Alt + t	Ignore target prefix.
Alt + a	Ignore target suffix.
Alt + g	Go.
Alt + d	Displayed mappings.
Alt + h	Help.

B Reserved Words

This appendix lists the reserved words that should not be used to name objects in Warehouse Builder.

Reserved Words

The following table lists reserved words. Do not use these words as physical object names within a Warehouse Builder Project.

■ ABORT	■ ACCEPT	■ ACCESS	■ ADD
■ ALL	■ ALTER	■ AND	■ ANY
■ ARRAY	■ ARRAYLEN	■ AS	■ ASC
■ ASSERT	■ ASSIGN	■ AT	■ AUDIT
■ AUTHORIZATION	■ AVG	■ BASE_ TABLE	■ BEGIN
■ BETWEEN	■ BINARY_ INTEGER	■ BODY	■ BOOLEAN
■ BY	■ CASE	■ CHAR	■ CHAR_BASE
■ CHECK	■ CLOSE	■ CLUSTER	■ CLUSTERS
■ COLAUTH	■ COLUMN	■ COMMENT	■ COMMIT
■ COMPRESS	■ CONNECT	■ CONSTANT	■ CRASH
■ CREATE	■ CURRENT	■ CURRVAL	■ CURSOR
■ DATA_BASE	■ DATABASE	■ DATE	■ DBA
■ DEBUGOFF	■ DEBUGON	■ DECIMAL	■ DECLARE
■ DEFAULT	■ DEFINITION	■ DELAY	■ DELETE
■ DELTA	■ DESC	■ DIGITS	■ DISPOSE
■ DISTINCT	■ DO	■ DROP	■ DUAL
■ ELSE	■ ELSIF	■ END	■ ENTRY
■ EXCEPTION	■ EXCEPTION_ INIT	■ EXCLUSIVE	■ EXISTS
■ EXIT	■ FALSE	■ FETCH	■ FILE
■ FLOAT	■ FOR	■ FORM	■ FROM
■ FUNCTION	■ GENERIC	■ GOTO	■ GRANT

- GROUP
- HAVING
- IDENTIFIED
- IF
- IMMEDIATE
- IN
- INCREMENT
- INDEX
- INDEXES
- INDICATOR
- INITIAL
- INSERT
- INTEGER
- INTERFACE
- INTERSECT
- INTO
- IS
- LEVEL
- LIKE
- LIMITED
- LOCK
- LONG
- LOOP
- MAX
- MAXEXTENTS
- MIN
- MINUS
- MLSLABEL
- MOD
- MODE
- MODIFY
- NATURAL
- NATURALN
- NEW
- NEXTVAL
- NOAUDIT
- NOCOMPRESS
- NOT
- NOWAIT
- NULL
- NUMBER
- NUMBER_BASE
- OF
- OFFLINE
- ON
- ONLINE
- OPEN
- OPTION
- OR
- ORDER
- OTHERS
- OUT
- PACKAGE
- PARTITION
- PCTFREE
- PLS_INTEGER
- POSITIVE
- POSITIVEN
- PRAGMA
- PRIOR
- PRIVATE
- PRIVILEGES
- PROCEDURE
- PUBLIC
- RAISE
- RANGE
- RAW
- REAL
- RECORD
- REF
- RELEASE
- REMR
- RENAME
- RESOURCE
- RETURN
- REVERSE
- REVOKE
- ROLLBACK
- ROW
- ROWID
- ROWLABEL
- ROWNUM
- ROWS
- ROWTYPE
- RUN
- SAVEPOINT
- SELECT
- SEPARATE
- SESSION
- SET
- SIZE
- SMALLINT
- SPACE
- SQL
- SQLCODE
- SQLERRM
- START
- STATEMENT
- STDDEF[1]
- STDDEV
- SUBTYPE
- SUCCESSFUL
- SUM
- SYNONYM
- SYSDATE
- TABAUTH
- TABLE
- TABLEDEF[1]
- TABLEPAT[1]
- TABLES
- TASK
- TERMINATE
- TIME
- THEN
- TO
- TRIGGER
- TRUE
- TYPE
- UID
- UNION
- UNIQUE
- UPDATE
- USE
- USER
- USERDEF[1]
- VALIDATE
- VALUES
- VARCHAR
- VARCHAR2
- VARIANCE
- VIEW
- WHEN
- WHENEVER
- WHERE
- WHILE
- WITH
- WORK
- WRITE
- XOR

[1] The words STDDEF, TABLEDEF, TABLEPAT, and USERDEF are only reserved if you have purchased the Oracle Warehouse Builder Name and Address component and are performing parse table maintenance.

C Using the XML Toolkit

This appendix describes how to retrieve and store data from XML documents in a target schema using the XML Toolkit. This toolkit can extract data from XML documents, which can reside in a variety of sources and formats, and store that data in Oracle Database tables, CLOB database columns, Advanced Queues, and other Oracle Database database objects.

This appendix includes the following topics:

Overview

An XML document conforms to the extensible Markup Language (XML) specification. XML allows developers to design their own customized markup documents which can be optimized for delivery of documents on the World Wide Web and to support implementation of e-commerce and numerous other applications.

The XML Toolkit is a set of PL/SQL procedures, functions, and packages that you can use to retrieve and load data from XML documents into Oracle Database database objects. For example, you can retrieve data from an XML document that resides in a file and load it into several tables that reside in a data warehouse.

Note: Due to current limitations, the XML Toolkit can only be used with relatively small files. Please refer to the Release Notes for more information.

Retrieving Data From Sources

You can retrieve data from XML documents that reside in the following:

- File
- Multiple files
- CLOB database column
- Raw Based Advanced Queue

- Object/CLOB based Advanced Queue
- URL

You identify the source of the data to the XML Toolkit using an element name defined by the Toolkit. For example, you identify a file with the element name file. Subsequent sections and examples describe individual element names, their respective attributes (if any), and the required syntax.

Storing Data in Targets

The XML Toolkit can store data retrieved from XML documents in a variety of Oracle Database database objects. The Toolkit can store data in the following objects:

- Table or multiple tables in a target Oracle Database database
- Updatable views
- Object tables
- Updatable object views
- Object/CLOB based Advanced Queues

You identify the target for the data using an element name defined by the Toolkit. For example, you identify a target table with the element name target.

The element name target has several attributes that provide control over runtime operations. For example, if the data extracted from the XML document does not match the target table, you can specify an XSL style sheet as an attribute that reformats the data accordingly. Subsequent sections and examples describe the individual element names, their respective attributes, and the required syntax.

Using Runtime Controls

During the extraction and load process, you can control when to commit the load as well as control the size of individual batches. These controls are specified using the element name `runtimeConfig`.

Calling the XML Toolkit

The XML Toolkit is implemented as a set of PL/SQL procedures and functions. To use the Toolkit, you must first create a Warehouse Builder transformation that invokes one of its procedures or functions. After the transformation has been created, a mapping can call the transformation using a pre-map or post-map trigger.

A typical scenario would be to create a transformation that extracts and loads data from an XML document into a staging table. The transformation could be generalized by referencing the XML document as a runtime parameter. After you create the transformation, you could then create a mapping that uses a pre-map process to call the transformation to load the staging table. The mapping could then transform the data in the staging table and load it into a target table. A post-map process could in turn truncate the staging table before the mapping terminates.

Two Entrances

A transformation can invoke the XML Toolkit using one of the following entry points into the API:

- The PL/SQL procedure **`wb_xml_load(control_file)`**

- The PL/SQL function **`wb_xml_load_f(control_file)`**

Both of these calls extract and load data from XML documents into database targets. The function, however, returns the number of documents read during the operation. 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 typically calls the transformation as a pre-map or post-map process.

The control file itself is an XML document, and the element name OWBXMLRuntime defines the top-level, or root element, of this document. The remaining element names which define the document sources and targets are self-explanatory.

After you create and name this transformation, you can reference it by name in a mapping. The most common reference for the transformation would be in a pre-map or post-map trigger.

Notes:

- The file names for the document source and the XSL style sheet are relative to the node that supports the Oracle Database database instance.
- The white space in the example control file that separates the text from the concatenate characters is unnecessary and is included only to improve readability.
- The control file is a well-formed XML document. See the control file's Document Type Definition (DTD) on page C-14.

Typical Control Files

The nine example control files in this section are well-formed and valid control files that extract and load data from the following sources into a database target:

1. A single file
2. A single file whose element names must be renamed
3. A single file to multiple targets
4. Multiple files
5. A large file split into several parts
6. CLOB column
7. URL
8. Raw Advanced Queue
9. Object Type Based Advanced Queue

The first five examples extract data from XML documents that reside in files; the remaining cases extract data from database objects or a document addressed by a URL. The examples cover all the sources that are managed by the XML Toolkit.

XML Documents Stored in Files

The XML Toolkit can extract data from documents stored in a single file or in multiple files when they reside in the same directory. The Toolkit can also extract data from these sources and store it into multiple tables.

Often, the data in an XML document fails to match the target object in which case you can reformat the data to match the target by including an XSL style sheet. The second example shows you how to reference an XSL style sheet by specifying a value for the XSLFile attribute for the target. The use of style sheets to reformat the source data is probably the most common case as the data from XML documents rarely match the column names of target tables in a data warehouse.

Finally, you can improve load efficiency for large XML documents by splitting the documents into parts and performing a separate load operation for each part. This is accomplished by specifying a value for the splitElement attribute for the source file.

When the element names of an XML document exactly match the column names in a target table and the document resides in a file, you can easily create a control file to extract and load data from the document into the target table.

The XML Document

The XML document in the following section (ORDERS) resides in a file named `<OWBhome>\owb\xmltoolkit`. This file name is relative to the Oracle database instance.

```
<ORDERS>
 <?xml version="1.0"?>
 <ROW>
     <ID>100</ID>
     <ORDER_DATE>2000.12.20</ORDER_DATE>
     <SHIPTO_NAME>Jeff Q. Vintner</SHIPTO_NAME>
     <SHIPTO_STREET>500 Marine World Parkway</SHIPTO_STREET>
     <SHIPTO_CITY>Redwood City</SHIPTO_CITY>
     <SHIPTO_STATE>CA</SHIPTO_STATE>
     <SHIPTO_ZIP>94065</SHIPTO_ZIP>
   </ROW>
</ORDERS>
```

The Target Table

The column names in the target table (PURCHASE_ORDERS) described by the DDL in the following section match the element names of the XML document described earlier.

```
create table Purchase_Orders (
id varchar2(10) not null,
order_date date not null,
shipto_name varchar2(60) not null,
shipto_street varchar2(80) not null,
shipto_city varchar2(30) not null,
shipto_state varchar2(2) not null,
shipto_zip varchar2(9))
```

The Control File

The control file in the following section directs the XML Toolkit to extract and load the data from the ORDERS document into the Purchase_Orders table.

```
'<OWBXMLRuntime>'||
  '<XMLSource>'||
    '<file>\ora817\examples\ex1.xml</file>'||
  '</XMLSource>'||
  '<targets>'||
    '<target dateFormat="yyyy.MM.dd">Purchase_Orders</target>'||
  '</targets>'||
'</OWBXMLRuntime>'
```

About the dateFormat Attribute

The dateFormat attribute defined for the target element is necessary so that the XML SQL Utility (XSU) can correctly store the text data in the table in the order_date column which has a data type of DATE. Refer to the Oracle Database XML Reference for additional information.

From the XSL processor, you can store the stored in various modes. Each example is a control file. When a transformation uses the wb_xml_load procedure of the wb_xml_load_f function to call the XML Toolkit, it must include a complete control file.

```
wb_xml_load (control_info VARCHAR2)
wb_xml_load_f (control_info VARCHAR2) RETURN NUMBER
```

The control file is an XML document that describes the sources, the targets, and any runtime controls. This section uses several examples that show you how to define control files for most situations.

A Warehouse Builder Transformation

If you create a transformation using this control file and call the transform from a pre-map process, the following actions occur:

1. The XML Toolkit reads the document ORDERS into a buffer and parses its data.
2. The Toolkit calls Oracle Database XML SQL utility (XSU) to load the parsed data into the Purchase_Orders table.
3. The Transformation then returns control to the mapping.

This is a simple case where the document and the table match up exactly. The next example shows you handle the more common case inexact matches.

Inexact Matches When the element names in the XML document fail to match the column names in the target column, you must include an XSL style sheet that reformats the data before it is loaded into the target table. This example shows you how to reformat the data by specifying a style sheet using the XSLFile attribute. The control file extracts and reformats the data before loading it into the PURCHASE_ORDERS table.

The XML Document The XML document in the following section (`purchaseORDER`) resides in the file `\ora817\examples\ex2.xml`. This file name is relative to the Oracle Database database instance.

```
<purchaseOrder>
<id>103123-4</id>
<orderDate>2000-10-20</orderDate>
<shipTo country="US">
     <name>Alice Smith</name>
     <street>123 Maple Street</street>
     <city>Mill Valley</city>
     <state>CA</state>
     <zip>90952</zip>
</shipTo>
<comment>Hurry, my lawn is going wild!</comment>
<items>
     <item>
          <partNum>872-AA</partNum>
          <productName>Lawnmower</productName>
          <quantity>1</quantity>
          <USPrice>148.95</USPrice>
          <comment>Confirm this is electric</comment>
     </item>

     <item>
          <partNum>845-ED</partNum>
          <productName>Baby Monitor</productName>
          <quantity>1</quantity>
          <USPrice>39.98</USPrice>
          <shipDate>1999-05-21</shipDate>
     </item>
</items>
</purchaseOrder>
```

The Target Table

The column names in the target table (PURCHASE_ORDERS) are described on page C-4.

The Control File

The control file in the following section directs the XML Toolkit to extract and reformat the data from the purchaseORDER document before loading it into the target table PURCHASE_ORDERS.

```
'<OWBXMLRuntime>'||
'     <XMLSource>'||
'            <file>\ora817\examples\ex2.xml</file>'||
'    </XMLSource>'||
'     <targets>'||
'            <target XSLFile="\ora817\examples\ex2_1.xsl" dateFormat="yyyy-MM-dd">'||
'                                    Purchase_Orders||'
'         </target>'||
'    </targets>'||
'</OWBXMLRuntime>'
```

The only addition to this control file is the XSLFile attribute for the element target.

The Style Sheet

The style sheet itself is a straightforward XML document that associates a parsed data item from the XML document with a column name in the target table. The following snippet from the style sheet shows how it associates a parsed data item with a column in the target:

```
<SHIPTO_NAME>
         <xsl:value-of select="shipTo/name"/>
</SHIPTO_NAME>
<SHIPTO_STREET>
       <xsl:value-of select="shipTo/street"/>
</SHIPTO_STREET>
<SHIPTO_CITY>
       <xsl:value-of select="shipTo/city"/>
</SHIPTO_CITY>
<SHIPTO_STATE>
       <xsl:value-of select="shipTo/state"/>
</SHIPTO_STATE>
```

This control statement could now be used in a transform to extract data from the XML document and load it into the target table.

A Warehouse Builder Transformation

If you create a transformation using this control file and call the transform from a pre-map process, the following actions occur:

1. The XML Toolkit reads the document ORDERS into a buffer and parses its data.
2. The Toolkit uses the style sheet to associate parsed data with table columns.
3. The Oracle Database XML SQL utility (XSU) loads the parsed data into PURCHASE_ORDERS.

The Transformation then returns control to the mapping.

Multiple Targets

This control statement extracts and reformats data from the purchaseORDER document and stores it into two target tables: PURCHASE_ORDERS and ITEMS. The tables have different column names, and so a style sheet must be specified for each target.

The XML Document The XML script for the previous example on page C-5 describes the source document (purchaseORDER).

The Target Tables The column names for Purchase_Orders are described in "The Target Table" on page C-4; the column names for items are not required to understand the control file script.

The Control File The control file in the following section directs the XML Toolkit to extract and reformat the data from the purchaseORDER document before loading it into two target tables.

```
'<OWBXMLRuntime>'||
'     <XMLSource>'||
'            <file>\ora817\examples\ex2.xml</file>'||
'    </XMLSource>'||
'     <targets>'||
'            <target XSLFile="\ora817\examples\ex2_1.xsl" dateFormat="yyyy-MM-dd">'||
'                                    Purchase_Orders||'
'           </target>'||
'            <target XSLFile="\ora817\examples\ex2_2.xsl" dateFormat="yyyy-MM-dd">'||
'                                    Items||'
'          </target>'||
'    </targets>'||
'</OWBXMLRuntime>'
```

This control file is like the previous example except that it specifies an additional target (ITEMS), and a style sheet (ex2_2.xsl) which associates parsed data items from the document with columns in the Items table.

Very Large XML Documents

When you extract data from a very large XML document, the memory requirements for the processing can impact load performance. In this case, you can often reduce the memory requirements and improve the efficiency of the operation by dividing the XML documents into multiple parts.

To divide an XML document into parts, specify a value for the splitElement attribute of the XMLSource element. The value is the name of an element within the source XML document, preferably one that delineates blocks of text.

A good example is an XML document that organizes books into categories using the element name Category. There are many categories, and each category defines numerous books. Thus, if you specify the splitElement attribute as in the example in the following section, then the XML Toolkit divides the source document into as many parts as there are categories.

The Control File The control file in the following section directs the XML Toolkit to extract the data from the BookAreUs document by dividing the document according to its catalogs, and then load each part into the target table books.

```
'<OWBXMLRuntime>'||
'     <XMLSource splitElement="Category">'||
'              <file>\ora817\examples\ex4.xml</file>'||
'    </XMLSource>'||
'     <targets>'||
'             <target XSLFile="ora817\examples\ex4.xsl">books</target>'||
'    </targets>'||
'</OWBXMLRuntime>'
```

How well this operation improves efficiency depends on the selection of the split element. If the split element defines only a few parts, then memory resources may not be reduced enough to improve performance.

XML Documents Stored as Other Objects

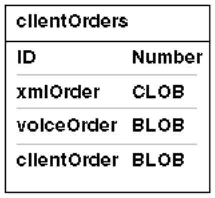
The XML Toolkit can extract data from documents stored as a database object or at a location defined by a URL. The examples in this section illustrate control files for these cases.

Document Stored as a CLOB

This example control file loads the PURCHASE_ORDERS table with data extracted from an XML document that is stored in a table (clientOrders) as a CLOB column.

The clientOrders table contains detailed information about each order, including multimedia information. The order itself is stored in the xmlOrder column, a voice recording of the order is stored in the voiceOrder column, and a picture of the client who made the order is stored in the clientOrder column.

Figure C–1 Sample Document Stored as CLOB



clientOrders	
ID	Number
xmlOrder	CLOB
voiceOrder	BLOB
clientOrder	BLOB

The XML Document The XML document which is stored in the first row of the clientOrders table is exactly the same as the order described in the first example. The complete XML document that defines the order is described on page C-4.

The Control File The control file in the following section directs the XML Toolkit to extract data from the document that resides in the first row of the clientOrders table and load it into the Purchase_Orders table.

```
'<OWBXMLRuntime>'||
  '<XMLSource>'||
'<CLOB whereClause="where id=''1''">'||
     '<table>clientOrders</table>'||
     '<CLOBColumn>xmlOrder</CLOBColumn>'||
'</CLOB>'||
'</XMLSource>'||
'<targets>'||
'<target dateFormat="yyyy.MM.dd">Purchase_Orders</target>'||
'</targets>'||
'</OWBXMLRuntime>'
```

Comments on the Control File A few comments are in order regarding this control file, especially regarding the values assigned to attributes.

1. The first row of the table is specified as an attribute of the CLOB element:

   ```
   CLOB whereClause="where id=''1''"
   ```

   The ID column of the clientOrders table identifies each row.
2. The dateFormat attribute describes the format of the character value parsed from the ORDER_DATE element. This information is necessary to load the parsed data into the DATE datatype column of Purchase_Orders.
3. A style sheet is not required because the element names in the XML document exactly match the column names of the Purchase_Orders table. If they did not, then a style sheet would be required.
4. You could store the data parsed from the XML document in multiple tables by including additional target elements. Of course, a style sheet would be required whenever the column names of a target table fail to match the XML elements.

Document Stored as an Object-Based Advanced Queue

This example control file loads the PURCHASE_ORDERS table with data extracted from XML documents that are placed on an Oracle Advanced Queue (AQ).

An e-commerce application can use an Advanced Queue to automate the acceptance, processing, routing, and completion of orders. You can configure an AQ to remain active and wait for entries to be placed on the queue and also specify a maximum wait time before the queue deactivates itself.

The following control file loads the PURCHASE_ORDERS table with data extracted XML documents that are periodically placed on newOrders, an Object-Based AQ. The control file specifies that the queue wait for new entries and specifies a time out value in seconds or forever. The CLOBColumn attribute indicated the column in the object type containing the XML to be loaded.

For additional information on Advanced Queues, refer to Oracle Database *Application Developer's Guide - Advanced Queuing*.

The XML Document The XML documents which are periodically placed on the newOrders queue is exactly the same as the order described in the first example. The complete XML document that defines the order is described on page C-4.

The Control File The control file in the following section directs the XML Toolkit to extract data from documents that are periodically place on the newOrdersAQ and store them into the PURCHASE_ORDERS table.

```
'<OWBXMLRuntime>'||
'<XMLSource>'||
```

```
'<AQ wait="WAIT" waitTime="WAIT_FOREVER">'||
'<AQName>newOrders</AQName>'||
'<ObjectType>xmlMessages</ObjectType>'||
'<CLOBColumn>xmlEntry</CLOBColumn>'||
'</AQ>'||
'</XMLSource>'||
'<targets>'||
'<target dateFormat="yyyy.MM.dd">Purchase_Orders</target>'||
'</targets>'||
'</OWBXMLRuntime>');
```

The waitTime attribute is configured so that a dequeue call wait until an entry is available on the queue before it is released.

Document Stored as a Raw Advanced Queue

In this example, a control file loads the PURCHASE_ORDERS table with data extracted from XML documents that are placed on an Oracle Database Advanced Queue (AQ).

An e-commerce application can use an Advanced Queue to automate the acceptance, processing, routing, and completion of orders. You can configure an AQ to remain active and wait for entries to be placed on the queue and also specify a maximum wait time before the queue deactivates itself.

The following control file loads the PURCHASE_ORDERS table with data extracted XML documents that are periodically placed on newOrders, a RAW AQ. The control file specifies that the queue wait for new entries and specifies a time out value in seconds or forever. The next example shows how to rewrite the control file for a queue that resides in an object based Advanced Queue.

For additional information on Advanced Queues, refer to *Oracle Database Application Developer's Guide - Advanced Queuing*.

The XML Document The XML documents which are periodically placed on the newOrdersAQ queue is exactly the same as the order described in the first example. The complete XML document that defines the order is described on page C-4.

