Oracle9*i* Warehouse Builder

User's Guide

Release 9.0.2

January 9, 2002

Part No. A95949-01

ORACLE

Oracle9*i* Warehouse Builder User's Guide, Release 9.0.2

Part No. A95949-01

Authors: Shirinne Alison, Gillian Robinson, Patrick Terhune

Contributing Authors: Kavita Nayar, Yu Gong

Contributors: Thomas Lau, Edwin Meijer, Joseph Zheng, Barry Mosbrucher, Thomas O'Shaughnassy, Dan Gallagher, Josephine Ho, David Allen, Krishna Behara, Gary Hostetler, Jaime Singson, Robert Velisar, Robert Paul, Joseph Klein, Josef Hasenberger, Jean-Pierre Dijcks, Olaf Fermum, Shauna O'Boyle, Debra Phelan, Elina Sternik, John Potter, Marie Pajanor, Barry Mosbrucher, Adrian Scott, Brian Jeffries, Arun Manchanda, Richard Whittington, Ramesh Uppala.

Contents

Send Us Your Comments xix

Preface xxi

Audience xxi
How This Guide Is Organized xxii
Conventions xxiv
Related Documents xxiv
Documentation Accessibility xxv

1 Overview

The Development Phases 1-2
The Definition Phase 1-3
The Generation Phase 1-3
The Load and Manage Phase 1-4

2 Getting Started on a Project

Logging on to a Repository 2-2
About the Warehouse Builder Console 2-3
Console Toolbar 2-4
Navigation Tree Views 2-5
Project Tree 2-5
Transformation Tree 2-6
Administration Tree 2-7
Utilities 2-7

Displaying Utility Properties 2-8
Adding a Utility 2-9
Updating a Utility 2-10
Removing a Utility 2-10
Recycle Bin and Clipboard 2-10
Recycle Bin 2-11
Clipboard 2-12
Clipboard / Recycle Preferences 2-13
Working with Projects 2-15
About Warehouse Builder Projects 2-15
Creating a Project 2-16
Module Contents 2-17
About Multi-User Access 2-17
Read / Write Mode 2-18
Read-Only Mode 2-18
Synchronization 2-18
Creating Objects 2-19
Using the Warehouse Builder Wizards 2-19
Object Names 2-20
Logical Name Mode 2-21
Physical Name Mode 2-21
Setting the Name Mode 2-21
Editing Objects 2-23
Object Editors 2-23
Property Sheets 2-23
Finding Objects 2-24
Ending a Warehouse Builder Session 2-25
Committing Your Work 2-25
Exiting Warehouse Builder 2-25

3 Defining Relational Targets

Creating a Warehouse Target Module 3-2
Displaying the Warehouse Module 3-6
Creating Table Definitions 3-7
Creating a Table Definition 3-7

Updating Table Definitions ... 3-9
Using the Table Editor ... 3-9
Using the Table Property Sheet ... 3-10
Creating Attribute Sets ... 3-16
Importing Definitions ... 3-18
Creating Materialized View Definitions ... 3-19
Creating a Materialized View Definition ... 3-19
Updating a Materialized View Definition ... 3-23
Creating Conventional View Definitions ... 3-23
Creating a View Definition ... 3-23
Updating a View Definition ... 3-26
Defining a Sequence Object ... 3-27

4 Defining Dimensional Targets

About Star Schemas ... 4-2
About Dimension Tables ... 4-3
About Fact Tables ... 4-3
About Materialized Views ... 4-4
About Conventional Views ... 4-4
Creating Dimension Definitions ... 4-4
Rules for Dimension Objects ... 4-5
About Levels and Hierarchies ... 4-5
About Unique Key Constraints ... 4-5
About Mixed Levels of Aggregation ... 4-6
Creating a Dimension Definition ... 4-7
Updating Dimension Definitions ... 4-12
Using the Dimension Editor ... 4-13
Using the Property Sheets ... 4-14
Creating a Time Dimension Definition ... 4-15
Creating Fact Table Definitions ... 4-19
Creating a Definition for a Fact Table ... 4-19
Updating a Fact Table Definition ... 4-22
Using the Fact Editor ... 4-23
Importing Definitions ... 4-26
Adding Transformations ... 4-27

About Transformations 4-27
About Transformation Parameters 4-27
About Oracle Transformation Libraries 4-28
Global Shared Library 4-28
Oracle Library 4-28
Accessing Transformation Libraries 4-29
Creating Transformation Libraries 4-29
Defining Custom Transformations 4-29
Editing Transformation Properties 4-33
Importing PL/SQL Packages 4-33
Defining Business Areas 4-38

5 Defining Source Modules

Creating Source Modules 5-2
Configuring Connection Information for Database Sources 5-2
Creating a Database Source Module 5-5
Creating a Source Module for an Oracle Designer Repository 5-8
Creating a Source Module for Designer 6i 5-9
Creating a Flat File Source Module 5-13
Create a Source Module for SAP Definitions 5-15
Updating a Source Module 5-18
Importing Definitions from Database Sources 5-19
Importing Definitions from a Database 5-19
Re-Importing Definitions from an Oracle Database 5-23
Updating Oracle Database Source Definitions 5-26
Update a Source Definition 5-27
Update the Connection 5-28
Diagram a Source Definition 5-29
Creating Definitions for Flat File Sources 5-30
About the Flat File Sample Wizard 5-31
Creating a Definition for a Fixed-Length File 5-33
Defining Multiple Records in a Fixed-Length File 5-39
Specifying Logical Records 5-42
Creating a Definition for a Delimited File 5-43
Defining Multiple Record Types in a Delimited File 5-45

Updating a File Definition 5-47
Importing Definitions for Pure Extract and Pure Integrate 5-49
Importing Definitions for SAP Data Sources 5-49
Business Components 5-49
Import Metadata from SAP sources 5-50

6 Defining Source to Target Mappings

Understanding Warehouse Builder Mappings 6-2
About Mapping Operators 6-2
About Operator Properties 6-3
About Attributes and Attribute Groups 6-3
About Display Sets 6-3
About Binding Mapping Operators 6-4
Defining Mappings 6-4
About the Mapping Editor 6-5
Creating a Mmapping 6-7
Displaying the Mapping Editor 6-10
Selecting Data Operators 6-10
Selecting a Flat File Operator 6-13
Selecting an SAP Source 6-14
Selecting a Data Flow Operator 6-16
Connecting Mapping Operators 6-17
Connecting Attributes 6-17
Connecting Attribute Groups 6-19
Editing Mapping Operator Attributes 6-22
Adding or Removing Operator Attribute Groups 6-22
Editing Operator Attributes 6-23
Adding or Removing Attributes 6-23
Renaming Attributes 6-25

7 Using Mapping Operators and Transformations

Using Expression Builder 7-2
Adding Flow Operators to a Mapping 7-4
Adding Mapping