The Control File The control file in the following section directs the XML Toolkit to extract data from documents that are periodically place on the newOrdersAQ Advanced Queue and store them into the Purchase_Orders table.

```
'<OWBXMLRuntime>'||
'<XMLSource>'||
'<AQ wait="WAIT" waitTime="20">'||
'<AQName>raw_queue</AQName>'||
'</AQ>'||
'</XMLSource>'||
'<targets>'||
          '<target dateFormat="yyyy.MM.dd">Purchase_Orders</target>'||
'</targets>'||
'</OWBXMLRuntime>');
```

The waitTime attribute specifies that a dequeue call made on an empty queue will wait a maximum of twenty seconds for a document to be placed on the queue.

Document Stored at a URL

This example control file loads the Books table with data extracted from a document produced by a URL.

```
'<OWBXMLRuntime>'||
'<XMLSource>'||
      '<URL parameterTable="ptable">'||
             ' <![CDATA[http://oracle.com//xmlserv/library]]>'||
      '</URL>'||
'</XMLSource>'||
'<targets>'||
     '<target XSLFile="\ora817\examples\lib.xsl"  ignoreCase="TRUE">Books'||
     '</target>'||
'</targets>'||
'</OWBXMLRuntime>'
```

The targets element includes two attributes: one for a style sheet to match the source element names with the target column names, and another to ignore case. The parameter table is used when dynamically generating XML, the column names of the parameter table are used as parameter names, and the data as parameter values. Only VARCHAR2, VARCHAR, CHAR, and NUMBER columns are used to form dynamic parameters.

Control File Element Names and Attributes

The control file for the XML Toolkit is an XML document, and the next two sections describe all of its elements. The first section informally describes each element that can be used to create a control file; the second section is the Document Type Definition (DTD) of the control file, which formally defines the control file structure.

The following description of the control file elements is divided into three parts: elements that describe the source for XML documents, elements that describe the target of the load operation, and elements that determine the runtime environment for the XML Toolkit.

Source Elements

Table C–1 describes all the elements used to define sources of XML documents and their respective attributes.

Table C–1 Source Elements

Name	Description
OWBXMLRuntime	Root element for the control file document. Attributes: None.
XMLSource	Defines all the sources using additional elements. Attributes: splitElement Divides the source XML documents into multiple documents. Used to reduce the memory requirements when loading large XML documents. If used, all information in the DOMParserConfig is ignored.
file	Specifies a single XML document that resides in a single file as the source. Attributes: None.

Table C–1 (Cont.) Source Elements

Name	Description
directory	Specifies a single directory as the source of XML documents. The Toolkit extracts all the XML documents that match a mask which is specified in the mask attribute. Attributes: mask Filter for files that reside in a directory. The Toolkit supports only Windows * masks. For example, "*.xml".
AQ	Specifies that the source of XML document resides in an Advanced Queue. Attributes: consumerName dequeueMode navigation visibility wait waitTime messageID correlation These attributes represent the various dequeue options supported by Advanced Queues. For complete information, refer to *Oracle Database Application Developer's Guide - Advanced Queuing*.
AQFullName	Specifies the complete name of an Advanced Queue. You can specify as SCHEMA.AQ_NAME or AQ_NAME. Attributes: None.
CLOBColumn	Specifies that an object based on an Advanced Queue contains a CLOB column (object/CLOB based AQ). If not specified, indicates that the AQ is a RAW based AQ. Attributes: None.
CLOB	Specifies the source of XML document resides in a CLOB column of a database table. Attributes: whereClause Specifies where clause for the query that retrieves the XML document from the CLOB column.
CLOB.table	Specifies the table or view that contains a CLOB column. Attributes: None.
CLOBColumn	Specifies the name of a CLOB column. Attributes: None.

Table C–1 (Cont.) Source Elements

Name	Description
URL	Specifies that the XML document resides at an address given as a URL. Attributes: proxy Port number of the proxy server. parameterTable Directs the Toolkit to retrieve documents from the URL multiple times according to the parameters. The parameter names are the parameter table's column names. The Toolkit reads only columns that have VARCHAR, VARCHAR2, CHAR and NUMBER data types. lowerParameterNames If TRUE, parameter names are in lowercase; if FALSE, parameter names are in uppercase.

Target Elements

Table C–2 describes all the elements used to define the load targets and their respective attributes. You can define multiple targets for the data parsed from a set of XML documents.

Table C–2 Target Elements

Name	Description
targets	Specifies all the database targets. You specify individual targets using target elements. Attributes: None.

Table C–2 (Cont.) Target Elements

Name	Description
target	Specifies the target of a database load. Attributes: truncateFirst If TRUE, then truncate the target object before the load. XSLfile Address of the XSL style sheet which the Toolkit uses to associate parsed data with individual target columns. XSLRefURL URL the Toolkit uses to resolve external references made by the XSL style sheet. LoadType Specifies the load operation as INSERT, UPDATE, or DELETE. When UPDATE or DELETE is specified, truncateFirst and truncateBeforeEachLoad are set to FALSE. The following attributes are all XU attributes which are defined in the Oracle Database XML Reference. ignoreCase commitBatch rowTag (to set, use VARCHAR 'NULL') dateFormat batchSize keyColumnList updateColumnList

Runtime Control

Table C–3 describes elements that determine the runtime environment.

Table C–3 Runtime Control

Name	Description
runtimeConfig	Specifies the runtime configuration for the XML Toolkit. **Attributes:** commitAfterLoad If TRUE, then issue commit at the end of the load.

Document Type Definition for the Control File

The Document Type Definition in the following section describes all the possible elements which may occur in a control file for the XML Toolkit, their respective attributes, their ordering, and the conditions of their use.

```
<!ELEMENT OWBXMLRuntime (XMLSource, targets, runtimeConfig?)>
<!ELEMENT XMLSource ((file | directory | URL | CLOB | AQ ),DOMParserConfig?)>
<!ATTLIST XMLSource splitElement CDATA #IMPLIED>
<!ELEMENT file (#PCDATA)>
<!ELEMENT directory (#PCDATA)>
<!ATTLIST directory mask CDATA "*.xml">
<!ELEMENT CLOB (table, CLOBColumn)>
<!ELEMENT table (#PCDATA)>
```

```
<!ATTLIST CLOB whereClause CDATA #IMPLIED>
<!ELEMENT URL (#PCDATA)>
<!ATTLIST URL proxy CDATA #IMPLIED
     proxyPort CDATA "80"
     parameterTable CDATA #IMPLIED
     lowerParameterNames (TRUE | FALSE) "TRUE">
<!ELEMENT AQ (AQName, (ObjectType, CLOBColumn)?)>
<!ELEMENT AQName (#PCDATA)>
<!ELEMENT ObjectType (#PCDATA)>
<!ELEMENT CLOBColumn (#PCDATA)>
<!ATTLIST AQ
     consumerName CDATA #IMPLIED
     dequeueMode (REMOVE | BROWSE | LOCKED) "REMOVE"
     navigation (NEXT_MESSAGE | NEXT_TRANSACTION | FIST_MESSAGE
          "NEXT_MESSAGE"
     visibility (ON_COMMIT | IMMEDIATE) "IMMEDIATE"
     wait (WAIT_FOREVER | WAIT_NONE | WAIT) "WAIT_FOREVER"
     waitTime CDATA #IMPLIED
     messageID CDATA #IMPLIED
     correlation CDATA #IMPLIED>
<!ELEMENT targets (target+)>
<!ELEMENT target (#PCDATA)>
<!ATTLIST target
      truncateFirst (TRUE | FALSE) "TRUE"
      truncateBeforeEachLoad (TRUE | FALSE) "FALSE"
     XSLFile CDATA #IMPLIED
     XSLRefURL CDATA #IMPLIED
     loadType (INSERT| UPDATE | DELETE) "INSERT"
     ignoreCase (TRUE | FALSE) "TRUE"
     commitBatch CDATA "0"
     rowTag CDATA "ROW"
     dateFormat CDATA "MM/dd/yyyy HH:mm:ss"
     batchSize CDATA "17"
     keyColumnList NMTOKENS #IMPLIED
     updateColumnList NMTOKENS #IMPLIED>
<!ELEMENT runtimeConfig EMPTY>
<!ATTLIST
     runtimeConfig
     commitAfterLoad (TRUE | FALSE) "TRUE"
     batchSize CDATA "10">
<!ELEMENT DOMParserConfig EMPTY>
<!ATTLIST DOMParserConfig
     showWarnings (TRUE | FALSE) "FALSE"
     retainCDATASection (TRUE | FALSE) "FALSE"
     debugMode (TRUE | FALSE) "FALSE"
     preserveWhitespace (TRUE | FALSE) "FALSE"
     validationMode (TRUE | FALSE) "FALSE"
     baseURL CDATA #IMPLIED>
```

Reference Materials

The XML Toolkit employs several Oracle Database XML SDK software components. For more information, refer to the following:

- *Application Developer's Guide - Advanced Queuing*
- *Oracle Database Application Developer's Guide* - XML
- *Oracle Database* XML R*eference*

Web sites that contain information on the XML specification are:

- http://www.w3.org/XML/
- http://www.XML.com

D

Warehouse Builder Public Views

The Warehouse Builder provides a set of pre-built views for both the Design and Runtime Repositories. These views are called the Warehouse Builder Public Views. Use these views to access to data stored in these repositories. This appendix contains a catalog of the Public Views and their descriptions.

Warehouse Builder Design Repository Public Views

The design repository contains all of the design metadata. Use these Public Views to access data about the design of your system. These views are used by Warehouse Builder Browser to provide metadata reporting.

General Model Views

General Model Views

Table D–1 ALL_IV_ALL_OBJECTS

Column Name	Data Type	Description
OBJECT_ID	NUMBER(9)	ID of the object
OBJECT_UOID	VARCHAR2(255)	UOID of the object
OBJECT_TYPE	VARCHAR2(4000)	Type of the object
OBJECT_NAME	VARCHAR2(4000)	Physical name of the object
BUSINESS_NAME	VARCHAR2(4000)	Business name of the object
CONTEXT_NAME	VARCHAR2(4000)	Name of the object, prefixed with its module's name, and project's name if existed.
DESCRIPTION	VARCHAR2(4000)	Description of the object
PARENT_OBJECT_ID	NUMBER(9)	Container object ID for the object. Container object could be a module, for example for a dimension, or a table for a column.
PARENT_OBJECT_TYPE	VARCHAR2(4000)	Type of the parent object
PARENT_OBJECT_NAME	VARCHAR2(4000)	Name of the parent object
IS_VALID	VARCHAR2(13)	Is the object valid? It only makes sense for the objects that can be validated.
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–2 ALL_IV_OBJECTS

Column Name	Data Type	Description
OBJECT_ID	NUMBER(9)	ID of the object (the difference between this view and 2.1 view is that 2.1 view includes all objects in this view, PLUS all archived snapshot objects (for MCM service))
OBJECT_TYPE	VARCHAR2(4000)	Type of the object
OBJECT_NAME	VARCHAR2(4000)	Physical name of the object
BUSINESS_NAME	VARCHAR2(4000)	Business name of the object
DESCRIPTION	VARCHAR2(4000)	Description of the object
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–3 ALL_IV_OBJECT_PROPERTIES

Column Name	Data Type	Description
OBJECT_ID	NUMBER(9)	ID of the object
OBJECT_TYPE	VARCHAR2(4000)	Type of the object
OBJECT_NAME	VARCHAR2(255)	Physical name of the object
PROPERTY_ID	NUMBER(9)	ID of the object's property
PROPERTY_NAME	VARCHAR2(255)	ID of the property name
PROPERTY_VALUE	VARCHAR2(4000)	Value of the property

Table D–4 ALL_IV_MLS_OBJECTS

Column Name	Data Type	Description
OBJECT_ID	NUMBER(9)	ID of the object (covers the same set as 2.2 view)
LANGUAGE_ID	VARCHAR2(255)	ID of the language (predefined internally by OWB). To get language name, please join with 2.5 view.
BUSINESS_NAME	VARCHAR2(4000)	Business name of the object
DESCRIPTION	VARCHAR2(4000)	Description of the object

Table D–5 ALL_IV_SUPPORTED_LANGUAGES

Column Name	Data Type	Description
LANGUAGE_ID	VARCHAR2(255)	ID of the language
LANGUAGE_NAME	VARCHAR2(64)	Name of the language
ISBASELANGUAGE	VARCHAR2(1)	Is it a base language (for example EN or FR)?

Table D–6 ALL_IV_MODULES

Column Name	Data Type	Description
PROJECT_ID	NUMBER(9)	ID of the project that this module belongs to
PROJECT_NAME	VARCHAR2(255)	Physical name of the project
INFORMATION_SYSTEM_ID	NUMBER(9)	ID of the module
INFORMATION_SYSTEM_NAME	VARCHAR2(255)	Physical name of the module
SCHEMA_ID	NUMBER(9)	ID of the module (repeated column, just to keep backward compatibility)
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
BUSINESS_NAME	VARCHAR2(4000)	Business name of the module
DESCRIPTION	VARCHAR2(4000)	Description of the module
STATUS	VARCHAR2(40)	Module status (dev, QA, prod)
IS_VALID	VARCHAR2(13)	Is this module valid?
LOCATION_ID	NUMBER(9)	ID of the associated location for this module
LOCATION_NAME	VARCHAR2(255)	Name of associated location for this module
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–7 ALL_IV_PROJECTS

Column Name	Data Type	Description
PROJECT_ID	NUMBER(9)	ID of the project
PROJECT_NAME	VARCHAR2(255)	Physical name of the project
BUSINESS_NAME	VARCHAR2(4000)	Business name of the project
DESCRIPTION	VARCHAR2(4000)	Description of the project
VERSION_LABEL	VARCHAR2(255)	Version of the project
IS_VALID	VARCHAR2(13)	Is this project valid?
UPDATED_ON	DATE	Update timestamp

Table D–7 (Cont.) ALL_IV_PROJECTS

Column Name	Data Type	Description
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–8 ALL_IV_INFORMATION_SYSTEMS

Column Name	Data Type	Description
PROJECT_ID	NUMBER(9)	ID of the project that this module belongs to
PROJECT_NAME	VARCHAR2(255)	Physical name of the project
INFORMATION_SYSTEM_ID	NUMBER(9)	ID of the module
INFORMATION_SYSTEM_NAME	VARCHAR2(255)	Physical name of the module
INFORMATION_SYSTEM_TYPE	VARCHAR2(4000)	Type of the module
BUSINESS_NAME	VARCHAR2(4000)	Business name of the module
DESCRIPTION	VARCHAR2(4000)	Description of the module
PRODUCT_TYPE	VARCHAR2(255)	Application type of the module (for example Oracle apps or File based apps)
SYSTEM_TYPE	VARCHAR2(255)	Type of system that holds this application (shown by PRODUCT_TYPE)
VERSION_LABEL	NUMBER(9)	Version of the module
VENDOR	VARCHAR2(40)	Vendor name
DATABASE_LINK	VARCHAR2(40)	Name of the database link that physical points to data storage of this module. It is meaningful only for database applications.
INTEGRATOR_NAME	VARCHAR2(255)	The name of OWB integrator component that is used to access external data source for the module.
IS_VALID	VARCHAR2(13)	Is this module valid?
LOCATION_ID	NUMBER(9)	ID of the associated location for this module
LOCATION_NAME	VARCHAR2(255)	Physical name of the associated location
STATUS	VARCHAR2(17)	Status (dev, QA, or prod)
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–9 ALL_IV_INSTALLATIONS

Column Name	Data Type	Description
INSTALLATION_ID	NUMBER(9)	ID of the OWB repository
INSTALLATION_NAME	VARCHAR2(255)	Physical name of the OWB repository
BUSINESS_NAME	VARCHAR2(4000)	Business name of the OWB repository
DESCRIPTION	VARCHAR2(4000)	Description of the OWB repository
INSTALLED_VERSION	VARCHAR2(40)	Version of the OWB repository
RELEASE	VARCHAR2(40)	Version of the OWB Client

Table D–9 (Cont.) ALL_IV_INSTALLATIONS

Column Name	Data Type	Description
REPOSITORY_MODEL_VERSION	NUMBER(9)	Version of the OWB model
PUBLIC_VIEW_VERSION	CHAR(5)	Version of the OWB Public Views
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–10 ALL_IV_FILE_MODULES

Column Name	Data Type	Description
PROJECT_ID	NUMBER(9)	ID of the project that this module belongs to
PROJECT_NAME	VARCHAR2(255)	Physical name of the project
INFORMATION_SYSTEM_ID	NUMBER(9)	ID of this file module
INFORMATION_SYSTEM_NAME	VARCHAR2(255)	Physical name of this file module
BUSINESS_NAME	VARCHAR2(4000)	Business name of this file module
DESCRIPTION	VARCHAR2(4000)	Description of this file module
PRODUCT_TYPE	VARCHAR2(255)	Application type of the module (for example Oracle apps or File based apps)
SYSTEM_TYPE	VARCHAR2(255)	Type of system that holds this application (represented by PRODUCT_TYPE)
VERSION_LABEL	NUMBER(9)	Version of the module
VENDOR	VARCHAR2(40)	Vendor name
DATABASE_LINK	VARCHAR2(40)	Name of the database link that physical points to data storage of this module. It is meaningful only for database applications.
DIRECTORY	VARCHAR2(4000)	Name of the directory this file module connects to
INTEGRATOR_NAME	VARCHAR2(255)	The name of OWB integrator component that is used to access external file system for the module.
IS_VALID	VARCHAR2(13)	Is this module valid?
LOCATION_ID	NUMBER(9)	ID of the associated location for this module
LOCATION_NAME	VARCHAR2(255)	Physical name of the associated location
STATUS	VARCHAR2(17)	Status (dev, QA, or prod)
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–11 ALL_IV_GATEWAY_MODULES

Column Name	Data Type	Description
PROJECT_ID	NUMBER(9)	ID of the project that this module belongs to
PROJECT_NAME	VARCHAR2(255)	Physical name of the project
INFORMATION_SYSTEM_ID	NUMBER(9)	ID of this module

Table D–11 (Cont.) ALL_IV_GATEWAY_MODULES

Column Name	Data Type	Description
INFORMATION_SYSTEM_ NAME	VARCHAR2(255)	Physical name of this module
BUSINESS_NAME	VARCHAR2(4000)	Business name of this module
DESCRIPTION	VARCHAR2(4000)	Description of this module
PRODUCT_TYPE	VARCHAR2(255)	Application type of the module (for example Oracle apps or File based apps)
SYSTEM_TYPE	VARCHAR2(255)	Type of system that holds this application (represented by PRODUCT_TYPE)
VERSION_LABEL	NUMBER(9)	Version of the module
VENDOR	VARCHAR2(40)	Vendor name
DATABASE_LINK	VARCHAR2(40)	Name of the database link that physical points to data storage of this module. It is meaningful only for database applications.
INTEGRATOR_NAME	VARCHAR2(255)	The name of OWB integrator component that is used to access external data system for the module.
STRONG_TYPE_NAME	VARCHAR2(255)	Used to differentiate which gateway component being employed, for example, Informix or Sybase.
IS_VALID	VARCHAR2(13)	Is this module valid?
LOCATION_ID	NUMBER(9)	ID of the associated location for this module
LOCATION_NAME	VARCHAR2(255)	Physical name of the associated location
STATUS	VARCHAR2(17)	Status (dev, QA, or prod)
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–12 ALL_IV_PACKAGED_APPS_MODULES

Column Name	Data Type	Description
PROJECT_ID	NUMBER(9)	ID of the project that this module belongs to
PROJECT_NAME	VARCHAR2(255)	Physical name of the project
INFORMATION_SYSTEM_ID	NUMBER(9)	ID of this module (basically, the views wraps Oracle Applications, SAP, etc)
INFORMATION_SYSTEM_ NAME	VARCHAR2(255)	Physical name of this module
BUSINESS_NAME	VARCHAR2(4000)	Business name of this module
DESCRIPTION	VARCHAR2(4000)	Description of this module
PRODUCT_TYPE	VARCHAR2(255)	Application type of the module (for example Oracle apps or File based apps)
SYSTEM_TYPE	VARCHAR2(255)	Type of system that holds this application (represented by PRODUCT_TYPE)
VERSION_LABEL	NUMBER(9)	Version of the module
VENDOR	VARCHAR2(40)	Vendor name
DATABASE_LINK	VARCHAR2(40)	Name of the database link that physical points to data storage of this module. It is meaningful only for database applications.

Table D–12 (Cont.) ALL_IV_PACKAGED_APPS_MODULES

Column Name	Data Type	Description
INTEGRATOR_NAME	VARCHAR2(255)	The name of OWB integrator component that is used to access external data system for the module.
IS_VALID	VARCHAR2(13)	Is this module valid?
LOCATION_ID	NUMBER(9)	ID of the associated location for this module
LOCATION_NAME	VARCHAR2(255)	Physical name of the associated location
STATUS	VARCHAR2(17)	Status (dev, QA, or prod)
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–13 ALL_IV_PREDEFINED_MODULES

Column Name	Data Type	Description
PROJECT_ID	NUMBER(9)	ID of the project that this module belongs to
PROJECT_NAME	VARCHAR2(255)	Physical name of the project
INFORMATION_SYSTEM_ID	NUMBER(9)	ID of this module (basically, the views wraps Oracle Pre-defined Transformations and Public Transformations)
INFORMATION_SYSTEM_ NAME	VARCHAR2(255)	Physical name of this module
BUSINESS_NAME	VARCHAR2(4000)	Business name of this module
DESCRIPTION	VARCHAR2(4000)	Description of this module
PRODUCT_TYPE	VARCHAR2(255)	Application type of the module (for example Oracle apps or File based apps)
SYSTEM_TYPE	VARCHAR2(255)	Type of system that holds this application (represented by PRODUCT_TYPE)
VERSION_LABEL	NUMBER(9)	Version of the module
VENDOR	VARCHAR2(40)	Vendor name
DATABASE_LINK	VARCHAR2(40)	Name of the database link that physical points to data storage of this module. It is meaningful only for database applications.
INTEGRATOR_NAME	VARCHAR2(255)	The name of OWB integrator component that is used to access external data system for the module.
IS_VALID	VARCHAR2(13)	Is this module valid?
STATUS	VARCHAR2(17)	Status (dev, QA, or prod)
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–14 ALL_IV_PROCESS_MODULES

Column Name	Data Type	Description
PROJECT_ID	NUMBER(9)	ID of the project that this module belongs to
PROJECT_NAME	VARCHAR2(255)	Physical name of the project
INFORMATION_SYSTEM_ID	NUMBER(9)	ID of this module (basically, the views wraps Oracle Process Flow Module)
INFORMATION_SYSTEM_ NAME	VARCHAR2(255)	Physical name of this module
BUSINESS_NAME	VARCHAR2(4000)	Business name of this module
DESCRIPTION	VARCHAR2(4000)	Description of this module
PRODUCT_TYPE	VARCHAR2(255)	Application type of the module (for example Oracle apps or File based apps)
SYSTEM_TYPE	VARCHAR2(255)	Type of system that holds this application (represented by PRODUCT_TYPE)
VERSION_LABEL	NUMBER(9)	Version of the module
VENDOR	VARCHAR2(40)	Vendor name
DATABASE_LINK	VARCHAR2(40)	Name of the database link that physical points to data storage of this module. It is meaningful only for database applications.
INTEGRATOR_NAME	VARCHAR2(255)	The name of OWB integrator component that is used to access external data system for the module.
IS_VALID	VARCHAR2(13)	Is this module valid?
LOCATION_ID	NUMBER(9)	ID of the associated location for this module
LOCATION_NAME	VARCHAR2(255)	Physical name of the associated location
STATUS	VARCHAR2(17)	Status (dev, QA, or prod)
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–15 ALL_IV_WAREHOUSE_MODULES

Column Name	Data Type	Description
PROJECT_ID	NUMBER(9)	ID of the project that this module belongs to
PROJECT_NAME	VARCHAR2(255)	Physical name of the project
INFORMATION_SYSTEM_ID	NUMBER(9)	ID of this module (basically, the views wraps Oracle Warehouse Module)
INFORMATION_SYSTEM_NAME	VARCHAR2(255)	Physical name of this module
BUSINESS_NAME	VARCHAR2(4000)	Business name of this module
DESCRIPTION	VARCHAR2(4000)	Description of this module
PRODUCT_TYPE	VARCHAR2(255)	Application type of the module (for example Oracle apps or File based apps)
SYSTEM_TYPE	VARCHAR2(255)	Type of system that holds this application (represented by PRODUCT_TYPE)
VERSION_LABEL	NUMBER(9)	Version of the module
VENDOR	VARCHAR2(40)	Vendor name
DATABASE_LINK	VARCHAR2(40)	Name of the database link that physical points to data storage of this module.

Table D–15 (Cont.) ALL_IV_WAREHOUSE_MODULES

Column Name	Data Type	Description
INTEGRATOR_NAME	VARCHAR2(255)	The name of OWB integrator component that is used to access external data system for the module.
IS_VALID	VARCHAR2(13)	Is this module valid?
LOCATION_ID	NUMBER(9)	ID of the associated location for this module
LOCATION_NAME	VARCHAR2(255)	Physical name of the associated location
STATUS	VARCHAR2(17)	Status (dev, QA, or prod)
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Data Model Views

Table D–16 ALL_IV_ADVANCED_QUEUES

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this queue belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
QUEUE_ID	NUMBER(9)	ID of this queue
QUEUE_NAME	VARCHAR2(255)	Physical name of this queue
BUSINESS_NAME	VARCHAR2(4000)	Business name of this queue
DESCRIPTION	VARCHAR2(4000)	Description of this queue
LOAD_TYPE_ID	NUMBER(9)	ID of the load type
LOAD_TYPE_NAME	VARCHAR2(255)	Name of the load type
QUEUE_TABLE_NAME	VARCHAR2(40)	Name of the queue table
IS_VALID	VARCHAR2(13)	Is this queue valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–17 ALL_IV_ATTR_GROUPS

Column Name	Data Type	Description
DATA_ENTITY_ID	NUMBER(9)	ID of the data entity this attribute group belongs to
DATA_ENTITY_TYPE	VARCHAR2(4000)	Type of the data entity
DATA_ENTITY_NAME	VARCHAR2(255)	Physical name of the data entity
ATTRIBUTE_GROUP_NAME	VARCHAR2(255)	Physical name of this attribute group
ATTRIBUTE_GROUP_ID	NUMBER(9)	ID of this attribute group
BUSINESS_NAME	VARCHAR2(4000)	Business name of this attribute group
DESCRIPTION	VARCHAR2(4000)	Description of this attribute group
ATTRIBUTE_GROUP_TYPE	VARCHAR2(40)	Type of attribute group
UPDATED_ON	DATE	Update timestamp

Table D–17 (Cont.) ALL_IV_ATTR_GROUPS

Column Name	Data Type	Description
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–18 ALL_IV_ATTR_GROUP_ITEM_USES

Column Name	Data Type	Description
ATTRIBUTE_GROUP_ID	NUMBER(9)	ID of the attribute group that this data item belongs to
ATTRIBUTE_GROUP_NAME	VARCHAR2(255)	Name of the attribute group
DATA_ITEM_ID	NUMBER(9)	ID of this data item
DATA_ITEM_TYPE	VARCHAR2(4000)	Type of this data item
DATA_ITEM_NAME	VARCHAR2(255)	Physical name of this data item
POSITION	NUMBER(9)	Position of this data item in the attribute group.

Table D–19 ALL_IV_CHECK_CONSTRAINTS

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this check constraint belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
RELATION_ID	NUMBER(9)	ID of the relation entity this check constraint belongs to
RELATION_NAME	VARCHAR2(255)	Physical name of the relation entity
CONSTRAINT_ID	NUMBER(9)	ID of this check constraint
CONSTRAINT_NAME	VARCHAR2(255)	Physical name of this check constraint
BUSINESS_NAME	VARCHAR2(4000)	Business name of this check constraint
DESCRIPTION	VARCHAR2(4000)	Description of this check constraint
CONSTRAINT_TEXT	VARCHAR2(255)	Textual expression of this check constraint
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–20 ALL_IV_COLUMNS

Column Name	Data Type	Description
ENTITY_ID	NUMBER(9)	ID of the data entity this column belongs to
ENTITY_TYPE	VARCHAR2(4000)	Type of the data entity
ENTITY_NAME	VARCHAR2(255)	Physical name of the data entity
COLUMN_ID	NUMBER(9)	ID of this column
COLUMN_NAME	VARCHAR2(255)	Physical name of this column
BUSINESS_NAME	VARCHAR2(4000)	Business name of this column
DESCRIPTION	VARCHAR2(4000)	Description of this column
POSITION	NUMBER(9)	Position of this column in the data entity
DATA_TYPE	VARCHAR2(255)	Data type of this column

Table D–20 (Cont.) ALL_IV_COLUMNS

Column Name	Data Type	Description
LENGTH	NUMBER(9)	Data length of this column
PRECISION	NUMBER(9)	Data precision of this column
SCALE	NUMBER(9)	Data scale of this column
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–21 ALL_IV_CONSTRAINTS

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this constraint belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
RELATION_ID	NUMBER(9)	ID of the relational entity this constraint belongs to
RELATION_NAME	VARCHAR2(255)	Physical name of the relational entity
CONSTRAINT_ID	NUMBER(9)	ID of this constraint
CONSTRAINT_NAME	VARCHAR2(255)	Physical name of this constraint
CONSTRAINT_TYPE	VARCHAR2(21)	Type of this constraint (check, primary, foreign key)
BUSINESS_NAME	VARCHAR2(4000)	Business name of this constraint
DESCRIPTION	VARCHAR2(4000)	Description of this constraint
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–22 ALL_IV_CUBES

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this cube belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
CUBE_ID	NUMBER(9)	ID of this cube
CUBE_NAME	VARCHAR2(255)	Physical name of this cube
BUSINESS_NAME	VARCHAR2(4000)	Business name of this cube
DESCRIPTION	VARCHAR2(4000)	Description of this cube
IS_VALID	VARCHAR2(13)	Is this cube valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–23 ALL_IV_CUBE_DIMENSIONS

Column Name	Data Type	Description
CUBE_ID	NUMBER(9)	ID of the cube this dimension has associated with
CUBE_NAME	VARCHAR2(255)	Physical name of the cube
DIMENSION_ID	NUMBER(9)	ID of this dimension
DIMENSION_NAME	VARCHAR2(255)	Physical name of this dimension

Table D–24 ALL_IV_CUBE_MEASURES

Column Name	Data Type	Description
CUBE_ID	NUMBER(9)	ID of the cube this measure belongs to
CUBE_NAME	VARCHAR2(255)	Physical name of the cube
MEASURE_ID	NUMBER(9)	ID of this measure
MEASURE_NAME	VARCHAR2(255)	Physical name of this measure
BUSINESS_NAME	VARCHAR2(4000)	Business name of this measure
DESCRIPTION	VARCHAR2(4000)	Description of this measure
POSITION		Position of this measure within the cube
DATA_TYPE	VARCHAR2(255)	Data type of this measure
LENGTH	NUMBER(9)	Data length of this measure
PRECISION	NUMBER(9)	Data precision of this measure
SCALE	NUMBER(9)	Data scale of this measure
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–25 ALL_IV_CUBE_MEASURE_DIM_USES

Column Name	Data Type	Description
CUBE_ID	NUMBER(9)	ID of the cube (Note, this view is redundant, it can be achieved by joining 2.23, 2.24 views. It will be removed in the future)
CUBE_NAME	VARCHAR2(255)	Physical name of the cube
MEASURE_ID	NUMBER(9)	ID of the measure belonging to this cube
MEASURE_NAME	VARCHAR2(255)	Physical name of the measure
DIMENSION_ID	NUMBER(9)	ID of the dimension associated with this cube
DIMENSION_NAME	VARCHAR2(255)	Physical name of the dimension
DIMENSION_ALIAS	VARCHAR2(255)	Alias of the dimension

Table D–26 ALL_IV_DIMENSIONS

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this dimension belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
DIMENSION_ID	NUMBER(9)	ID of this dimension
DIMENSION_NAME	VARCHAR2(255)	Physical name of this dimension
BUSINESS_NAME	VARCHAR2(4000)	Business name of this dimension

Table D–26 (Cont.) ALL_IV_DIMENSIONS

Column Name	Data Type	Description
DESCRIPTION	VARCHAR2(4000)	Description of this dimension
PLURAL_NAME	VARCHAR2(40)	Plural name of this dimension
DIMENSION_PREFIX	VARCHAR2(40)	The prefix for this dimension
IS_VALID	VARCHAR2(13)	Is this dimension valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–27 ALL_IV_DIM_HIERARCHIES

Column Name	Data Type	Description
DIMENSION_ID	NUMBER(9)	ID of the dimension this hierarchy belongs to
DIMENSION_NAME	VARCHAR2(255)	Physical name of this dimension
HIERARCHY_ID	NUMBER(9)	ID of this hierarchy
HIERARCHY_NAME	VARCHAR2(255)	Physical name of this hierarchy
HIERARCHY_PREFIX	VARCHAR2(40)	Prefix for this hierarchy
BUSINESS_NAME	VARCHAR2(4000)	Business name of this hierarchy
DESCRIPTION	VARCHAR2(4000)	Description of this hierarchy
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–28 ALL_IV_DIM_HIERARCHY_LEVELS

Column Name	Data Type	Description
LEVEL_USE_ID	NUMBER(9)	ID of the level relationship that this level and this parent level participates in
HIERARCHY_ID	NUMBER(9)	ID of the hierarchy that this level and this parent level belongs to
HIERARCHY_NAME	VARCHAR2(255)	Physical name of the hierarchy
LEVEL_ID	NUMBER(9)	ID of this level
LEVEL_NAME	VARCHAR2(255)	Physical name of this level
LEVEL_DESCRIPTION	VARCHAR2(4000)	Description of this level
PARENT_LEVEL_ID	NUMBER(9)	ID of this parent level
PARENT_LEVEL_NAME	VARCHAR2(255)	Physical name of this parent level
POSITION	NUMBER(9)	Position of this level

Table D–29 ALL_IV_DIM_LEVELS

Column Name	Data Type	Description
DIMENSION_ID	NUMBER(9)	ID of the dimension this level belongs to
DIMENSION_NAME	VARCHAR2(255)	Physical name of the dimension
LEVEL_ID	NUMBER(9)	ID of this level

Table D–29 (Cont.) ALL_IV_DIM_LEVELS

Column Name	Data Type	Description
LEVEL_NAME	VARCHAR2(255)	Physical name of this level
LEVEL_PREFIX	VARCHAR2(40)	Prefix for this level
BUSINESS_NAME	VARCHAR2(4000)	Business name of this level
DESCRIPTION	VARCHAR2(4000)	Description of this level
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–30 ALL_IV_DIM_LEVEL_ATTRIBUTES

Column Name	Data Type	Description
LEVEL_ID	NUMBER(9)	ID of the level this attribute belongs to
LEVEL_NAME	VARCHAR2(255)	Physical name of the level
ATTRIBUTE_ID	NUMBER(9)	ID of this attribute
ATTRIBUTE_NAME	VARCHAR2(255)	Physical name of this attribute
BUSINESS_NAME	VARCHAR2(4000)	Business name of this attribute
DESCRIPTION	VARCHAR2(4000)	Description of this attribute
POSITION		Position of this attribute within the level
DATA_TYPE	VARCHAR2(255)	Data type of this attribute
LENGTH	NUMBER(9)	Data length of this attribute
PRECISION	NUMBER(9)	Data precision of this attribute
SCALE	NUMBER(9)	Data scale of this attribute
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–31 ALL_IV_EXTERNAL_COLUMNS

Column Name	Data Type	Description
ENTITY_ID	NUMBER(9)	ID of the external table that this column belongs to
ENTITY_NAME	VARCHAR2(255)	Name of the external table
COLUMN_ID	NUMBER(9)	ID of this column
COLUMN_NAME	VARCHAR2(255)	Physical name of this column
BUSINESS_NAME	VARCHAR2(4000)	Business name of this column
DESCRIPTION	VARCHAR2(4000)	Description of this column
POSITION	NUMBER(9)	Position of this column within the external table
DATA_TYPE	VARCHAR2(255)	Data type of this column
LENGTH	NUMBER(9)	Data length of this column
PRECISION	NUMBER(9)	Data precision of this column
SCALE	NUMBER(9)	Data scale of this column
SOURCE_FIELD_ID	NUMBER(9)	ID of the field that this column maps to

Table D–31 (Cont.) ALL_IV_EXTERNAL_COLUMNS

Column Name	Data Type	Description
SOURCE_FIELD_NAME	VARCHAR2(255)	Physical name of the source field
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–32 ALL_IV_EXTERNAL_TABLES

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this external table belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
LOCATION_ID	NUMBER(9)	ID of the location where the module is deployed to
LOCATION_NAME	VARCHAR2(255)	Physical name of the location
TABLE_ID	NUMBER(9)	ID of the external table
TABLE_NAME	VARCHAR2(255)	Physical name of the external table
BUSINESS_NAME	VARCHAR2(4000)	Business name of the external table
DESCRIPTION	VARCHAR2(4000)	Description of the external table
SOURCE_RECORD_ID	NUMBER(9)	ID of the record that this external table maps to
SOURCE_RECORD_NAME	VARCHAR2(255)	Physical name of the source record
SOURCE_FILE_NAME	VARCHAR2(255)	Physical name of the file that this source record belongs to
ACCESS_PARAMETERS	VARCHAR2(4000)	Expression for parameters that are used to access the source record
IS_VALID	VARCHAR2(13)	Is this external table valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–33 ALL_IV_FOREIGN_KEYS

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module that this foreign key belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
ENTITY_ID	NUMBER(9)	ID of the data entity this foreign key belongs to
ENTITY_NAME	VARCHAR2(255)	Physical name of the data entity
ENTITY_TYPE	VARCHAR2(4000)	Type of the data type (for example table, view)
FOREIGN_KEY_ID	NUMBER(9)	ID of this foreign key
FOREIGN_KEY_NAME	VARCHAR2(255)	Physical name of this foreign key
BUSINESS_NAME	VARCHAR2(4000)	Business name of this foreign key
DESCRIPTION	VARCHAR2(4000)	Description of this foreign key
KEY_ID	NUMBER(9)	ID of the associated key for this foreign key
KEY_NAME	VARCHAR2(255)	Physical name of the key
IS_DISABLED	CHAR(1)	Is this foreign key disabled?

Table D–33 (Cont.) ALL_IV_FOREIGN_KEYS

Column Name	Data Type	Description
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–34 ALL_IV_KEYS

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this key belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of this module
ENTITY_ID	NUMBER(9)	ID of the data entity this key belongs to
ENTITY_NAME	VARCHAR2(255)	Physical name of the data entity
ENTITY_TYPE	VARCHAR2(4000)	Type of the data entity (for example table, view)
KEY_ID	NUMBER(9)	ID of this key
KEY_NAME	VARCHAR2(255)	Physical name of this key
BUSINESS_NAME	VARCHAR2(4000)	Business of this key
DESCRIPTION	VARCHAR2(4000)	Description of this key
IS_PRIMARY	VARCHAR2(9)	Is this key primary key?
IS_DISABLED	CHAR(1)	Is this key disabled?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–35 ALL_IV_KEY_COLUMN_USES

Column Name	Data Type	Description
KEY_ID	NUMBER(9)	ID of the key that this column is associated with
KEY_NAME	VARCHAR2(255)	Physical name of the key
KEY_TYPE	VARCHAR2(11)	Type of the key (primary, unique, foreign)
COLUMN_ID	NUMBER(9)	ID of this column
COLUMN_NAME	VARCHAR2(255)	Physical name of this column
POSITION	NUMBER(9)	Position of this column with the key

Table D–36 ALL_IV_MATERIALIZED_VIEWS

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this materialized view belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
VIEW_ID	NUMBER(9)	ID of this materialized view
VIEW_NAME	VARCHAR2(255)	Physical name of this materialized view
BUSINESS_NAME	VARCHAR2(4000)	Business name of this materialized view
DESCRIPTION	VARCHAR2(4000)	Description of this materialized view

Table D–36 (Cont.) ALL_IV_MATERIALIZED_VIEWS

Column Name	Data Type	Description
QUERY_TEXT	VARCHAR2(4000)	Textual expression of query statement for this materialized view
IS_VALID	VARCHAR2(13)	Is this materialized view valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–37 ALL_IV_OBJECT_TYPES

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this object type belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
FOLDER_ID	NUMBER(9)	ID of the folder this object type belongs to
FOLDER_NAME	VARCHAR2(255)	Physical name of the folder
OBJECT_TYPE_ID	NUMBER(9)	ID of this object type
OBJECT_TYPE_NAME	VARCHAR2(255)	Physical name of this object type
BUSINESS_NAME	VARCHAR2(4000)	Business name of this object
DESCRIPTION	VARCHAR2(4000)	Description of this object type
TYPE	VARCHAR2(40)	Type of this object type
IS_VALID	VARCHAR2(13)	Is this object type valid
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–38 ALL_IV_RECORD_FIELDS

Column Name	Data Type	Description
FIRSTCLASS_OBJECT_ID	NUMBER(9)	ID of the first class object that this record field belongs to (normally, this ID will be the same as the following relational ID)
FRISTCLASS_OBJECT_NAME	VARCHAR2(255)	Physical name of the first class object
RELATION_ID	NUMBER(9)	ID of the relational entity this record field belongs to
RELATION_NAME	VARCHAR2(255)	Physical name of the relational entity
RECORDFIELD_ID	NUMBER(9)	ID of this record field
RECORDFIELD_NAME	VARCHAR2(255)	Physical name of this record field
BUSINESS_NAME	VARCHAR2(4000)	Business name of this record field
DESCRIPTION	VARCHAR2(4000)	Description of this record field
POSITION	NUMBER(9)	Position of this record field
DATA_TYPE	VARCHAR2(255)	Data type of this record field
LENGTH	NUMBER(9)	Data length of this record field
PRECISION	NUMBER(9)	Data precision of this record field
SCALE	NUMBER(9)	Data scale of this record field

Table D–38 (Cont.) ALL_IV_RECORD_FIELDS

Column Name	Data Type	Description
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–39 ALL_IV_RELATIONS

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this relational entity belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
RELATION_ID	NUMBER(9)	ID of this relational entity
RELATION_NAME	VARCHAR2(255)	Physical name of this relational entity
RELATION_TYPE	VARCHAR2(16)	Type of this relational entity (such as table, view, sequence and materialized view)
BUSINESS_NAME	VARCHAR2(4000)	Business name of this relational entity
DESCRIPTION	VARCHAR2(4000)	Description of this relational entity
IS_VALID	VARCHAR2(13)	Is this relational entity valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–40 ALL_IV_SEQUENCES

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this sequence belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
SEQUENCE_ID	NUMBER(9)	ID of this sequence
SEQUENCE_NAME	VARCHAR2(255)	Physical name of this sequence
BUSINESS_NAME	VARCHAR2(4000)	Business name of this sequence
DESCRIPTION	VARCHAR2(4000)	Description of this sequence
IS_VALID	VARCHAR2(13)	Is this sequence valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–41 ALL_IV_VIEWS

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this view belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
VIEW_ID	NUMBER(9)	ID of this view
VIEW_NAME	VARCHAR2(255)	Physical name of this view
QUERY_TEXT	VARCHAR2(4000)	Textual expression of the query for this view

Table D–41 (Cont.) ALL_IV_VIEWS

Column Name	Data Type	Description
BUSINESS_NAME	VARCHAR2(4000)	Business name of this view
DESCRIPTION	VARCHAR2(4000)	Description of this view
IS_VALID	VARCHAR2(13)	Is this view valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–42 ALL_IV_TABLES

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this table belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
TABLE_ID	NUMBER(9)	ID of this table
TABLE_NAME	VARCHAR2(255)	Physical name of this table
BUSINESS_NAME	VARCHAR2(4000)	Business name of this table
DESCRIPTION	VARCHAR2(4000)	Description of this table
IS_VALID	VARCHAR2(13)	Is this table valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Implementation Model Views

Table D–43 ALL_IV_CUBE_IMPLS

Column Name	Data Type	Description
IMPLEMENTATION_ID	NUMBER(9)	ID of this cube (this column will be updated in future)
ITEM_ID	NUMBER(9)	ID of the item belonging to this cube
ITEM_TYPE	VARCHAR2(18)	Two kinds of type: cube measures and foreign keys pointing to dimension (called: cube dimension use)
ITEM_NAME	VARCHAR2(255)	Physical name of the item
CUBE_ID	NUMBER(9)	ID of this cube
CUBE_NAME	VARCHAR2(255)	Physical name of this cube
DIMENSION_ID	NUMBER(9)	ID of the associated dimension
DIMENSION_NAME	VARCHAR2(255)	Physical name of the associated name
DIMENSION_ALIAS	VARCHAR2(255)	Alias of the associated dimension (name of the foreign key in the cube)
COLUMN_ID	NUMBER(9)	ID of the implementing column for the item
COLUMN_NAME	VARCHAR2(255)	Physical name of the implementing column for the item
POSITION	NUMBER	Position of the implementing column
TABLE_ID	NUMBER(9)	ID of the implementing table for this cube

Table D–43 (Cont.) ALL_IV_CUBE_IMPLS

Column Name	Data Type	Description
TABLE_NAME	VARCHAR2(255)	Physical name of the implementing table
FOREIGN_KEY_ID	NUMBER(9)	ID of the foreign key pointing to the dimension
FOREIGN_KEY_NAME	VARCHAR2(255)	Physical name of the foreign key
DIM_IMPLEMENTATION_ID	NUMBER(9)	Current value set to NULL, will be updated in future

Table D–44 ALL_IV_DIM_IMPLS

Column Name	Data Type	Description
IMPLEMENTATION_ID	NUMBER(9)	ID of the item belonging to this dimension (this column will be updated in future)
ITEM_ID	NUMBER(9)	ID of the item belonging to this dimension
ITEM_TYPE	VARCHAR2(18)	Type of the item (constant value: Level Attribute)
ITEM_NAME	VARCHAR2(255)	Physical name of the item
DIMENSION_ID	NUMBER(9)	ID of the dimension
DIMENSION_NAME	VARCHAR2(255)	Physical name of the dimension
COLUMN_ID	NUMBER(9)	ID of the implementing column for the item
COLUMN_NAME	VARCHAR2(255)	Physical name of the implementing column
POSITION	NUMBER	Position of the column
TABLE_ID	NUMBER(9)	ID of the implementation table for this dimension
TABLE_NAME	VARCHAR2(255)	Physical name of the implementation table for this dimension

Table D–45 ALL_IV_DIM_LEVEL_IMPLS

Column Name	Data Type	Description
IMPLEMENTATION_ID	NUMBER(9)	ID of the item belonging to this level (this column will be updated in future)
ITEM_ID	NUMBER(9)	ID of the item belonging to this level
ITEM_TYPE	VARCHAR2(18)	Type of the item (constant value: Level Attribute)
ITEM_NAME	VARCHAR2(255)	Physical name of the item
DIMENSION_ID	NUMBER(9)	ID of the dimension
DIMENSION_NAME	VARCHAR2(255)	Physical name of the dimension
LEVEL_ID	NUMBER(9)	ID of this level
LEVEL_NAME	VARCHAR2(255)	Physical name of this level
COLUMN_ID	NUMBER(9)	ID of the implementation column
COLUMN_NAME	VARCHAR2(255)	Physical name of the implementation column
POSITION	NUMBER	Position of the column
TABLE_ID	NUMBER(9)	ID of the implementation table for this level
TABLE_NAME	VARCHAR2(255)	Physical name of the implementation table for this level

Table D–46 ALL_IV_FUNCTION_IMPLS

Column Name	Data Type	Description
FUNCTION_ID	NUMBER(9)	ID of the function
FUNCTION_NAME	VARCHAR2(255)	Physical name of the function
FUNCTION_ IMPLEMENTATION_ID	NUMBER(9)	ID of the function implementation for this function
FUNCTION_ IMPLEMENTATION_NAME	VARCHAR2(255)	Physical name of the function implementation
LANGUAGE	VARCHAR2(255)	Name of the language being used in the implementation
SCRIPT	VARCHAR2(4000)	Implementation script for this function
BUSINESS_NAME	VARCHAR2(4000)	Business name for this implementation
DESCRIPTION	VARCHAR2(4000)	Description for this implementation
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Flat Files Views

Table D–47 ALL_IV_FIELDS

Column Name	Data Type	Description
RECORD_ID	NUMBER(9)	ID of the record this field belongs to
RECORD_NAME	VARCHAR2(255)	Physical name of the record
FIELD_ID	NUMBER(9)	ID of this field
FIELD_NAME	VARCHAR2(255)	Physical name of this field
BUSINESS_NAME	VARCHAR2(4000)	Business name of this field
DESCRIPTION	VARCHAR2(4000)	Description of this field
POSITION	NUMBER(9)	Position of this field
DATA_TYPE	VARCHAR2(255)	Data type of this field
LENGTH	NUMBER(9)	Data length of this field
PRECISION	NUMBER(9)	Data precision of this field
SCALE	NUMBER(9)	Data scale of this field
PICTURE	VARCHAR2(40)	Picture of the field
SIGN_TYPE	NUMBER(9)	Sign type of the field
USAGE	VARCHAR2(40)	Usage of the field
MASK	VARCHAR2(255)	Mask of the field
NULLIF	VARCHAR2(40)	Nullif value of the field
DEFAULTIF	VARCHAR2(40)	Defaultif value of the field
SQL_DATA_TYPE	VARCHAR2(40)	SQL data type of the field
SQL_LENGTH	NUMBER(9)	SQL data length of the field
SQL_PRECISION	NUMBER(9)	SQL precision of the field
SQL_SCALE	NUMBER(9)	SQL data scale of the field
START_POSITION	VARCHAR2(40)	Start position of the field
OCCURS	NUMBER(9)	Occurs of the attribute array within the structure (identified by next column)

Table D–47 (Cont.) ALL_IV_FIELDS

Column Name	Data Type	Description
STRUCTURE_ID	NUMBER(9)	ID of the structure containing field
STRUCTURE_NAME	VARCHAR2(255)	Physical name of the structure
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–48 ALL_IV_FILES

Column Name	Data Type	Description
INFORMATION_SYSTEM_ID	NUMBER(9)	ID of the module this file belongs to
INFORMATION_SYSTEM_NAME	VARCHAR2(255)	Physical name of the module
FILE_ID	NUMBER(9)	ID of this file
FILE_NAME	VARCHAR2(255)	Physical name of this file
BUSINESS_NAME	VARCHAR2(4000)	Business name of this file
DESCRIPTION	VARCHAR2(4000)	Description of the file
FILE_FORMAT	VARCHAR2(10)	Format of this file
IS_VALID	VARCHAR2(13)	Is this file valid?
RECORD_CLASSIFIER_POSITION	NUMBER(9)	Record classifier position of this file
RECORD_CLASSIFIER_LENGTH	NUMBER(9)	Record classifier length of this file
RECORD_SIZE	VARCHAR2(40)	Record size of this file
N_PHYSICAL_RECORDS_IN_LOGICAL	NUMBER(9)	Number of physical records for each logical record
CONTINUATION_AT_END	CHAR(1)	Continuation at end or not
CONTINUATION_DELIMITER	VARCHAR2(40)	Continuation delimiter symbol
RECORD_DELIMITER	VARCHAR2(40)	Record delimiter symbol
FIELD_DELIMITER	VARCHAR2(40)	Field delimiter symbol
TEXT_START_DELIMITER	VARCHAR2(1)	Text start delimiter symbol
TEXT_END_DELIMITER	VARCHAR2(1)	Text end delimiter symbol
SOURCE_FROM	VARCHAR2(4000)	Directory path of this file
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–49 ALL_IV_RECORDS

Column Name	Data Type	Description
FILE_ID	NUMBER(9)	ID of the file this record belongs to
FILE_NAME	VARCHAR2(255)	Physical name of the file
RECORD_ID	NUMBER(9)	ID of this record
RECORD_NAME	VARCHAR2(255)	Physical name of this record

Table D–49 (Cont.) ALL_IV_RECORDS

Column Name	Data Type	Description
BUSINESS_NAME	VARCHAR2(4000)	Business name of this record
DESCRIPTION	VARCHAR2(4000)	Description of this record
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Collections Views

Table D–50 ALL_IV_COLLECTIONS

Column Name	Data Type	Description
PROJECT_ID	NUMBER(9)	ID of the project this collection belongs to (this view replaces old classification view)
PROJECT_NAME	VARCHAR2(255)	Physical name of the project
COLLECTION_ID	NUMBER(9)	ID of this collection
COLLECTION_NAME	VARCHAR2(255)	Physical name of this collection
BUSINESS_NAME	VARCHAR2(4000)	Business name of this collection
DESCRIPTION	VARCHAR2(4000)	Description of this collection
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–51 ALL_IV_COLLECTION_REFERENCES

Column Name	Data Type	Description
COLLECTION_ID	NUMBER(9)	ID of the collection this reference belongs to (this view replaces old classification_item view)
COLLECTION_NAME	VARCHAR2(255)	Physical name of the collection
COLLECTION_REFERENCE_ID	NUMBER(9)	ID of this collection reference
COLLECTION_REFERENCE_TYPE	VARCHAR2(4000)	Type of this collection reference
COLLECTION_REFERENCE_NAME	VARCHAR2(255)	Physical name of the collection reference
BUSINESS_NAME	VARCHAR2(4000)	Business name of this collection reference
DESCRIPTION	VARCHAR2(4000)	Description of this collection reference
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Function Model Views

Table D–52 ALL_IV_FUNCTIONS

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this function belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
FUNCTION_LIBRARY_ID	NUMBER(9)	ID of the function library this function belongs to
FUNCTION_LIBRARY_NAME	VARCHAR2(255)	Physical name of the function library
FUNCTION_ID	NUMBER(9)	ID of this function
FUNCTION_NAME	VARCHAR2(255)	Physical name of this function
FUNCTION_TYPE	VARCHAR2(13)	Type of this function (function, procedure, table function)
SIGNATURE	VARCHAR2(4000)	Signature of this function
IS_VALID	VARCHAR2(13)	Is this function valid?
BUSINESS_NAME	VARCHAR2(4000)	Business name of this function
DESCRIPTION	VARCHAR2(4000)	Description of this function
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–53 ALL_IV_FUNCTION_LIBRARIES

Column Name	Data Type	Description
INFORMATION_SYSTEM_ID	NUMBER(9)	ID of the module this function library belongs to
INFORMATION_SYSTEM_NAME	VARCHAR2(255)	Physical name of the module
FUNCTION_LIBRARY_ID	NUMBER(9)	ID of this function library
FUNCTION_LIBRARY_NAME	VARCHAR2(255)	Physical name of this function library
FUNCTION_LIBRARY_TYPE	VARCHAR2(40)	Type of this function library
IS_VALID	VARCHAR2(13)	Is this function library valid?
BUSINESS_NAME	VARCHAR2(4000)	Business name of this function library
DESCRIPTION	VARCHAR2(4000)	Description of this function library
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–54 ALL_IV_FUNCTION_PARAMETERS

Column Name	Data Type	Description
FUNCTION_ID	NUMBER(9)	ID of the function this parameter belongs to
FUNCTION_NAME	VARCHAR2(255)	Physical name of the function
PARAMETER_ID	NUMBER(9)	ID of this parameter
PARAMETER_NAME	VARCHAR2(255)	Physical name of this parameter
PARAMETER_TYPE	VARCHAR2(40)	Type of this parameter
BUSINESS_NAME	VARCHAR2(4000)	Business name of this parameter

Table D–54 (Cont.) ALL_IV_FUNCTION_PARAMETERS

Column Name	Data Type	Description
DESCRIPTION	VARCHAR2(4000)	Description of this parameter
POSITION	NUMBER(9)	Position of this parameter within the function
DATA_TYPE	VARCHAR2(255)	Data type of this parameter
LENGTH	NUMBER(9)	Data length of this parameter
PRECISION	NUMBER(9)	Data precision of this parameter
SCALE	NUMBER(9)	Data scale of this parameter
DEFAULT_VALUE	VARCHAR2(4000)	Default value of this parameter
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–55 ALL_IV_TABLE_FUNCTIONS

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this table function belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
FUNCTION_LIBRARY_ID	NUMBER(9)	ID of the function library this table function belongs to
FUNCTION_LIBRARY_NAME	VARCHAR2(255)	Physical name of the function library
FUNCTION_ID	NUMBER(9)	ID of this table function
FUNCTION_NAME	VARCHAR2(255)	Physical name of this table function
FUNCTION_TYPE	VARCHAR2(13)	Type of this table function (constant value: Table Function)
SIGNATURE	VARCHAR2(4000)	Signature of this table function
IS_VALID	VARCHAR2(13)	Is this table function valid?
BUSINESS_NAME	VARCHAR2(4000)	Business name of this table function
DESCRIPTION	VARCHAR2(4000)	Description of this table function
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Configuration Model Views

Table D–56 ALL_IV_OBJECT_CONFIGURATIONS

Column Name	Data Type	Description
CONFIGURED_OBJECT_ID	NUMBER(9)	ID of the object being configured
CONFIGURED_OBJECT_NAME	VARCHAR2(255)	Physical name of the object
CONFIGURED_OBJECT_TYPE	VARCHAR2(4000)	Type of the object
CONFIGURATION_PARAMETER_KEY	VARCHAR2(128)	Key of configuration parameter
CONFIGURATION_PARAMETER_NAME	VARCHAR2(64)	Name of configuration parameter

Table D–56 (Cont.) ALL_IV_OBJECT_CONFIGURATIONS

Column Name	Data Type	Description
PARAMETER_NLSKEY	VARCHAR2(64)	National Language Support (NLS) key of the parameter
CONFIGURATION_ PARAMETER_TYPE	CHAR(23)	Type of the configuration parameter
ARGUMENT	VARCHAR2(128)	Value of the configuration parameter
GROUP_NAME	VARCHAR2(322)	Name of the configuration group
GROUP_NLSKEY	VARCHAR2(64)	National Language Support (NLS) key of the configuration group
LANGUAGE	VARCHAR2(64)	Name of the language being used for this configuration

Deployment Model Views

Table D–57 ALL_IV_CONNECTORS

Column Name	Data Type	Description
LOCATION_ID	NUMBER(9)	ID of the location owning this connector
LOCATION_NAME	VARCHAR2(255)	Physical name of the location
CONNECTOR_ID	NUMBER(9)	ID of this connector
CONNECTOR_NAME	VARCHAR2(255)	Physical name of this connector
BUSINESS_NAME	VARCHAR2(4000)	Business name of this connector
DESCRIPTION	VARCHAR2(4000)	Description of this connector
REFERENCED_LOCATION_ID	NUMBER(9)	ID of the location this connector references to
REFERENCED_LOCATION_ NAME	VARCHAR2(255)	Physical name of the location this connector references to
IS_VALID	VARCHAR2(13)	Is this connector valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–58 ALL_IV_LOCATIONS

Column Name	Data Type	Description
PROJECT_ID	NUMBER(9)	ID of the project this location belongs to
PROJECT_NAME	VARCHAR2(255)	Physical name of the project
LOCATION_ID	NUMBER(9)	ID of this location
LOCATION_NAME	VARCHAR2(255)	Physical name of this location
LOCATION_TARGET_TYPE	VARCHAR2(40)	Target type of this location
LOCATION_TARGET_ VERSION	VARCHAR2(40)	Target version of this location
APPLICATION_TYPE	VARCHAR2(255)	Application type of the location connected to
SYSTEM_TYPE	VARCHAR2(255)	System type of this location connected to
IS_VALID	VARCHAR2(13)	Is this location valid?
BUSINESS_NAME	VARCHAR2(4000)	Business name of this location
DESCRIPTION	VARCHAR2(4000)	Description of this location
UPDATED_ON	DATE	Update timestamp

Table D–58 (Cont.) ALL_IV_LOCATIONS

Column Name	Data Type	Description
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–59 ALL_IV_RUNTIME_REPOSITORIES

Column Name	Data Type	Description
PROJECT_ID	NUMBER(9)	ID of the project this runtime repository belongs to (also called runtime location, or simply, location)
PROJECT_NAME	VARCHAR2(255)	Physical name of the project
LOCATION_ID	NUMBER(9)	ID of this runtime location
LOCATION_NAME	VARCHAR2(255)	Physical name of this runtime location
LOCATION_TYPE	VARCHAR2(255)	Type of this runtime location
APPLICATION_TYPE	VARCHAR2(255)	Type of the application this location connected to
SYSTEM_TYPE	VARCHAR2(255)	Type of the system this location connected to
BUSINESS_NAME	VARCHAR2(4000)	Business name of this runtime location
DESCRIPTION	VARCHAR2(4000)	Description of this runtime location
HOST	VARCHAR2(40)	Host name of the connection for this location
SERVICE_NAME	VARCHAR2(40)	Service name of the connection for this location
PORT	NUMBER(9)	Port of the connection for this location
USERNAME	VARCHAR2(40)	User name of the connection for this location
SCHEMA	VARCHAR2(40)	Schema name of the connection for this location
IS_VALID	VARCHAR2(13)	Is this runtime location valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Mapping Model Views

Table D–60 ALL_IV_XFORM_MAPS

Column Name	Data Type	Description
INFORMATION_SYSTEM_ID	NUMBER(9)	ID of the module this map belongs to
INFORMATION_SYSTEM_NAME	VARCHAR2(255)	Physical name of the module
MAP_ID	NUMBER(9)	ID of this map
MAP_NAME	VARCHAR2(255)	Physical name of this map
BUSINESS_NAME	VARCHAR2(4000)	Business name of this map
DESCRIPTION	VARCHAR2(4000)	Description of this map
COMPOSITE_MAP_COMPONENT_ID	NUMBER(9)	ID of this map (this column is redundant, will be removed in future)
COMPOSITE_MAP_COMPONENT_NAME	VARCHAR2(255)	Physical name of this map (this column is redundant, will be removed in future)
IS_VALID	VARCHAR2(13)	Is this map valid?

Table D–60 (Cont.) ALL_IV_XFORM_MAPS

Column Name	Data Type	Description
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–61 ALL_IV_XFORM_MAP_COMPONENT

Column Name	Data Type	Description
MAP_ID	NUMBER(9)	ID of the map this map component belongs to
MAP_NAME	VARCHAR2(255)	Physical name of the map
MAP_COMPONENT_ID	NUMBER(9)	ID of this map component (also called map operator)
MAP_COMPONENT_NAME	VARCHAR2(255)	Physical name of this map component
BUSINESS_NAME	VARCHAR2(4000)	Business name of this map component
DESCRIPTION	VARCHAR2(4000)	Description of this map component
OPERATOR_TYPE	VARCHAR2(4000)	Type of this map component (for example Filter, Joiner, Table)
COMPOSITE_MAP_ COMPONENT_ID	NUMBER(9)	ID of the map (this column is redundant, will be removed in future)
COMPOSITE_MAP_ COMPONENT_NAME	VARCHAR2(255)	Physical name of the map (this column is redundant, will be removed in future)
DATA_ENTITY_ID	NUMBER(9)	ID of the data entity this map component reconciled to
DATA_ENTITY_NAME	VARCHAR2(255)	Physical name of the data entity
DATA_ENTITY_TYPE	VARCHAR2(4000)	Type of the data entity
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–62 ALL_IV_XFORM_MAP_DETAILS

Column Name	Data Type	Description
MAP_COMPONENT_ID	NUMBER(9)	ID of the map component
PARAMETER_ID	NUMBER(9)	ID of the parameter owned by this map component
POSITION	NUMBER(9)	Position of the parameter
BUSINESS_NAME	VARCHAR2(4000)	Business name of the parameter
PARAMETER_NAME	VARCHAR2(255)	Physical name of the parameter
TRANSFORMATION_ EXPRESSION	VARCHAR2(4000)	Textual expression of the transformation of this parameter
DESCRIPTION	VARCHAR2(4000)	Description of this parameter
SOURCE_EXPRESSION	VARCHAR2(4000)	Data source expression of this parameter

Table D–63 ALL_IV_XFORM_MAP_PARAMETERS

Column Name	Data Type	Description
MAP_COMPONENT_ID	NUMBER(9)	ID of the map component this parameter belongs to
MAP_COMPONENT_NAME	VARCHAR2(255)	Physical name of the map component
PARAMETER_ID	NUMBER(9)	ID of this parameter
PARAMETER_NAME	VARCHAR2(255)	Physical name of this parameter
BUSINESS_NAME	VARCHAR2(4000)	Business name of this parameter
DESCRIPTION	VARCHAR2(4000)	Description of this parameter
MAP_ID	NUMBER(9)	ID of the map containing the map component
MAP_NAME	VARCHAR2(255)	Physical name of the map
PARAMETER_GROUP_NAME	VARCHAR2(255)	Physical name of the parameter group name
PARAMETER_GROUP_ID	NUMBER(9)	ID of the parameter group
PARAMETER_TYPE	VARCHAR2(5)	Type of the parameter(IN, OUT, INOUT)
POSITION	NUMBER(9)	Position of the parameter within the group
DATA_TYPE	VARCHAR2(40)	Data type of the parameter
TRANSFORMATION_ EXPRESSION	VARCHAR2(4000)	Textual expression of the transformation for this parameter
DATA_ITEM_ID	NUMBER(9)	ID of the data item this parameter reconciled to
DATA_ITEM_TYPE	VARCHAR2(40)	Type of the data item
DATA_ITEM_NAME	VARCHAR2(255)	Physical name of the data item
SOURCE_PARAMETER_ID	NUMBER(9)	ID of the source parameter (where this parameter connected from)
SOURCE_PARAMETER_ NAME	VARCHAR2(255)	Physical name of the source parameter
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–64 ALL_IV_XFORM_MAP_MAP_PROPERTIES

Column Name	Data Type	Description
MAP_COMPONENT_ID	NUMBER(9)	ID of the map component this property belongs to
MAP_COMPONENT_NAME	VARCHAR2(255)	Physical name of the map component
PROPERTY_ID	NUMBER(9)	ID of this property
PROPERTY_NAME	VARCHAR2(255)	Physical name of this property
BUSINESS_NAME	VARCHAR2(4000)	Business name of this property
DESCRIPTION	VARCHAR2(4000)	Description of this property
PROPERTY_GROUP_NAME	VARCHAR2(255)	Physical name of this property group
PROPERTY_VALUE	VARCHAR2(4000)	Value of this property

Process Flow Model Views

Table D–65 ALL_IV_PACKAGES

Column Name	Data Type	Description
SCHEMA_ID	NUMBER(9)	ID of the module this process package belongs to
SCHEMA_NAME	VARCHAR2(255)	Physical name of the module
PACKAGE_ID	NUMBER(9)	ID of this process package
PACKAGE_NAME	VARCHAR2(255)	Physical name of this process package
BUSINESS_NAME	VARCHAR2(4000)	Business name of this process package
DESCRIPTION	VARCHAR2(4000)	Description of this process package
IS_VALID	VARCHAR2(13)	Is this process package valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–66 ALL_IV_PROCESSES

Column Name	Data Type	Description
PACKAGE_ID	NUMBER(9)	ID of the process package this process belongs to
PACKAGE_NAME	VARCHAR2(255)	Physical name of the process package
PARENT_PROCESS_ID	NUMBER(9)	ID of the parent process for this process
PARENT_PROCESS_NAME	VARCHAR2(255)	Physical name of the parent process for this process
PROCESS_ID	NUMBER(9)	ID of this process
PROCESS_NAME	VARCHAR2(255)	Physical name of this process
BUSINESS_NAME	VARCHAR2(4000)	Business name of this process
DESCRIPTION	VARCHAR2(4000)	Description of this process
BOUND_OBJECT_ID	NUMBER(9)	ID of the bound object
BOUND_OBJECT_NAME	VARCHAR2(255)	Name of the bound object
IS_VALID	VARCHAR2(13)	Is this process valid?
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–67 ALL_IV_PROCESS_ACTIVITIES

Column Name	Data Type	Description
PROCESS_ID	NUMBER(9)	ID of the process this activity belongs to
PROCESS_NAME	VARCHAR2(255)	Physical name of the process
ACTIVITY_ID	NUMBER(9)	ID of this process activity
ACTIVITY_NAME	VARCHAR2(255)	Physical name of this activity
ACTIVITY_TYPE	VARCHAR2(4000)	Type of this activity
BOUND_OBJECT_ID	NUMBER(9)	ID of the bound object
BOUND_OBJECT_NAME	VARCHAR2(255)	Name of the bound object

Table D–67 (Cont.) ALL_IV_PROCESS_ACTIVITIES

Column Name	Data Type	Description
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–68 ALL_IV_PROCESS_PARAMETERS

Column Name	Data Type	Description
PARAMETER_OWNER_ID	NUMBER(9)	ID of the owning object for this parameter
PARAMETER_OWNER_NAME	VARCHAR2(255)	Physical name of the owning object for this parameter
PARAMETER_OWNER_TYPE	CHAR(14)	Type of the owning object
PARAMETER_ID	NUMBER(9)	ID of this parameter
PARAMETER_NAME	VARCHAR2(255)	Physical name of this parameter
BUSINESS_NAME	VARCHAR2(4000)	Business name of this parameter
DESCRIPTION	VARCHAR2(4000)	Description of this parameter
POSITION	NUMBER(9)	Position of this parameter
DATA_TYPE	VARCHAR2(40)	Data type of this parameter
DEFAULT_VALUE	VARCHAR2(4000)	Default value for this parameter
DIRECTION	VARCHAR2(3)	Direction of this parameter (IN, OUT)
IS_FINAL	CHAR(1)	Is process final
BOUNDDATA_ID	NUMBER(9)	ID of the bound data for this parameter
BOUNDDATA_NAME	VARCHAR2(255)	Physical name of the bound data
BOUNDDATA_TYPE	VARCHAR2(40)	Type of the bound data
BOUNDDATA_VALUE	VARCHAR2(4000)	Value of the bound data
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–69 ALL_IV_PROCESS_TRANSITIONS

Column Name	Data Type	Description
PROCESS_ID	NUMBER(9)	ID of the process this transition belongs to
PROCESS_NAME	VARCHAR2(255)	Physical name of the process
TRANSITION_ID	NUMBER(9)	ID of this transition
TRANSITION_NAME	VARCHAR2(255)	Physical name of this transition
BUSINESS_NAME	VARCHAR2(4000)	Business name of this transition
DESCRIPTION	VARCHAR2(4000)	Description of this transition
CONDITION	VARCHAR2(255)	Condition of this transition
TRANSITION_ORDER	NUMBER(9)	Order of this transition
SOURCE_ACTIVITY_ID	NUMBER(9)	ID of the source activity for this transition
SOURCE_ACTIVITY_NAME	VARCHAR2(255)	Physical name of the source activity
TARGET_ACTIVITY_ID	NUMBER(9)	ID of the target activity for this transition

Table D–69 (Cont.) ALL_IV_PROCESS_TRANSITIONS

Column Name	Data Type	Description
TARGET_ACTIVITY_NAME	VARCHAR2(255)	Physical name of the target activity
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Table D–70 ALL_IV_PROCESS_VARIABLES

Column Name	Data Type	Description
PROCESS_ID	NUMBER(9)	ID of the process this variable belongs to
PROCESS_NAME	VARCHAR2(255)	Physical name of the process
VARIABLE_ID	NUMBER(9)	ID of this process variable
VARIABLE_NAME	VARCHAR2(255)	Physical name of this process variable
BUSINESS_NAME	VARCHAR2(4000)	Business name of this process variable
DESCRIPTION	VARCHAR2(4000)	Description of this process variable
POSITION	NUMBER(9)	Position of this variable
DATA_TYPE	VARCHAR2(40)	Data type of this variable
DEFAULT_VALUE	VARCHAR2(4000)	Default value of this variable
IS_FINAL	CHAR(1)	Is process final
UPDATED_ON	DATE	Update timestamp
CREATED_ON	DATE	Creation timestamp
UPDATED_BY	VARCHAR2(255)	Updated by who
CREATED_BY	VARCHAR2(255)	Created by who

Warehouse Builder Runtime Repository Public Views

The Runtime Repository contains all of the deployment and execution audit data. Use these Public Views to access this data. These views are used by Runtime Audit Browser to provide audit reporting.

Deployment Auditing Views

- ALL_RT_AUDIT_LOCATIONS on page D-35
- ALL_RT_AUDIT_LOCATION_MESSAGES on page D-36
- ALL_RT_AUDIT_LOCATION_FILES on page D-36
- ALL_RT_AUDIT_OBJECTS on page D-36
- ALL_RT_AUDIT_SCRIPT_RUNS on page D-37
- ALL_RT_AUDIT_SCRIPT_MESSAGES on page D-37
- ALL_RT_AUDIT_SCRIPT_FILES on page D-37

Execution Auditing Views

- ALL_RT_AUDIT_EXECUTIONS on page D-38
- ALL_RT_AUDIT_EXECUTION_PARAMS on page D-38
- ALL_RT_AUDIT_EXEC_MESSAGES on page D-39

- ALL_RT_AUDIT_EXEC_FILES on page D-39
- ALL_RT_AUDIT_MAP_RUNS on page D-39
- ALL_RT_AUDIT_MAP_RUN_SOURCES on page D-40
- ALL_RT_AUDIT_MAP_RUN_TARGETS on page D-40
- ALL_RT_AUDIT_STEP_RUNS on page D-40
- ALL_RT_AUDIT_STEP_RUN_SOURCES on page D-41
- ALL_RT_AUDIT_STEP_RUN_TARGETS on page D-41
- ALL_RT_AUDIT_MAP_RUN_ERRORS on page D-42
- ALL_RT_AUDIT_MAP_RUN_TRACE on page D-42

Deployment Auditing Views

Table D–71 ALL_RT_AUDIT_LOCATIONS

Column Name	Type	Description
location_audit_id	Number (22)	Internal primary key to audit_location.
runtime_version	Varchar2 (64)	Runtime version number.
client_version	Varchar2 (64)	Design client version number.
client_repository	Varchar2 (30)	Name of the client repository.
client_repository_version	Varchar2 (64)	Client repository version number.
repository_user	Varchar2 (30)	Username of the design repository.
generation_time	Date	When the deployment was generated.
deployment_audit_id	Number (22)	Internal audit ID of the deployment.
deployment_sequence_number	Number (10)	Sequence number of this location in the deployment.
deployment_audit_name	Varchar2 (64)	Audit name of the location.
deployment_audit_status	Varchar2	INACTIVE, READY, or COMLETE.
location_audit_status	Varchar2	INACTIVE, READY, BUSY_PREPARE, BUSY_UNPREPARE, BUSY_DEPLOY, BUSY_UNDO, BUSY_FINALIZE, or COMPLETE.
location_uoid	Varchar2 (32)	Client UOID of the location.
location_name	Varchar2 (64)	Name of the location.
location_type	Varchar2 (64)	Type of the location. (ODB, OWF, OEM)
location_type_version	Varchar2 (64)	Version of the target.
number_script_run_errors	Number (10)	Number of errors detected.
number_script_run_warnings	Number (10)	Number of warnings detected.
created_on	Date	The time audit data was created.
created_by	Varchar2 (30)	Database username.
updated_on	Date	The time audit data was updated.
updated_by	Varchar2 (30)	Database username.

Table D–72 ALL_RT_AUDIT_LOCATION_MESSAGES

Column Name	Type	Description
message_audit_id	Number (22)	Internal key to audit_location_message. Primary when used with message_line_number.
location_audit_id	Number (22)	Internal key to audit_location.
message_severity	Varchar2	INFORMATIONAL, WARNING, ERROR, or RECOVERY.
message_line_number	Number (10)	1 for single line messages. >0 for multiple line messages. (Forms primary key when used with message_audit_id)
message_text	Varchar2 (4000)	plain_text or nls_key
created_on	Date	The time audit data was created.
created_by	Varchar2 (30)	Database username.

Table D–73 ALL_RT_AUDIT_LOCATION_FILES

Column Name	Type	Description
file_audit_id	Number (22)	Internal primary key to audit_location_file.
location_audit_id	Number (22)	Internal key to audit_location.
file_type	Varchar2 (64)	SQLLoaderLogFile, ShellOutputStream, ShellErrorStream, FTPOutputStream, or FTPErrorStream.
file_text	CLOB	Contents of the file.
format	Varchar2	TEXT or HTML.
created_on	Date	The time audit data was created.
created_by	Varchar2 (30)	Database username.

Table D–74 ALL_RT_AUDIT_OBJECTS

Column Name	Type	Description
object_audit_id	Number (22)	Internal primary key to audit_object.
parent_object_audit_id	Number (22)	Internal key to parent audit_script_run.
location_audit_id	Number (22)	Internal key to audit_location.
location_sequence_number	Number (10)	Sequence number of this object in the location.
object_uoid	Varchar2 (32)	UOID of the deployed object.
object_name	Varchar2 (64)	Name of the deployed object.
object_type	Varchar2 (64)	Type of deployed object. (PLSQLMap, Table, Dimension, SQLLoaderControlFile)
client_version_tag	Varchar2 (80)	Client version identifier of this object.
number_script_run_errors	Number (10)	Number of errors detected.
number_script_run_warnings	Number (10)	Number of warnings detected.
status_when_deployed	Varchar2	VALID, INVALID, REMOVED, or UNCERTAIN.
created_on	Date	The time audit data was created.
created_by	Varchar2 (30)	Database username.
update_on	Date	The time audit data was updated.
updated_by	Varchar2 (30)	Database username.

Table D–75 ALL_RT_AUDIT_SCRIPT_RUNS

Column Name	Type	Description
script_run_audit_id	Number (22)	Internal primary key to audit_script_run.
location_audit_id	Number (22)	Internal key to audit_location.
object_audit_id	Number (22)	Internal key to audit_object.
script_run_audit_status	Varchar2	BUSY, COMPLETE, UNCERTAIN, FAILED or INACTIVE.
operation	Varchar2	DEPLOY, or UNDO.
script_action	Varchar2	CREATE, DROP, UPGRADE or REPORT.
script	CLOB	Script used to perform the action.
script_format	Varchar2	TEXT or HTML.
script_generation_time	Date	The time the script was created.
number_script_run_errors	Number	The number of errors detected.
number_script_run_warnings	Number	The number of warnings detected.
elapse_time	Number (10)	The number of seconds that elapsed.
created_on	Date	The time the audit data was created.
created_by	Varchar2 (30)	Database username.
updated_on	Date	The time the audit data updated.
updated_by	Varchar2 (30)	Database username.

Table D–76 ALL_RT_AUDIT_SCRIPT_MESSAGES

Column Name	Type	Description
message_audit_id	Number (22)	Internal primary key to audit_script_file.
script_run_audit_id	Number (22)	Internal key to audit_script_run.
message_severity	Varchar2	INFORMATIONAL, WARNING, ERROR, or RECOVERY.
message_line_number	Number (10)	1 for single line messages. >0 for multiple line messages. (Forms primary key when used with message_audit_id)
message_text	Varchar2 (4000)	plain_text or nls_key
created_on	Date	The time the audit data was created.
created_by	Varchar2 (30)	Database username.

Table D–77 ALL_RT_AUDIT_SCRIPT_FILES

Column Name	Type	Description
file_audit_id	Number (22)	Internal primary key to audit_script_file.
script_run_audit_id	Number (22)	Internal key to audit_script_run.
file_type	Varchar2 (64)	SQLLoaderLogFile, ShellOutputStream, ShellErrorStream, FTPOutputStream, or FTPErrorStream.
file_text	CLOB	Contents of the file.
format	Varchar2	TEXT or HTML.
created_on	Date	The time the audit data was created.
created_by	Varchar2 (30)	Database username.

Execution Auditing Views

Table D–78 ALL_RT_AUDIT_EXECUTIONS

Column Name	Type	Description
execution_audit_id	Number (22)	Internal primary key to audit_execution.
parent_execution_audit_id	Number (22)	Internal key to parent audit_execution.
top_level_execution_audit_id	Number (22)	Internal key to top-level audit_execution.
execution_name	Varchar2 (64)	Name of the execution run.
task_name	Varchar2 (64)	Name of the task executed.
task_type	Varchar2 (64)	Type of task executed. (PL/SQL, ProcessFlow)
task_input	CLOB	Input stream for the task.
exec_location_uoid	Varchar2 (32)	UOID of the location where execution is performed.
exec_location_name	Varchar2 (64)	Name of the location where execution is performed.
exec_location_type	Varchar2 (64)	Type of the location where execution is performed. (Runtime Platform, OEM)
exec_location_type_version	Varchar2 (64)	Version of the location where execution is performed.
object_uoid	Varchar2 (32)	Client UOID of mapping executed.
object_name	Varchar2 (64)	Name of mapping executed.
object_type	Varchar2 (64)	Type of mapping executed.
object_location_uoid	Varchar2 (32)	Location UOID where mapping deployed.
object_location_name	Varchar2 (64)	Location name where mapping deployed.
object_location_type	Varchar2 (64)	Location type where mapping deployed.
object_location_type_version	Varchar2 (64)	Location version where mapping deployed.
return_result	Varchar2 (64)	FAILURE, OK, OK_WITH_WARNINGS, or OK_ WITH_ERRORS.
return_code	Number (10)	<0 : Failure >= 0 : Success
execution_audit_status	Varchar2	INACTIVE, BUSY, READY or COMPLETE.
elapse_time	Number (10)	Number of seconds elapsed.
number_task_errors	Number (10)	Number of errors detected.
number_task_warnings	Number (10)	Number of warnings detected.
created_on	Date	The time the audit data was created.
created_by	Varchar2 (30)	Database username.
updated_on	Date	The time the audit data was updated.
updated_by	Varchar2 (30)	Database username.

Table D–79 ALL_RT_AUDIT_EXECUTION_PARAMS

Column Name	Type	Description
parameter_audit_id	Number (22)	Internal primary key to audit_execution_param.
execution_audit_id	Number (22)	Internal key to audit_execution.
custom_parameter_uoid	Varchar2 (32)	UOID of custom parameter.
parameter_name	Varchar2 (64)	Name of parameter.

Table D–79 (Cont.) ALL_RT_AUDIT_EXECUTION_PARAMS

Column Name	Type	Description
parameter_type	Varchar2	BOOLEAN, CHAR, DATE, FLOAT, NUMBER, VARCHAR, VARCHAR2, OPERATING_MODE or AUDIT_LEVEL.
parameter_kind	Varchar2	SYSTEM or CUSTOM.
parameter_mode	Varchar2	IN, OUT or INOUT.
value_kind	Varchar2 (12)	INPUT VALUE or OUTPUT VALUE.
value	Varchar2 (4000)	Character representation of parameter value.

Table D–80 ALL_RT_AUDIT_EXEC_MESSAGES

Column Name	Type	Description
message_audit_id	Number (22)	Internal key to audit_exec_message. Primary key when used with message_line_number.
execution_audit_id	Number (22)	Internal key to audit_execution.
message_severity	Varchar2	INFORMATIONAL, WARNING, ERROR or RECOVERY.
message_line_number	Number (10)	1 for single line messages. >0 for multiple line messages. (Forms primary key when used with message_ audit_id))
message_text	Varchar2 (4000)	Plain_text or nls_key.
created_on	Date	The time the audit data was created.
created_by	Varchar2 (30)	Database username.

Table D–81 ALL_RT_AUDIT_EXEC_FILES

Column Name	Type	Description
file_audit_id	Number (22)	Internal primary key to audit_exec_file.
execution_audit_id	Number (22)	Internal key to audit_execution.
file_type	Varchar2 (64)	Type of the file.
file_text	Clob	Content of the file.
format	Varchar2	TEXT or HTML.
created_on	Date	The time the audit data was created.
created_by	Varchar2 (30)	Database username.

Table D–82 ALL_RT_AUDIT_MAP_RUNS

Column Name	Type	Description
map_run_id	Number (22)	Internal primary key to audit_map_run
execution_audit_id	Number (22)	Internal key to audit_execution
map_uoid	Varchar2 (255)	UOID of the mapping.
map_name	Varchar2 (80)	Name of the mapping.
map_type	Varchar2 (30)	PLSQLMap or SQLLoaderControlFile.
start_time	Date	The time the mapping started.
end_time	Date	The time the mapping ended.
elapse_time	Number (10)	Number of seconds elapsed.
run_status	Varchar2 (8)	RUNNING, FAILURE or COMPLETE.

Table D–82 (Cont.) ALL_RT_AUDIT_MAP_RUNS

Column Name	Type	Description
physical_name	Varchar2 (80)	Full hierarchic name of .dat file for a SQL*Loader run.
load_date	Varchar2 (30)	Load date for a SQL*Loader run.
load_time	Varchar2 (30)	Load time for a SQL*Loader run.
number_errors	Number (10)	Number of errors detected.
number_records_selected	Number (10)	Number of records selected from source tables.
number_records_inserted	Number (10)	Number of records inserted into target tables.
number_records_updated	Number (10)	Number of records updated in target tables.
number_records_deleted	Number (10)	Number of records deleted in target tables.
number_records_discarded	Number (10)	Number of records discarded in SQL*Loader run.
number_records_merged	Number (10)	Number of records merged in target tables.
number_records_corrected	Number (10)	Number of records corrected in target tables.
created_on	Date	The time the audit data was created.
created_by	Varchar2 (30)	Database username.
updated_on	Date	The time the audit data was updated.
updated_by	Varchar2 (30)	Database username.

Table D–83 ALL_RT_AUDIT_MAP_RUN_SOURCES

Column Name	Type	Description
map_run_id	Number (22)	Internal key to audit_map_run.
source_name	Varchar2 (2000)	Name of mapping operator representing source table.
source_dblink	Varchar2 (2000)	Name of database link for mapping operator representing source table.

Table D–84 ALL_RT_AUDIT_MAP_RUN_TARGETS

Column Name	Type	Description
map_run_id	Number (22)	Internal key to audit_map_run.
target_name	Varchar2 (2000)	Name of mapping operator representing target table.

Table D–85 ALL_RT_AUDIT_STEP_RUNS

Column Name	Type	Description
step_id	Number (22)	Internal primary key to audit_step_run.
map_run_id	Number (22)	Internal key to audit_map_run.
map_step	Number (22)	Step number 0 or 1. For a PL/SQL mapping, this number is normally 0 for a set-based run, and 1 for a row-based, or row-based-target run.
step_name	Varchar2 (80)	Name of the mapping for a set-based run, or the name of a mapping object for a set-based or set-based-target run.
step_type	Varchar2 (18)	Set-based, Row-based or Row-based target.
start_time	Date	The time the mapping step started.

Table D–85 (Cont.) ALL_RT_AUDIT_STEP_RUNS

Column Name	Type	Description
end_time	Date	The time the mapping step ended.
elapse_time	Number (10)	Number of seconds taken.
run_status	Varchar2 (8)	RUNNING or COMPLETE.
number_errors	Number (10)	Number of errors detected.
number_records_selected	Number (10)	Number of records selected from source tables.
number_records_inserted	Number (10)	Number of records inserted into target tables.
number_records_updated	Number (10)	Number of records updated in target tables.
number_records_deleted	Number (10)	Number of records deleted in target tables.
number_records_discarded	Number (10)	Number of records discarded in a SQL*Loader run.
number_records_merged	Number (10)	Number of records merged in target tables.
number_records_corrected	Number (10)	Number of records corrected in target tables.
created_on	Date	The time the audit data was created.
created_by	Varchar2 (30)	Database username.
updated_on	Date	The time the audit data was updated.
updated_by	Varchar2 (30)	Database username.

Table D–86 ALL_RT_AUDIT_STEP_RUN_SOURCES

Column Name	Type	Description
step_id	Number (22)	Internal key to audit_step_run.
map_run_id	Number (22)	Internal key to audit_map_run.
map_step	Number (22)	Step number 0 or 1. For a PL/SQL mapping, this number is normally 0 for a set-based run, and 1 for a row-based, or row-based-target run.
source_name	Varchar2 (2000)	Name of mapping operator representing source table.
source_dblink	Varchar2 (2000)	Name of database link for mapping operator representing source table.

Table D–87 ALL_RT_AUDIT_STEP_RUN_TARGETS

Column Name	Type	Description
step_id	Number (22)	Internal key to audit_step_run.
map_run_id	Number (22)	Internal key to audit_map_run.
map_step	Number (22)	Step number 0 or 1. For a PL/SQL mapping, this number is normally 0 for a set-based run, and 1 for a row-based, or row-based-target run.
target_name	Varchar2 (2000)	Name of mapping operator representing target table.

Table D–88 ALL_RT_AUDIT_MAP_RUN_ERRORS

Column Name	Type	Description
run_error_id	Number (22)	Internal primary key for map_run_error.
step_id	Number (22)	Internal key to audit_step_run.
map_run_id	Number (22)	Internal key to audit_map_run.
map_step	Number (22)	Step number 0 or 1. For a PL/SQL mapping, this number is normally 0 for a set-based run, and 1 for a row-based, or row-based-target run.
cursor_rowkey	Number (22)	Value identifying row returned by cursor. This is 0 for errors in a set-based run.
run_error_number	Number (10)	Message number.
run_error_message	Varchar2 (2000)	Message text.
target_name	Varchar2 (80)	Name of mapping operator representing target table.
target_column	Varchar2 (80)	Column name, or '*' if not known or not applicable.
statement	Varchar2 (2000)	Value such as INSERT or BATCH INSERT, or a PL/SQL statement.
created_on	Date	The time the audit data was created.
created_by	Varchar2 (30)	Database username.
updated_on	Date	The time the audit data was updated.
updated_by	Varchar2 (30)	Database username.

Table D–89 ALL_RT_AUDIT_MAP_RUN_TRACE

Column Name	Type	Description
trace_id	Number (22)	Internal primary key for map_run_trace.
map_run_id	Number (22)	Internal key to audit_map_run.
map_step	Number (22)	Step number 0 or 1. For a PL/SQL mapping, this number is normally 0 for a set-based run, and 1 for a row-based, or row-based-target run.
cursor_rowkey	Number (22)	Value identifying error row returned by cursor. This is 0 for set-based run.
type	Varchar2 (30)	NEW for trace or ERROR for error.
role	Varchar2 (30)	S for source or T for target.
action	Varchar2 (30)	Value such as SELECT or a PL/SQL statement.
table_name	Varchar2 (80)	Name of mapping operator representing source/target table.
created_on	Date	The time the audit data was created.
created_by	Varchar2 (30)	Database username.
updated_on	Date	The time the audit data was updated.
updated_by	Varchar2 (30)	Database username.

E

Public Objects

Warehouse Builder architecture comprises several classes of objects: First Class Objects, Second Class Objects, and so on. You can create user-defined properties and metadata snapshots on various Class Definition objects in the model that describes the Oracle Warehouse Builder metadata.

This appendix includes the following topics:

About First Class Objects

A First Class Object (FCO) represents a component in the metadata repository that can be manipulated through the Warehouse Builder interface. First Class Objects often, but not always, own other objects. For example, a `TABLE` is a First Class Object that may own the following second class objects: `TABLE_COLUMN`, `UNIQUE_KEY`, `FOREIGN_KEY`, and `CHECK_CONSTRAINT`.

As a rule of thumb for those accessing Warehouse Builder through the graphic user interface, first class objects generally appear on the navigation tree. Similarly, users who access Warehouse Builder through OMB Plus can generalize FCOs as objects of `OMBCREATE`, `OMBALTER`, `OMBRETRIEVE`, and `OMBDELETE` commands.

About Second Class Objects

A Second Class Object (SCO) represents a dependent object component. An SCO is always owned by another object, and can, in turn, own objects itself. For example, the First Class Object `MAPPING` contains Second Class Object `MAPPING_OPERATOR`, which contains `ATTRIBUTES`.

As a rule of thumb for those accessing Warehouse Builder through the graphic user interface, Second Class Objects can only be manipulated through a First Class Object. Similarly, users who access Warehouse Builder through OMB Plus can only manipulate Second Class Object definitions through a command against a First Class Object.

About Third and Fourth Class Objects

Third and Fourth Class objects are relative rankings of objects owned by other objects. These refer only to objects whose ownership spans several layers. For example, `INDEX_COLUMN` is a Second Class Object in the scenario where a `DIMENSTION_TABLE` (which is a First Class Object) owns `INDEX_COLUMN`. However, `INDEX_COLUMN` becomes a Third Class Object in the scenario where the First Class Object `CUBE_TABLE` owns the Second Class Object `INDEX`, which in turn owns `INDEX_COLUMN`.

For more information on these rankings, refer to "Object Ownership Tree" on page E-3.

Warehouse Builder Class Definition Objects

Warehouse Builder class definition objects can be used as arguments to the `OMBDESCRIBE CLASS_DEFINITION` and `OMBREDEFINE CLASS_DEFINITION` commands.

- ACTIVITY
- ACTIVITYPARAMETER
- ADVANCED_QUEUE
- ADVANCED_QUEUE OPERATOR
- AGGREGATOR OPERATOR
- BUSINESS_AREA
- BUSINESS_AREA_SHORTCUT
- CHECK_CONSTRAINT
- COLLECTION
- COLUMN
- CONNECTOR
- CUBE OPERATOR
- CUBE_TABLE
- DEDUPLICATOR OPERATOR
- DIMENSION OPERATOR
- DIMENSION_TABLE
- FUNCTION_BASE OPERATOR
- GATEWAY_MODULE
- GROUP
- HIERARCHY
- INDEX
- INDEX_COLUMN
- INPUT_PARAMETER OPERATOR
- JOINER OPERATOR
- KEY_LOOKUP OPERATOR
- LEVEL
- LEVEL_ATTRIBUTE
- LOCATION
- MAPPING
- MATERIALIZED_VIEW
- MATERIALIZED_VIEW OPERATOR
- MEASURE
- POSTMAPPING_PROCESS OPERATOR
- PREMAPPING_PROCESS OPERATOR
- PROCESS_DATA
- PROCESS_FLOW
- PROCESS_FLOW_MODULE
- PROCESS_FLOW_PACKAGE
- PROJECT
- RECORD
- REF_CURSOR_TYPE
- RUNTIME_REPOSITORY_CONNECTION
- SAP_MODULE
- SEQUENCE
- SEQUENCE OPERATOR
- SET_OPERATION OPERATOR
- SORTER OPERATOR
- SPLITTER OPERATOR

■ EXPRESSION OPERATOR	■ NAME_AND_ ADDRESS OPERATOR	■ SUBPROCESS
■ EXTERNAL_TABLE	■ OBJECT_TYPE	■ TABLE
■ EXTERNAL_TABLE OPERATOR	■ ORACLE_MODULE	■ TABLE OPERATOR
■ EXTERNAL_TABLE_ COLUMN	■ OUTPUT_ PARAMETER OPERATOR	■ TABLE_FUNCTION OPERATOR
■ FIELD	■ PACKAGE	■ TRANSFORMATION OPERATOR
■ FILTER OPERATOR	■ PARAMETER	■ TRANSITION
■ FLAT_FILE	■ PARAMETER_BASE OPERATOR	■ UNIQUE_KEY
■ FLAT_FILE OPERATOR	■ PIVOT OPERATOR	■ UNPIVOT OPERATOR
■ FLAT_FILE_ MODULE	■ PLSQL_RECORD_ TYPE	■ VARIABLES OPERATOR
■ FOREIGN_KEY	■ PLSQL_TABLE_ TYPE	■ VIEW
■ FUNCTION	■ PROCEDURE	■ VIEW OPERATOR

Possible datatypes include:

- STRING
- INTEGER
- BOOLEAN
- DATE

Object Ownership Tree

Use this tree to identify First Class and Second Class Objects, and to identify which objects own other objects. Use this information to understand user-defined properties and metadata change management (metadata snapshots), as well as to orient yourself to Oracle Warehouse Builder architecture.

The following First Class Objects do not own other objects:

- PROJECT
- ORACLE_MODULE
- FLATFILE_MODULE
- SAP_MODULE
- GATEWAY_MODULE
- COLLECTION
- PROCESS_FLOW_MODULE

- PROCESS_FLOW_PACKAGE
- CONNECTOR
- LOCATION
- RUNTIME_REPOSITORY_CONNECTION

The following First Class Objects own other objects:

- **ADVANCED_QUEUE**

  Contains the following Second Class Object:

  – OBJECT_TYPE
- **CUBE_TABLE**

  Contains the following Second Class Objects:

  – COLUMN
  – FOREIGN_KEY
  – UNIQUE_KEY
  – INDEX: This contains the Third Class Object INDEX_COLUMN.
  – PARTITION
  – PARTITION_KEY
- **DIMENSION TABLE**

  Contains the following Second Class Objects:

  – COLUMN
  – FOREIGN_KEY
  – HIERARCHY
  – INDEX
  – INDEX_COLUMN
  – LEVEL: This contains the Third Class Objects: FOREIGN_KEY, LEVEL_ATTRIBUTE, and UNIQUE_KEY.
  – UNIQUE_KEY
  – PARTITION
  – PARTITION_KEY
- **EXTERNAL TABLE**

  Contains the following Second Class Objects:

  – CHECK_CONSTRAINT
  – EXTERNAL_TABLE_COLUMN
  – FOREIGN_KEY
  – UNIQUE_KEY
- **FLAT_FILE_MODULE**

  Contains the following Second Class Objects:

  – FLAT_FILE

  - RECORD: This contains the Third Class Object FIELD.
  - FIELD

- **FUNCTION**

  Contains the following Second Class Objects:

  - PARAMETER

- **FUNCTION CATEGORY**

  Contains the following Second Class Objects:

  - PLSQL_RECORD_TYPE
  - PLSQL_TABLE_TYPE
  - REF_CURSOR_TYPE

- **MAPPING**

  Contains the following Second Class Objects:

  - OPERATOR
  - ADVANCED_QUEUE OPERATOR
  - AGGREGATOR OPERATOR
  - CUBE OPERATOR
  - DATA_ENTITY OPERATOR
  - DATA_ENTITY_KEYS OPERATOR
  - DIMENSION OPERATOR
  - EXPRESSION OPERATOR
  - EXTERNAL_TABLE OPERATOR
  - FILTER OPERATOR
  - FLAT_FILE OPERATOR
  - FUNCTION_BASE OPERATOR
  - INPUT_PARAMETER OPERATOR
  - JOINER OPERATOR
  - KEY_LOOKUP OPERATOR
  - MATERIALIZED_VIEW OPERATOR
  - NAME_AND_ADDRESS OPERATOR
  - PARAMETER_BASE OPERATOR
  - PIVOT OPERATOR
  - POSTMAPPING_PROCESS OPERATOR
  - PREMAPPING_PROCESS OPERATOR
  - SEQUENCE OPERATOR
  - SET_OPERATION OPERATOR
  - SORTER OPERATOR
  - SPLITTER OPERATOR

  - TABLE OPERATOR
  - TABLE_FUNCTION OPERATOR
  - TRANSFORMATION OPERATOR
  - UNPIVOT OPERATOR
  - VARIABLES OPERATOR
  - VIEW OPERATOR: This contains the Third Class Object Group which contains the Fourth Class Object Parameter.

- **MATERIALIZED_VIEW**

  Contains the following Second Class Objects:

  - CHECK_CONSTRAINT
  - COLUMN
  - FOREIGN_KEY
  - UNIQUE_KEY
  - INDEX: This contains the Third Class Object INDEX_COLUMN.
  - PARTITION
  - PARTITION_KEY

- **PROCEDURE**

  Contains the following Second Class Object:

  - PARAMETER

- **PROCESS_FLOW**

  Contains the following Second Class Objects:

  - SUBPROCESS
  - PROCESS_DATA
  - ACTIVITY: This contains the Third Class Object ACTIVITYPARAMETER.

- **SEQUENCE**

  Contains the following Second Class Object:

  - COLUMN

- **TABLE**

  Contains the following Second Class Objects:

  - CHECK_CONSTRAINT
  - COLUMN
  - FOREIGN_KEY
  - UNIQUE_KEY
  - INDEX: This contains the Third Class Object INDEX_COLUMN.
  - PARTITION
  - PARTITION_KEY

- **VIEW**

Contains the following Second Class Objects:

– CHECK_CONSTRAINT
– COLUMN
– FOREIGN_KEY
– UNIQUE_KEY
– INDEX: This contains the Third Class Object INDEX_COLUMN.

Glossary

ABAP

ABAP is a programming language for developing applications for the SAP R/3 system, a business application subsystem.

activity

In a **process flow**, activities represent units of work to be performed by the workflow engine. These units of work can involve components internal and external to **Oracle Warehouse Builder**. An example of an *external activity* is the Files Exists activity which checks for the existence of a **file** on a local or remote machine before continuing the process flow. An example of an *internal activity* is the FORK activity which specifies logic for launching concurrent activities.

additive

Describes a **fact/measure** that can be summarized through addition. An additive fact is the most common type of fact. Examples include Sales, Cost, and Profit. (Contrast with: **nonadditive**, **semi-additive**.)

advanced queue (AQ)

A database-integrated system containing message queues. AQs play a central role in enterprise data integration. Advanced Queues enable message management and communication required for application integration. **Oracle Warehouse Builder** supports Advanced Queues as data sources and targets for your warehouse design. In Warehouse Builder architecture, an AQ is a **First Class Object (FCO)**. (See also: **source**, **target**.)

agent

A software routine that runs on a remote computer and communicates with a central **server**. The server can delegate the execution of software that runs locally to an agent. An agent allows Warehouse Builder to call software on different machines.

aggregate data

Summarized data. For example, unit sales of a particular product could be aggregated by day, month, quarter and yearly sales. (See also: **aggregation**.)

aggregation

The process of consolidating data values into a single value. For example, sales data could be collected on a daily basis and then be aggregated to the week level, the week data could be aggregated to the month level, and so on. The data can then be referred to as **aggregate data**. *Aggregation* is synonymous with *summarization*, and *aggregate data* is synonymous with *summary data.*

analytic workspace

A single **file** containing objects that organize and store data in a form that **OLAP** Services can use. You determine the structure and contents of an analytic workspace by defining objects, examples of which are dimensions, variables, and programs. Once these definitions are in the analytic workspace dictionary, you can enter, change, or use the data with OLAP Services.

ancestor

A value at any **level** higher than a given value in a **hierarchy**. For example, in a Time **dimension**, the value 1999 might be the ancestor of the values Q1-99 and Jan-99.

API

See: **application program interface (API)**.

application program interface (API)

A set of public programmatic interfaces that consist of a language and message format to communicate with an **operating system** or other programmatic environment, such as databases, Web servers, JVMs, and so forth. These messages typically call functions and methods available for application development. You can access the Oracle Warehouse Builder Java API Reference through **Oracle MetaBase Plus (OMB Plus)**.

AQ

See: **advanced queue (AQ)**.

archive metadata

To backup **metadata** in **Oracle Warehouse Builder** by exporting it with version information using the **Metadata Loader (MDL)**. (See also: **export metadata**, **restore metadata**.)

attribute

Defined item of data that an object can store; a field represented by a **column** within an object (entity). In Oracle Warehouse Builder the term is used in two contexts:

1. A column in a **dimension** that characterizes elements of a single **level**.
2. An attribute represents a data level entry in an **operator** on a **mapping**.

attribute set

For tables and views in Warehouse Builder, an attribute set contains a chosen set of columns in the order you specify. Warehouse Builder generates a predefined **attribute** sets for tables and defined constraints. You can create the your own attribute sets when designing mappings. For exporting an attribute set to Oracle Discoverer, you can create a **bridge**-type attribute set.

AW

See: **analytic workspace**.

base language

The language assigned to the **Design Repository**. All **metadata** is stored in this language. The display language must be set to this language in order to create new objects in the **repository**. Currently the only supported base language is American English.

bind

The action that connects an **operator** in the Mapping Editor with an object in the **Design Repository**.

bitmap index

The purpose of an **index** is to provide pointers to the rows in a **table** that contain a given **key** value. A regular index achieves this by storing a list of `ROWID`s for each key corresponding to the rows with that key value. A bitmap index uses a bitmap instead of a list of `ROWID`s for each key value. Each bit in the bitmap corresponds to a possible `ROWID`, and if the bit is set, it means that the **row** with the corresponding `ROWID` contains the key value. A mapping function converts the bit position to an actual `ROWID`. Bitmap indexes are most effective for queries that contain multiple conditions in the `WHERE` clause.

bound name

The **physical name** of the object that is connected to an object **operator** in a **mapping**.

bound object

A **mapping** object **operator** in **Oracle Warehouse Builder** that is connected to an object defined in the **Design Repository**. (Contrast with: **unbound object**.)

breakpoint

A point of discontinuity, change, or cessation. In **Oracle Warehouse Builder**, this term is used to describe a point in a data flow that can be interrupted in order to ensure that the **operator** is functioning as expected.

bridge

A component in **Oracle Warehouse Builder** that enables you to import **metadata** from and export metadata to various tools, such as **CWM** compliant applications, Oracle Discoverer, Oracle Express, and Oracle9*i* **OLAP** server.

business name

The name of the **Oracle Warehouse Builder** object that is used in business applications. In previous releases of Warehouse Builder, this was known as the **logical name**. (Contrast with: **physical name**.)

cascade snapshot

A **metadata snapshot** in **Oracle Warehouse Builder** of a parent object (such as a **project**, folder, or **mapping**) that contains information about not only the **parent** object, but all its **child** objects as well. For example, if you take a cascade snapshot of a **database module**, Module A, which contains two tables, Table1 and Table2, and then change one of the tables in the repository, a comparison report will show Module A to have changed because one of its child objects has changed. (Contrast with: **no cascade snapshot**.)

child

1. In reference to dimensional hierarchies: A value at the **level** under a given value in a **hierarchy**. For example, in a Time **dimension**, the value Jan-99 might be the child of the value Q1-99. A value can be a child for more than one **parent** if the child value belongs to multiple hierarchies.
2. In reference to **Oracle Warehouse Builder metadata** objects: An object on the **navigation tree** that belongs to a higher-level object. For instance, a **table** is a child

object of a **database module**; a database module, in turn, is a child object of a **project**.

cleansing

The process of resolving inconsistencies and fixing the anomalies in **source** data, typically as part of the **ETL** process. (See also: **extraction, transformation, and loading (ETL)**.)

client

In **client**/**server** architecture, the front-end **database** application, which interacts with a user through the keyboard, display, and pointing device such as a mouse. The client portion has no data access responsibilities. It concentrates on requesting, processing, and presenting data managed by the server portion.

clipboard

The area in **Oracle Warehouse Builder client** where cut or copied objects are stored. Only one object can be stored at a time.

code generation

The **Oracle Warehouse Builder** process of taking **metadata** stored in the **Design Repository** and generating code that will create the designed **model** in the **target** system.

collection

A collection is a grouping mechanism you can use to group objects within a **project** in **Oracle Warehouse Builder**. Collections contain shortcuts to objects from the same project. If you want to **export metadata** to tools such as Oracle Discoverer, group the desired **metadata** in a collection. In Warehouse Builder architecture, a collection is a **First Class Object (FCO)**.

column

Vertical space in a **database** table that represents a particular domain of data. A column has a column name and a specific **datatype**. For example, in a **table** of employee information, all of the employees' dates of hire would constitute one column. (See also: **row**.)

commit

Make permanent changes to data (inserts, updates, deletes) in the **database**. Before changes are committed, both the old and new data exist so that changes can be stored or the data can be restored to its prior state. (See also: **rollback**.)

Common Warehouse Metamodel (CWM)

A **repository** standard used by Oracle data warehousing, decision support, and **OLAP** tools including **Oracle Warehouse Builder**. The CWM repository **schema** is an open standard repository that other products can share.

connection

Communication pathway between a user process and an Oracle **instance**.

connectivity agent

A generic connectivity agent is included with the Oracle Database **database server**. This **agent** can be used for low-end data integration solutions when the transfer of data is subject to the rules of specific ODBC or OLE DB drivers installed on the **client** system.

connector

An object on the **Oracle Warehouse Builder navigation tree** that is used to define **connection** information between locations. Only Oracle **database module** locations can have connectors defined. In Warehouse Builder architecture, a connector is a **First Class Object (FCO)**. (See also: **location**.)

console

The main window of the **Oracle Warehouse Builder** application. It contains a menu bar, launcher, and **navigation tree**.

constraint

Constraints provide a mechanism for ensuring that data conforms to guidelines specified by the **database** administrator. The most common types of constraints include unique constraints (ensuring that every value for a given **column** is unique), not null constraints, and foreign key constraints (which ensure that two keys in different tables share a **primary key**:**foreign key** relationship).

conventional view

See: **view**.

crosstab

A crosstab is a display layout that arranges items in a matrix of rows and columns. Items can appear on the row, column page edges. A crosstab can be used to display summary information and show how data varies across dimensions, such as sales by region by month. A crosstab is sometimes called a *matrix*.

cube

Fundamental structure for data in a multidimensional **schema**. A cube contains dimensions, hierarchies, levels and measures. In Warehouse Builder architecture, a cube is a **First Class Object (FCO)**. (See also: **dimension**, **hierarchy**, **level**, **measure**.)

CWM

See: **Common Warehouse Metamodel (CWM)**.

Data Definition Language (DDL)

Includes statements like `CREATE`/`ALTER TABLE`/`INDEX`, which define or change data structure.

data flow operator

In mappings, data flow operators alter or transform data as it flows from sources to targets. The Joiner operator is an example of a data flow operator because it joins multiple row sets from different sources with different cardinalities and produces a single output **row set**. (See also: **mapping**, **operator**, **source operator**, **target operator**.)

Data Manipulation Language (DML)

Includes statements like `INSERT`, `UPDATE`, and `DELETE`, which change data in tables.

data mart

A simple form of a **data warehouse** that is designed for a particular line of business, such as sales, marketing, or finance. Data marts are often built and controlled by a single department in an organization. Given their single-subject focus, data marts typically draw data from only a few sources. In a dependent data mart, the data can be derived from an enterprise-wide data warehouse. In an independent data mart, data can be collected directly from internal operational systems or external data.

data source

A **database**, application, data definition **source**, or **file** that contributes data.

data warehouse

A **relational database** that is designed for **query** and analysis rather than transaction processing. A data warehouse usually contains historical data that is derived from **transaction** data, but it can include data from other sources. It separates analysis workload from transaction workload and enables a business to consolidate data from several sources.

In addition to a relational database, a data warehouse environment often consists of an **ETL** solution, an **OLAP** engine, **client** analysis tools, and other applications that manage the process of gathering data and delivering it to business users. (See also: **extraction, transformation, and loading (ETL)**, **Online Analytical Processing (OLAP)**.)

database

Collection of data that is treated as a unit. The purpose of a database is to store and retrieve related information.

An Oracle database is a set of **operating system** files, treated as a unit, in which an Oracle database **server** stores a set of data dictionary tables and user tables. A database requires three types of files: database files, redo log files, and control files. (See also: **table**, **relational database**, **relational database management system (RDBMS)**.)

database link

An object stored in the local **database** that identifies a **remote database**, a communication path to the remote database, and optionally, a **username** and password for it. Once defined, a database link can be used to perform queries on tables in the remote database. Also called *DBlink*. In **SQL*Plus**, you can reference a database link in a `DESCRIBE` or `COPY` command.

database object

Something created and stored in a **database**. Tables, views, synonyms, indexes, sequences, clusters, and columns are all examples of database objects. (See also: **table**, **view**, **index**, **sequence**, **column**.)

database server

The computer that runs the ORACLE Server kernel and contains the **database**.

datatype

1. A standard form of data. The Oracle datatypes are `CHAR`, `NCHAR`, `VARCHAR2`, `NVARCHAR2`, `DATE`, `NUMBER`, `LONG`, `CLOB`, `NCLOB`, `RAW`, and `LONG RAW`; however, the Oracle **database server** recognizes and converts other standard datatypes.
2. A named set of fixed attributes that can be associated with an item as a property. Data typing provides a way to define the behavior of data. (See also: **attribute**.)

DDL

See: **Data Definition Language (DDL)**.

debug

To locate and correct malfunctioning elements or errors in a computer program code. In **Oracle Warehouse Builder**, this term is used to describe the process of looking for errors or problems in a data flow.

delimited flat file

A **flat file** whose fields are separated by delimiters. Contrast with a **fixed-length flat file**. (See also: **record**, **delimiter**.)

delimiter

A character or combination of characters used to separate one item or set of data from another. For example, in comma delimited records, a comma is used to separate each field of data. There are also fixed-length records, where each field's beginning and end is specified by position rather than any character acting as a delimiter. (See also: **flat file**, **record**.)

denormalize

The process of allowing redundancy in a **table** so that it can remain flat. (Contrast with: **normalize**.)

deployment

The process of creating the physical **target** system from a previously designed logical **model**.

Deployment Manager

Oracle Warehouse Builder's comprehensive **deployment** console that enables you to view and manage all aspects of deployment including configuration and **validation**. It also enables you to view the deployment history of an object to determine how you want to deploy the object.

derived fact (or measure)

A **fact/measure** that is generated from existing data using a mathematical operation or a data **transformation**. Examples include averages, totals, percentages, and differences.

Design Repository

Installed in an Oracle database, the Design Repository stores the **metadata** definitions for all of the objects used in **Oracle Warehouse Builder**. This is where all of the design information is stored for the **target** systems you are creating. You can access metadata stored here using the client user interface, or through **Oracle MetaBase Plus (OMB Plus)**. This repository is created using the Repository Assistant. (Contrast with: **Runtime Repository**.)

dimension

A structure, often composed of one or more hierarchies, that categorizes data. Several distinct dimensions, combined with measures, enable end users to answer business questions. Commonly used dimensions are Customer, Product, and Time. In Oracle9*i*, a dimension is a **database object** that defines hierarchical (**parent**/**child**) relationships between pairs of **column** sets. In Warehouse Builder architecture, a dimension is a **First Class Object (FCO)**.

dimension value

One element in the list that makes up a **dimension**. For example, a computer company might have dimension values in the Product dimension called LAPPC and DESKPC. Values in the Geography dimension might include Boston and Paris. Values in the Time dimension might include MAY96 and JAN97.

dimensional object

Dimensional objects are similar to relational objects, but 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. (Contrast with: **relational object**.)

display set

In mappings, a display set is a graphical representation of a subset of **operator** attributes. Use display sets to limit the number of attributes visible in an operator and therefore simplify the display of complex mappings.

DML

See: **Data Manipulation Language (DML)**.

drill

To navigate from one item to a set of related items. Drilling typically involves navigating up and down through the levels in a **hierarchy**. When selecting data, you can expand or collapse a hierarchy by drilling down or up in it, respectively. (See also: **level**, **drill down**, **drill up**.)

drill down

To expand the view to include **child** values that are associated with **parent** values in the **hierarchy**. (See also: **drill**, **drill up**, **level**.)

drill up

To collapse the list of descendant values that are associated with a **parent** value in the **hierarchy**. (See also: **drill**, **drill down**, **level**.)

editor

A window in **Oracle Warehouse Builder** used to define or edit objects and their relationships to each other.

element

An object or process. For example, a **dimension** is an object, a **mapping** is a process, and both are elements.

endian

Refers to *byte order*, the numeric arrangement of bytes in a *word*, which is the basic unit of storage in a computer (words are 8, 16, 32 and 64 bits long). *Big-endian* is the normal order and the way humans deal with arithmetic: the most significant byte or digits are placed leftmost in the structure. Some CPUs deal with words in *little-endian* order, which is the reverse and places the least significant digits on the left. Since numbers are calculated by the CPU starting with the least significant digits, little-endian numbers are already set up in the required processing order.

enqueue

Enqueues are shared memory structures that serialize access to **database** resources and are associated with a session or **transaction**. (See also: **lock**.)

ETL

See: **extraction, transformation, and loading (ETL)**.

execution

The process of running deployed mappings and process flows in **Oracle Warehouse Builder**. (See also: **deployment**.)

export metadata

The **extraction** of a copy of **metadata** (that is, not physical files) from a **repository** using the Export utility. You can then use the Import utility to import the data into a repository. (See also: **Metadata Loader (MDL)**, **archive metadata**.)

expression

A formula, such as `SALARY + COMMISSION`, used to calculate a new value from existing values. An expression can be made up of **column** names, functions, operators, and constants. Formulas are found in commands or **SQL** statements.

eXtensible Markup Language (XML)

An open standard for describing data developed by the W3C using a subset of the SGML syntax and designed for Internet use. Version 1.0 is the current standard, having been published as a W3C Recommendation in February 1998.

external table

An external table is a read-only **table** 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. In Warehouse Builder architecture, an external table is a **First Class Object (FCO)**.

extraction

The process of taking data out of a **source** as part of an initial phase of **ETL**. (See also: **extraction, transformation, and loading (ETL)**.)

extraction, transformation, and loading (ETL)

ETL refers to the methods involved in accessing and manipulating **source** data and loading it into a **data warehouse** or other type of **target**. The order in which these processes are performed varies. *ETT* (**extraction**, **transformation**, transportation) and *ETM* (extraction, transformation, move) are sometimes used instead of *ETL*.

fact table

A **table** in a **star schema** that contains **fact/measure**s. A fact table typically has two types of **column**s: those that contain facts and those that are **foreign key**s to **dimension** tables. The **primary key** of a fact table is usually a composite key that is made up of all of its foreign keys.

A fact table can contain either detail level facts or facts that have been aggregated (fact tables that contain aggregated facts are often instead called aggregate tables). A fact table usually contains facts with the same level of **aggregation**.

In **Oracle Warehouse Builder**, facts and fact tables are referred to as *cubes*. (See: **cube**.)

fact/measure

Data, usually numeric and **additive**, that can be examined and analyzed. Values for facts or measures are usually not known in advance; they are observed and stored. Examples include Sales, Cost, and Profit. *Fact* and *measure* are synonymous; *fact* is more commonly used with relational environments; *measure* is more commonly used with multi-dimensional environments. (See also: **dimension**, **attribute**.)

In **Oracle Warehouse Builder**, facts and fact tables are referred to as *cubes*. (See also: **derived fact (or measure)**, **cube**.)

FCO

See: **First Class Object (FCO)**.

file

A collection of data treated as a unit, such as a list, document, index, note, set of procedures, and so on. Generally used to refer to data stored on magnetic tapes or disks. In Warehouse Builder architecture, a file is a **First Class Object (FCO)**.

file-to-table mapping

Maps data from flat files to tables in the **data warehouse**. (See also: **flat file**, **table**.)

filter

To select data. Filters use patterns (masks) against which all data are compared and only matching data are "passed through," hence the concept of a filter. For example, e-mail clients and servers can filter out important messages and alert the user or look for text patterns that appear to be spam and delete it. In **SQL**, the `WHERE` clause contains the filter conditions on the data that is being queried.

First Class Object (FCO)

In **Oracle Warehouse Builder** architecture, a First Class Object (FCO) represents a component in the metadata repository that can be manipulated through the Warehouse Builder interface. First Class Objects often, but not always, own other objects. For example, a `TABLE` is a First Class Object that may own the following second class objects: `TABLE_COLUMN`, `UNIQUE_KEY`, `FOREIGN_KEY`, and `CHECK_CONSTRAINT`. Other examples of FCOs include `EXTERNAL_TABLE`, `CONNECTOR`, and `MAPPING`.

As a rule of thumb for those accessing Warehouse Builder through the graphic user interface, first class objects generally appear on the **navigation tree**. Similarly, users who access Warehouse Builder through **Oracle MetaBase Plus (OMB Plus)** can generalize FCOs as objects of `OMBCREATE`, `OMBALTER`, `OMBRETRIEVE`, and `OMBDELETE` commands.

(See also: **Second Class Object (SCO)**, **Third Class Object**.)

fixed-length field

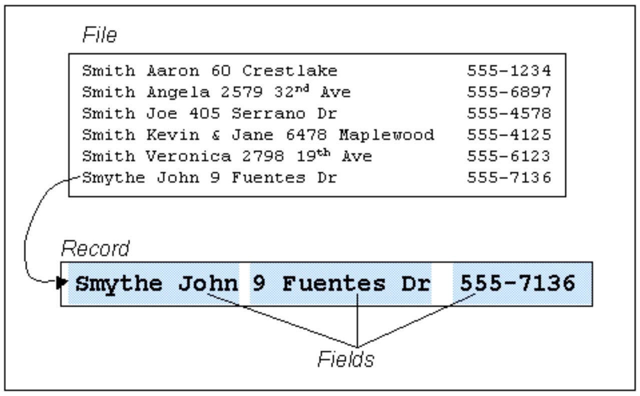
A field whose beginning and end is specified by position rather than any character acting as a **delimiter**. Fixed-length fields conform to specified field sizes; for example, a 25-byte name field takes up 25 bytes in each **record**. (See also: **fixed-length flat file**, **flat file**.)

fixed-length flat file

A **flat file** whose records have fixed-length fields. Contrast with a **delimited flat file**, whose records contain variable-length fields separated by a **delimiter**. (See also: **record**, **fixed-length field**.)

flat file

A relatively simple **database** system in which each database is contained in a single **table**. A flat file is not related to, or does not contain any linkages to, another **file**. It is generally used for standalone lists. Flat files can be related, but only if the applications are programmed to do so. In contrast, **relational database** systems can use multiple tables to store information, and each table can have a different **record** format. When files must be related (such as customers to orders, vendors to purchases), a relational database manager is used, not a flat file manager. (See also: **delimited flat file**, **fixed-length flat file**, **logical record**.)


File
Smith Aaron 60 Crestlake 555-1234
Smith Angela 2579 32nd Ave 555-6897
Smith Joe 405 Serrano Dr 555-4578
Smith Kevin & Jane 6478 Maplewood 555-4125
Smith Veronica 2798 19th Ave 555-6123
Smythe John 9 Fuentes Dr 555-7136
Record
Smythe John 9 Fuentes Dr 555-7136
Fields

foreign key

An **integrity constraint** that requires each value in a **column** or set of columns to match a value in a the unique or **primary key** of a related **table**. Foreign key integrity constraints also define referential integrity actions that dictate what Oracle should do with dependent data if the data it references is altered.

Fourth Class Object

In **Oracle Warehouse Builder** architecture, Third and Fourth Class objects are relative rankings of objects owned by other objects. These refer only to objects whose ownership spans several layers. For example, `INDEX_COLUMN` is a Second Class Object in the scenario where a `DIMENSTION_TABLE` (which is a First Class Object) owns `INDEX_COLUMN`. However, `INDEX_COLUMN` becomes a Third Class Object in the scenario where the First Class Object `CUBE_TABLE` owns the Second Class Object `INDEX`, which in turn owns `INDEX_COLUMN`.

(See also: **First Class Object (FCO)**, **Second Class Object (SCO)**.)

full snapshot

A **metadata snapshot** that contains enough information about the object to use not only for comparison purposes, but for restore purposes as well. A Full snapshot consumes more space than a **signature snapshot**, but you can only **restore metadata** from full snapshots. Full snapshots can be converted to signature snapshots, but the reverse is not true.

generation

See: **code generation**.

group

In mappings, attributes in each **operator** are designated into groups. Groups indicate the directional nature of the attributes. Therefore, an input group contains attributes that are input into the operator. An output group contains operator output attributes. (See also: **attribute**.)

hierarchy

Hierarchies are **logical structures** that use ordered levels as a means of organizing data. A hierarchy can be used to define data **aggregation**. For example, in a Time **dimension**, a hierarchy might be used to aggregate data from the Month **level** to the Quarter level to the Year level. A hierarchy can also be used to define a navigational **drill** path, regardless of whether the levels in the hierarchy represent aggregated totals.

HTML

See: **Hypertext Markup Language (HTML)**.

Hypertext Markup Language (HTML)

The markup language used to create the files sent to Web browsers and that serves as the basis of the World Wide Web. The next version of HTML is called xHTML and is an **XML** application.

import metadata

To bring previously exported **metadata** into an **Oracle Warehouse Builder Design Repository** using the **Metadata Loader (MDL)** utility. Imported metadata can be merged with or can overwrite existing objects. MDL has its own **file** format; only `.MDL` files can be exported and imported using the MDL utility. (See also: **restore metadata**, **export metadata**.)

index

Indexes are optional structures associated with tables and clusters. You can create indexes on one or more **column**s of a **table** to speed **SQL** statement execution on that table. (See also: **bitmap index**.)

inner join

This is the default type of **join**. It produces a resulting **row** if there is a matching condition. For example, matching shipments with receipts would produce only those shipments that have been received. (Contrast with: **outer join**.)

input role

In the Name and Address operator, input roles indicate what kind of name or address information resides in a line of data being extracted from the source. Input roles can be *non-discrete* (line oriented) for free-form data, or *discrete* (such as first name, primary address, or city) for specific input attributes. Whenever possible, choose discrete input roles (such as 'Person'), rather than non-discrete roles (such as 'Line1'). Discrete roles give the Name and Address operator more information about the content of the source attribute. (See also: **output component**.)

instance

A system global area (SGA) and the Oracle background processes constitute an Oracle instance. Every time a **database** is started, a system global area is allocated and Oracle background processes are started. The SGA is de-allocated when the instance shuts down.

integrator

Software that works with **Oracle Warehouse Builder** to facilitate definition, design, and **extraction** of **source** data. Examples of integrators include the Oracle Applications Integrator and the SAP Integrator.

integrity constraint

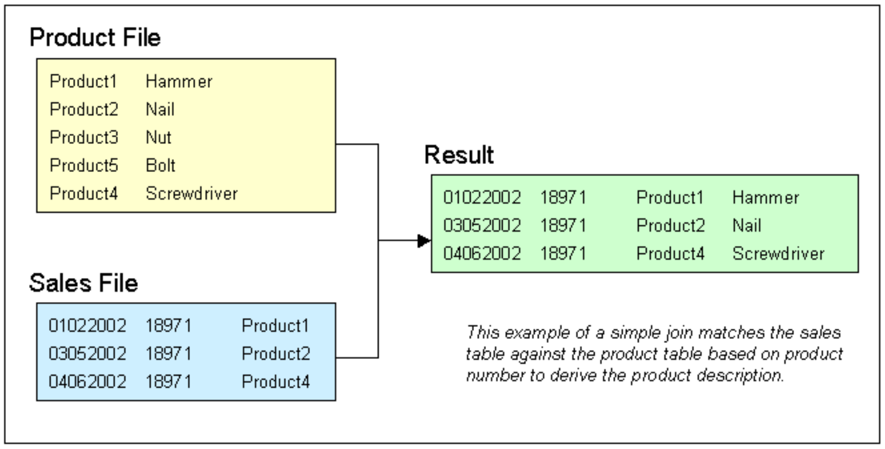
Declarative method of defining a rule for a **column** of a **table**. Integrity constraints enforce the business rules associated with a **database** and prevent the entry of invalid information into tables.

IP address

Used to identify a node on a network. Each computer on the network is assigned a unique IP address, which is made up of the network ID, and a unique host ID. This address is typically represented in dotted-decimal notation, with the decimal value of each octet separated by a period, for example 144.45.9.22.

join

In **relational database** management, a **query** that selects data from more than one **table**. A join matches one table (**file**) against another based on some condition, creating a third table with data from the matching tables. For example, a customer table can be joined with an order table creating a table for all customers who purchased a particular product. A join is characterized by multiple tables in the `FROM` clause. Oracle pairs the rows from these tables using the condition specified in the `WHERE` clause and returns the resulting rows. This condition is called the join condition and usually compares columns of all the joined tables. (See also: **inner join**, **outer join**.)



Product File

Product1	Hammer
Product2	Nail
Product3	Nut
Product5	Bolt
Product4	Screwdriver

Sales File

01022002	18971	Product1
03052002	18971	Product2
04062002	18971	Product4

Result

01022002	18971	Product1	Hammer
03052002	18971	Product2	Nail
04062002	18971	Product4	Screwdriver

This example of a simple join matches the sales table against the product table based on product number to derive the product description.

key

A **column** or set of columns included in the definition of certain types of integrity constraints. Keys describe the relationships between the different tables and columns of a **relational database**. (See also: **integrity constraint**, **foreign key**, **primary key**.)

launcher

The panel in the **Oracle Warehouse Builder** console window that contains buttons to control the active environment. These environments include the Project environment, Administration environment, and Transformation Library environment.

level

A position in a **hierarchy**. For example, a Time **dimension** might have a hierarchy that represents data at the Month, Quarter, and Year levels.

level value table

A **database table** that stores the values or data for the levels you created as part of your dimensions and **hierarchy** definitions. (See also: **dimension**, **level**.)

local enqueues

Local enqueues are **synchronization** mechanisms utilized to coordinate concurrent access to shared data structures in cluster **database**s. Local enqueues are called **local locks** in single **instance** Oracle. (See also: **enqueue**.)

local locks

In single **instance** Oracle, local locks are **synchronization** mechanisms utilized to coordinate concurrent access to shared data structures. In a cluster **database**, local locks are called **local enqueues**.

locale

A collection of information regarding the linguistic and cultural preferences from a particular region. Typically, a locale consists of language, territory, character set, linguistic, and calendar information defined in National Language Support or Globalization Support data files.

location

An object on the **Oracle Warehouse Builder navigation tree** that represents the physical locations to which **target** systems will be deployed. In Warehouse Builder architecture, a location is a **First Class Object (FCO)**. (See also: **connector**, **deployment**.)

lock

Locks are used in a multiuser environment where one user is given control to modify an object while the others have read only access. Maintains integrity of objects while sharing projects. A lock is kept by a user until the user commits or rolls back, and closes any open editors and property sheets. (See also: **project**, **commit**, **rollback**.)

logical name

See: **business name**.

logical record

In **flat file**s, records can be organized logically with one logical **record** corresponding to multiple physical records.

logical structures

Logical structures of an Oracle **database** include tablespaces, **schema** objects, data blocks, extents, and segments. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures. (Contrast with: **physical structures**.)

lookup

A data search performed within a predefined **table** of values (such as an array or matrix) or within a data **file**.

mapping

The definition of the relationship and data flow between **source** and **target** objects. In Warehouse Builder architecture, a mapping is a **First Class Object (FCO)**.

master-detail

A multiple-record-type **flat file** where the master records have corresponding detail records. For example, a simple master-detail flat file can have two **record** types: master records with Department information, and detail records for each master record, containing Employee information for each employee in that department.

materialized view

A pre-computed table comprising **aggregate data** and/or joined data from cubes and possibly **dimension** tables. Also known as a summary or aggregate table. A materialized view provides indirect access to **table** data by storing the results of a **query** in a separate **schema** object. Like an **index**, a materialized view consumes storage space, must be refreshed when the data in its master tables changes, improves performance of **SQL** execution, and is transparent to SQL applications and users. Unlike an index, a materialized view can be accessed directly, using `SELECT`, `INSERT`, `UPDATE`, and `DELETE` statements. In Warehouse Builder architecture, a materialized view is a **First Class Object (FCO)**. (Contrast with: **conventional view**.)

measure

Data, usually numeric and **additive**, that can be examined and analyzed. Values for facts or measures are usually not known in advance; they are observed and stored. Examples include Sales, Cost, and Profit. Fact and measure are synonymous; fact is more commonly used with relational environments; measure is more commonly used with multi-dimensional environments.

metadata

Data that describes data and other structures, such as objects, business rules, and processes. For example, the **schema** design of a **data warehouse** is typically stored in a **repository** as metadata, which is used to generate scripts that build and populate the data warehouse. A repository contains metadata.

Examples include: for data, the definition of a **source** to **target transformation** that is used to generate and populate the data warehouse; for information, definitions of tables, columns, and associations that are stored inside a relational modeling tool; for business rules, discount by 10 percent after selling 1,000 items.

Metadata Loader (MDL)

The **Oracle Warehouse Builder** utility that enables you to backup your Design Repository. You can **export metadata** from a **Design Repository** into an `.MDL` text **file**, and you can **import metadata** from an `.MDL` file into a **repository**. If desired, you can export metadata with version information. When importing, you can overwrite existing **metadata** objects, or merge using other options.

metadata snapshot

A method of capturing the state of your **Design Repository** (or selected objects in the repository) for **metadata** history management. Comparison reports can be generated between a metadata snapshot and the current **repository**, or between two metadata snapshots.

A **full snapshot** enables you to not only generate comparison reports, but to restore objects from the snapshot into the current repository, whereas a **signature snapshot** can only be used for comparison purposes. A **cascade snapshot** captures information about a selected object and all of its **child** objects (if any), where as a **no cascade snapshot** only captures information about the selected object. In Warehouse Builder architecture, a snapshot is a **First Class Object (FCO)**.

model

An object that represents something to be made. A representative style, plan, or design. Metadata that defines the structure of the **data warehouse**. (See also: **metadata**.)

module

Modules are storage objects within projects that help to organize **source** and **target** objects. Visible on the **Oracle Warehouse Builder** main **project navigation tree**, modules are used as **metadata** containers for databases, files, applications, and process flows. Source modules contain the metadata from existing source systems from which you are pulling. Target modules contain the metadata you are designing.

natural key

A **key** that is native to the data and is used to identify each **row**. For example, the state codes of CA and DC can be used as natural keys.

navigation tree

The main console workspace where **Oracle Warehouse Builder** objects are organized into projects. This is similar to a **file** system with all of the objects organized into expandable folders. (See also: **project**, **First Class Object (FCO)**.)

no cascade snapshot

A **metadata snapshot** that only captures information about the selected object, and not about its **child** objects. For example, if you take a no cascade snapshot of a **database module**, Module A, which owns two tables, Table1 and Table2, and then change one of the tables in the repository, a comparison report does not show Module A to have changed. (Contrast with: **cascade snapshot**.)

nonadditive

Describes a **fact/measure** that cannot be summarized through addition. An example is average. (Contrast with **additive**, **semi-additive**.)

normalize

In a **relational database**, the process of removing redundancy in data by separating the data into multiple **tables**. (Contrast with **denormalize**.)

OLAP

See: **Online Analytical Processing (OLAP)**. Contrast with: **Online Transactional Processing (OLTP)**.

OLTP

See: **Online Transactional Processing (OLTP)**. Contrast with **Online Analytical Processing (OLAP)**.

Online Analytical Processing (OLAP)

OLAP functionality is characterized by dynamic, multi-dimensional analysis of historical data, which supports activities such as the following:

- Calculating across dimensions and through hierarchies
- Analyzing trends

- Drilling up and down through hierarchies
- Rotating to change the dimensional orientation

OLAP tools can run against a multidimensional **database** or interact directly with a **relational database**. Contrast with **Online Transactional Processing (OLTP)**. (See also: **drill**, **hierarchy**.)

Online Transactional Processing (OLTP)

OLTP systems are application systems that are characterized by updates to the **database**. Examples of OLTP systems include e-business systems and ERP applications (such as Oracle Applications and SAP R/3). More specialized examples include telephone call and billing systems, credit card transactions, and airline reservation systems. OLTP systems are also known as *transaction systems*. Contrast with **Online Analytical Processing (OLAP)**. (See also: **transaction**.)

Open Database Connectivity (ODBC)

Oracle ODBC Driver provides a standard interface that allows one application to access many different data sources. The application's source code does not have to be recompiled for each data **source**. A **database** driver links the application to a specific data source. A database driver is a dynamic link library that an application can invoke on demand to gain access to a particular data source. Therefore, the application can access any data source for which a database driver exists.

operating system

The system software that manages a computer's resources, performing basic tasks such as allocating memory and allowing computer components to communicate. **Oracle Warehouse Builder** is available on these operating systems:

- Windows: NT, 2000, and XP
- UNIX: Solaris, Linux, and HP-UX

operator

The operator is the basic design element for a **mapping**. (See also: **data flow operator**, **source operator**, **target operator**.)

Oracle Enterprise Manager (OEM)

A separate Oracle product that combines a graphical console, agents, common services, and tools to provide an integrated and comprehensive systems management platform for managing Oracle products.

Oracle MetaBase (OMB)

The Oracle Warehouse Builder repository, including its services, such as version control, extensibility, multiuser locking, copy/paste.

Oracle MetaBase Plus (OMB Plus)

A tool that provides access to Oracle Warehouse Builder scripting capabilities.

Oracle Warehouse Builder Name and Address

Licensed separately from Oracle Warehouse Builder , this product consists of third-party data files that provide the name and address data to be used with the Oracle Warehouse Builder Name and Address **operator** to cleanse user data. These files are updated quarterly and updates are available on MetaLink.

Oracle Server

The **relational database management system (RDBMS)** sold by Oracle Corporation. Components of Oracle Server include the kernel and various utilities for use by DBAs and **database** users.

Oracle Universal Installer (OUI)

A graphical **user interface (UI)** that facilitates the installation of the Oracle **database** software and its related components.

Oracle Warehouse Builder

Oracle Warehouse Builder is a business intelligence tool that provides an integrated solution for designing and deploying enterprise data warehouses, data marts, and business intelligence applications. It solves the complex problem of data integration between various dispersed data sources and targets. In addition, Warehouse Builder provides all the necessary functionality to maintain the life cycle of the system you develop. (See also: **data warehouse**, **data mart**, **source**, **target**.)

It is a comprehensive toolset designed for practitioners who move and transform data, develop and implement data warehouses, perform **metadata** management, or create and manage Oracle **database**s and metadata. In addition to its graphical **user interface (UI)**, Warehouse Builder provides an **API** in the form of **Oracle MetaBase Plus (OMB Plus)**, where all the Warehouse Builder functionality can be accessed using the OMB Scripting Language.

ORACLE_HOME

Corresponds to the environment in which Oracle products run. This environment includes the location of installed product files, the `PATH` variable pointing to the products' binary files, registry entries, net service names, and program groups.

If you install an OFA-compliant **database**, using Oracle Universal Installer defaults, Oracle home (known as `\ORACLE_HOME` in this guide) is located beneath `X:\ORACLE_BASE`. It contains subdirectories for Oracle software executables and network files. (See also: **service name**.)

outer join

A **join** condition using the outer join **operator** (+) with one or more columns of one of the tables. Oracle returns all rows that meet the join condition. Oracle also returns all rows from the **table** without the outer join operator for which there are no matching rows in the table with the outer join operator. For example, matching shipments with receipts would produce not only those shipments that have been received, but would create a **record** for every shipment whether or not it was received. The data for received items would be attached to the shipments, and empty, or null, fields would be attached to shipments without receipts. (Contrast with: **inner join**.)

output component

In the Name and Address operator, the output component indicates which component of a name or address an attribute constitutes. Each output component represents a discrete name or address entity, such as a title, a standardized first name, a street number, a street name, or a street type. Every output attribute from the Name and Address operator must have an output component that identifies its role in the name or address. (See also: **input role**.)

dual address

In the Name and Address operator, 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, one address line becomes the *normal address* and one becomes the *dual address*. A sample dual address is:

PO Box 2589
4439 Mormon Coulee Rd
La Crosse WI 54601-8231

package

A method of encapsulating and storing related procedures, functions, and other package constructs together as a unit in the **database**. While packages provide the database administrator or application developer organizational benefits, they also offer increased functionality and database performance. (See also: **PL/SQL**.)

parent

1. A value at the **level** higher than a given value in a **hierarchy**. For example, in a Time **dimension**, the value Q1-99 might be the parent of the value Jan-99. (See also: **child**.)
2. In reference to **Oracle Warehouse Builder metadata** objects: An object on the **navigation tree** that owns to a lower-level objects. For instance, a **project** is a parent object of the modules it owns; a **database** module is, in turn, a parent object of the **table** objects it owns.

partition

A unique, non-overlapping directory naming context that is stored on one directory **server**.

partition exchange loading (PEL)

A runtime performance feature in **Oracle Warehouse Builder** where data partitioning is used improve performance when loading or purging data in a **target** system. (See also: **partition**.)

payload

A protocol structure encapsulating a set of logical ICE operations delivered at discrete intervals. A payload is a single instance of an **XML** document formatted according to the protocol definitions contained in this specification.

physical instance

A collection of related **database** objects that, when deployed, becomes a **schema**. Objects include tables, views, and other objects.

physical name

Unique name of a **repository** object. Conforms to the basic syntax rules for **schema** objects, as defined in the Oracle Database **SQL** Reference. Used in the generation of **DDL** scripts. Users can create a physical name of a new object or modify the physical name of an existing object.

physical structures

Physical **database** structures of an Oracle database include data files, redo log files, and control files. (See also: **logical structures**.)

pivot

In a **mapping**, the pivot **operator** enables you to transform a single **row** of attributes into multiple rows. Use this operator in a mapping when you want to transform data that is contained across attributes instead of rows. (See also: **unpivot**.)

PL/SQL

The 3GL Oracle procedural language extension of **SQL**. PL/SQL enables you to mix SQL statements with procedural constructs. PL/SQL combines the ease and flexibility of SQL with the procedural functionality of a structured programming language, such as `IF...THEN`, `WHILE`, and `LOOP`. Even when PL/SQL is not stored in the **database**, applications can send blocks of PL/SQL to the database rather than individual SQL statements, thereby reducing network traffic. With PL/SQL, you can define and execute PL/SQL program units such as procedures, functions, and packages. PL/SQL is interpreted and parsed at runtime; it does not need to be compiled. (See also: **package**, **Structured Query Language (SQL)**, **SQL*Plus**.)

portlet

A portlet are components of information from other web sites or applications that you can use to build a customized web page. Warehouse Builder Browser, for example, consists of portlets that you can add to customize a web page for navigate and report on **metadata** in Warehouse Builder.

primary key

The **column** or set of columns included in the definition of a table's `PRIMARY KEY` constraint. A primary key's values uniquely identify the **row**s in a **table**. Only one primary key can be defined for each table. (See also: **foreign key**.)

privilege

A right to execute a particular type of **Structured Query Language (SQL)** statement or to access another user's object.

process flow

A process flow describes dependencies between Warehouse Builder mappings and external activities such as email, FTP, and **operating system** commands. It provides a visual representation of a sequence of actions in a flow. Basic design elements for a process flow include the **activity** and the **transition**. In Warehouse Builder architecture, a process flow is a **First Class Object (FCO)**.

project

Oracle Warehouse Builder projects are the largest storage objects within the Warehouse Builder **Design Repository** and help users organize their work. Projects store and organize **metadata** definitions required to perform **extraction, transformation, and loading (ETL)** for a specific set of sources and targets. These definitions include data sources, target warehouse objects, mappings from source to targets, transformation operations, and configuration parameters. These definitions are organized into folders within the project. Typically a project contains related metadata. Creating multiple projects enables you to create your design in an organized manner.

query

A **Structured Query Language (SQL)** `SELECT` statement that retrieves data, in any combination, **expression**, or order. Queries are read-only operations; they do not change any data; they only retrieve data. Queries are often considered to be **DML** statements.

query results

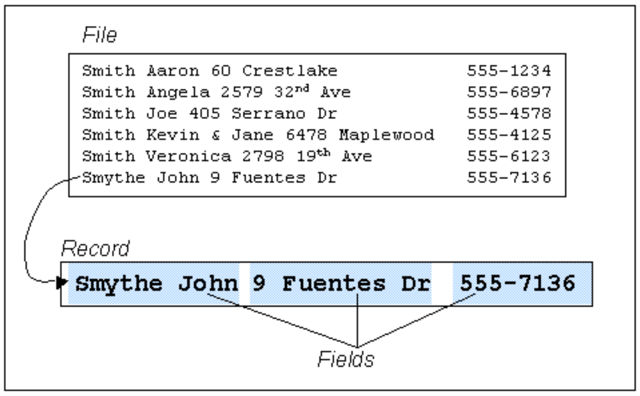
The data retrieved by a **query**.

reconcile

In mappings, every **operator** you define can have a corresponding definition in the **Oracle Warehouse Builder** repository. When you reconcile operators, you ensure that the operator matches the repository object it represents. You can reconcile inbound (into the **mapping**) or outbound (out from the mapping). Inbound reconciliation updates the operator with the definition of a specified repository object. Outbound reconciliation updates a selected **repository** object to reflect changes you made to the operator in a mapping.

record

In **flat file database** management systems, a record is a complete set of information. Records are composed of *fields*, each of which contains one item of information. A set of records constitutes a **file**. For example, a personnel file might contain records that have three fields: a name field, an address field, and a phone number field.



```
Smith Aaron 60 Crestlake            555-1234
Smith Angela 2579 32nd Ave          555-6897
Smith Joe 405 Serrano Dr            555-4578
Smith Kevin & Jane 6478 Maplewood   555-4125
Smith Veronica 2798 19th Ave        555-6123
Smythe John 9 Fuentes Dr            555-7136

Smythe John 9 Fuentes Dr  555-7136
```

relational database

A structured collection of data that stores data in the form of related tables consisting of one or more rows, each containing the same set of columns. Data in multiple tables is linked because the relationships between the tables are defined. The link is based on one or more fields common to both tables. Relational databases are powerful because they require few assumptions about how data is related or how it will be extracted from the **database**. As a result, the same database can be viewed in many different ways. An important feature of relational systems is that a single database can be spread across several tables. This differs from **flat file** databases, in which each database is self-contained in a single table.

relational database management system (RDBMS)

A computer program designed to store and retrieve shared data. In a relational system, data is stored in tables consisting of one or more rows, each containing the same set of columns. Oracle is a relational database management system. Other types of **database** systems are called hierarchical or network database systems.

relational object

A relational object, like a **relational database**, relies on tables and table-derived objects to store and link all of its data. The relational objects you define in **Oracle Warehouse Builder** 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. (Contrast with: **dimensional object**.)

remote database

A **database** other than your default database, which may reside on a remote computer; in particular, one that you reference in the CONNECT, COPY, and **SQL*Plus** commands.

repository

A **metadata** store; the repository environment contains the complete set of a business's metadata.

restore metadata

The act of importing archived **metadata** into the current **Design Repository** using the **Metadata Loader (MDL)** utility. Restoring metadata creates an additional **project** on the **navigation tree** and will not overwrite existing objects. (See also: **archive metadata**, **import metadata**.)

RDBMS

See: **relational database management system (RDBMS)**.

role

A named group of related privileges that is granted to users or other roles.

rollback

To discard pending changes made to the data in the current **transaction** using the **SQL** ROLLBACK command. (See also: **commit**.)

row

The basic unit for the processing of data in any **mapping**. A row is a set of attributes or values pertaining to one entity or **record** in a **table**. It is a collection of **column** information corresponding to a single record. A row has a structure and is defined by attributes, where each **attribute** is given a name and **datatype**, and length, scale, and precision.

row based

Operating mode option. Row based mode allows for the maximum amount of runtime auditing. A **mapping** that runs in **row**-based mode will process data from a **source** record by **record**. (Contrast with: **set based**.)

row set

A row set consists of zero or more rows of structured data brought into or emerging from an **operator** in a **mapping**. A mapping defines how row sets are extracted from a **source**, transformed, and loaded into a **target** using operators. The number of rows in a row set is called the *cardinality* of that row set.

runtime environment

Refers to post-**deployment** environments. This is the location where objects are deployed. The **Runtime Repository** stores all data about deployments and runtime environments.

Runtime Platform Service (RTPS)

The server-side component of **Oracle Warehouse Builder** software that provides **execution** and **deployment** services. In order for you to be able to run these services, the Runtime Platform Service must be active. The Runtime Platform Service manages the execution of mappings and process flows from within Warehouse Builder and ensures that all execution and deployment audit data is stored in a **Runtime**

Repository. For remote executions, it connects to the **Oracle Enterprise Manager (OEM)** Management Server. The Runtime Platform Service is invoked through a database job that is automatically started when the **database** is started and shut down when the database is shut down.

Runtime Repository

The **Oracle Warehouse Builder** repository that stores all of the **deployment** and **execution** data. This **repository** is created using the Runtime Assistant. (Contrast with: **Design Repository**.)

schema

Collection of related **database** objects, including **logical structures** such as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links. A schema is owned by a database user and has the same name as that user; relational schemas are grouped by database user ID. (See also: **snowflake schema**, **star schema**.)

SCO

See: **Second Class Object (SCO)**.

Second Class Object (SCO)

In **Oracle Warehouse Builder** architecture, a Second Class Object (SCO) represents a dependent object component. An SCO is always owned by another object, and can, in turn, own objects itself. For example, the First Class Object `MAPPING` contains Second Class Object `MAPPING_OPERATOR`, which contains `ATTRIBUTES`. Other examples of SCOs include `COLUMN`, `FOREIGN_KEY`, and all the mapping operators.

As a rule of thumb for those accessing Warehouse Builder through the graphic user interface, Second Class Objects can only be manipulated through a First Class Object. Similarly, users who access Warehouse Builder through OMB Plus can only manipulate Second Class Object definitions through a command against a First Class Object.

(See also: **First Class Object (FCO)**, **Third Class Object**).

semi-additive

Describes a **fact/measure** that can be summarized through addition along some, but not all, dimensions. Examples include Headcount and On Hand Stock. (Contrast with: **additive**, **nonadditive**.)

sequence

A **database schema** object that is a series of sequential, generated numbers. Oracle stores sequences as rows in a single data dictionary **table** in the **SYSTEM tablespace**. A sequence definition indicates general information: the name of the sequence, whether it ascends or descends, the interval between numbers, and other information. In Warehouse Builder architecture, a sequence is a **First Class Object (FCO)**.

server

In a **client**/server architecture, the computer that runs Oracle software and handles the functions required for concurrent, shared data access. The server receives and processes the **SQL** and **PL/SQL** statements that originate from client applications.

service name

A service name is a short, convenient name mapped to a network address contained in a TNS connect descriptor. Users need only know the appropriate service name to make a TNS connection. (See also: **tnsnames.ora**.)

set based

Operating mode option. Enables Warehouse Builder to insert all data in a single **SQL** command. Using set based mode increases the speed of **DML** operations when the data is clean and does not require extensive auditing. (Contrast with: **row based**.)

short name

The short name is a unique identifier that is used as the root name for all related objects created in **Oracle Warehouse Builder**.

signature

A compact definition of a Warehouse Builder **metadata** object that contains just enough information about the object to perform a difference analysis. (See also: **signature snapshot**.)

signature snapshot

A **metadata snapshot** that captures **signature** information only. A signature snapshot can be used to generate comparison reports between the snapshot and the current repository object or another snapshot. (Contrast with: **full snapshot**.)

slowly changing dimension (SCD)

This is a **dimension** created to manage both current and historical data over time in a **data warehouse**. It is considered and implemented as one of the most critical **ETL** tasks in respect to keeping history of dimension records.

snapshot

See: **metadata snapshot**.

snowflake schema

A type of **star schema** in which the **dimension** tables are partly or fully normalized. (See also: **normalize**.)

Software Library

Contains the integrators currently installed for **Oracle Warehouse Builder**. Accessed through the Administration environment.

source

A **database**, application, **file**, or other storage facility from which the data in a **mapping** is derived.

source operator

An **operator** that serves as a **data source** in a **mapping**. A mapping depicts data **extraction** from a source operator, **transformation** using one **data flow operator** or more, and loading to a **target operator**. Examples of source operators include the Table operator, the Flat File operator, and the External Table operator.

SQL

See: **Structured Query Language (SQL)**.

SQL*Loader

An Oracle tool used to load data from **operating system** files into Oracle **database** tables. It is the most efficient way to load large amounts of data. (See also: **Structured Query Language (SQL)**.)

SQL*Plus

Oracle tool used to execute **Structured Query Language (SQL)** statements against an Oracle **database**. Oracle SQL includes many extensions to the ANSI/ISO standard SQL language. (See also: **PL/SQL**.)

star schema

A relational **schema** whose design represents a multidimensional data **model**. The star schema consists of at least one **fact table** and at least one **dimension table**. The tables are related through foreign keys. (See also: **snowflake schema**.)

Structured Query Language (SQL)

Structured Query Language, or SQL, is an industry-standard programming language developed specifically to support access to a **relational database**. Users describe in SQL what they want done, and the SQL language compiler automatically generates a procedure to navigate the **database** and perform the desired task. The general characteristics of the SQL language are:

- It supports operations on sets of data, as opposed to **row**-by-row processing.
- It can access data independently of the data's physical location
- It is non-procedural. In other words, it only describes the data to be retrieved, not the method by which the data is to be retrieved.

Oracle Database database supports SQL and **PL/SQL**. (See also: **SQL*Plus**.)

subject area

A classification system that represents and distinguishes parts of an organization or areas of knowledge. A **data mart** is often developed to support a subject area such as sales, marketing, or geography.

surrogate key

A unique **primary key** that is generated by an **RDBMS**. This key is not derived from any data in the **database** and its only significance is to act as the primary key. In **Oracle Warehouse Builder**, the Sequence **mapping operator** can be used to generate this type of key. (See also: **sequence**.)

synchronization

A method used to update object definitions with the changes committed by other users in a multiuser environment. The objects displayed in the **navigation tree** are synchronized with the object definitions in the **repository**. (See also: **commit**.)

SYS username

One of two standard DBA **username**s automatically created with each **database** (the other is the **SYSTEM username**). The Oracle user `SYS` is created with the password `MANAGER`.

SYSDBA

A special **database** administration **role** that contains all system privileges with the `ADMIN OPTION`, and the `SYSOPER` system privilege. `SYSDBA` also permits `CREATE DATABASE` actions and time-based recovery. (See also: **SYSOPER**.)

SYSOPER

A special **database** administration **role** that permits a database administrator to perform `STARTUP`, `SHUTDOWN`, `ALTER DATABASE OPEN/MOUNT`, `ALTER DATABASE`

BACKUP, ARCHIVE LOG, and RECOVER, and includes the RESTRICTED SESSION privilege. (See also: **SYSDBA**.)

system identifier (SID)

A unique name for an Oracle **instance**. To switch between Oracle databases, users must specify the desired system identifier. The system identifier is included in the CONNECT DATA parts of the connect descriptors in a **tnsnames.ora** file, and in the definition of the network listener in the alistener.ora **file**.

SYSTEM tablespace

The SYSTEM **tablespace** differs from other tablespaces in that all data files contained in the tablespace must be online for Oracle to function. If a media failure affects one of the datafiles in SYSTEM, then you must mount the **database** and recover.

SYSTEM username

One of two standard **database** administrator usernames automatically created with each database. (The other username is the **SYS username**.) The Oracle user SYSTEM is created with the password MANAGER. The SYSTEM username is the preferred **username** for database administrators to use for database maintenance.

table

The basic unit of storage in a **relational database management system (RDBMS)**. A table represents entities and relationships, and consists of one or more units of information (rows), each of which contains the same kinds of values (columns). Each column is given a column name, a **datatype** (such as CHAR, NCHAR, VARCHAR2, NVARCHAR2, DATE, or NUMBER), and a width (the width may be predetermined by the datatype, as in DATE). 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 and triggers can also be defined for a table. In Warehouse Builder architecture, a table is a **First Class Object (FCO)**. (See also: **row**, **column**.)

tablespace

A **database** storage unit that groups related **logical structures** together. A database is divided into one or more logical storage units called tablespaces. Tablespaces are divided into logical units of storage called *segments*, which are further divided into *extents*.

target

Holds the intermediate or final results of any part of the **extraction, transformation, and loading (ETL)** process. The **target** of the entire ETL process is often the **data warehouse**.

target operator

An **operator** that serves as a data **target** in a **mapping**. A mapping depicts data **extraction** from a **source operator**, **transformation** using one **data flow operator** or more, and loading to a target operator. Examples of target operators include the Table operator and the Flat File operator.

temporary (TEMP) tablespace

The **tablespace** of temporary tables or indexes created during the processing of a **SQL** statement.

Third Class Object

In **Oracle Warehouse Builder** architecture, Third and Fourth Class objects are relative rankings of objects owned by other objects. These refer only to objects whose ownership spans several layers. For example, `INDEX_COLUMN` is a Second Class Object in the scenario where a `DIMENSTION_TABLE` (which is a First Class Object) owns `INDEX_COLUMN`. However, `INDEX_COLUMN` becomes a Third Class Object in the scenario where the First Class Object `CUBE_TABLE` owns the Second Class Object `INDEX`, which in turn owns `INDEX_COLUMN`.

(See also: **First Class Object (FCO)**, **Second Class Object (SCO)**.)

tnsnames.ora

A **file** that contains connect descriptors mapped to net service names. The file may be maintained centrally or locally, for use by all or individual clients. (See also: **service name**, **client**.)

Transfer Wizard

The **Oracle Warehouse Builder** utility for synchronizing, integrating, and using **metadata** that is stored in the **Design Repository** in a variety of business intelligence tools. Metadata can be imported and exported using a bridge. Metadata can also be exchanged with OMG files, Oracle Discoverer, Oracle Express, ERwin, Predesignate, and Oracle Database OLAP Server.

transaction

1. In the context of databases: A logical unit of work that comprises one or more **SQL** statements executed by a single user. All statements in a transaction are committed or rolled back together. According to the ANSI/ISO SQL standard, with which Oracle is compatible, a transaction begins with the user's first executable SQL statement. A transaction ends when it is explicitly committed or rolled back by the user. (See also: **Online Transactional Processing (OLTP)**, **commit**, **rollback**.)
2. In the context of data warehousing concepts: The lowest grain of a business activity. Examples include: a bank deposit, an inventory movement of a single item, and a retail sale. **OLTP** systems are used to process transactions, whereas **OLAP** systems are used to analyze transactional data at a higher level. (See also: **Online Transactional Processing (OLTP)**, **Online Analytical Processing (OLAP)**.)

transformation

The process of manipulating data. Any manipulation beyond copying is a transformation. Examples include cleansing, aggregating, and integrating data from multiple **source** objects. In Warehouse Builder architecture, a transformation is a **First Class Object (FCO)**.

Transformation Library

Stores the reusable formulas for the **transformation** of data as it moves between **source** and **target** objects.

transition

In a **process flow**, transitions indicate the sequence and conditions in which activities occur in the process flow. You can use transitions to execute an **activity** based on the completion state of the preceding activity.

transparent gateway

Software product that allows **SQL**-based applications to access relational and non-relational data sources as if they were an Oracle **database**.

UDP

See: **user-defined properties (UDP)**.

unbound object

A **mapping** object **operator** that is not connected to an object defined in the **Design Repository**. (Contrast with: **bound object**.)

unpivot

In a **mapping**, the 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. (See also: **pivot**.)

user-defined properties (UDP)

To enhance the metadata for your unique needs, **Oracle Warehouse Builder** enables you to create additional properties for objects. You can create user-defined properties for any object type on the **navigation tree**. User-defined properties typically store additional business, design, or versioning information for objects. Any user-defined properties must be prefixed with `UDP_`.

user interface (UI)

The combination of menus, screens, keyboard commands, mouse clicks, and command language that defines how a user interacts with a software application. **Oracle Warehouse Builder** offers a graphic user interface, as well as a scripting interface in **Oracle MetaBase Plus (OMB Plus)**, and some command-line and **SQL** extensions.

username

The name by which a user is known to the Oracle **server** and to other users. Every username is associated with a password, and both must be entered to connect to an Oracle **database**.

validation

The process of verifying **metadata** definitions and configuration parameters.

view

A view can be thought of as a "stored **query**" with a custom-tailored presentation of data from one or many tables. A view does not actually contain or store data, but derives data from the base tables on which it is 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. In Warehouse Builder architecture, a view is a **First Class Object (FCO)**. (See also: **materialized view**.)

warehouse administrator

The warehouse administrator is the information specialist who manages the warehouse **database** and warehouse management applications. For example, the warehouse administrator would be responsible for managing and monitoring periodic updates of the warehouse database. (See also: **data warehouse**.)

watch

A point in a data flow that is used during a **debug** session to monitor the data that is passing through or has already passed through a particular **operator**.

XML

See: **eXtensible Markup Language (XML)**.

Index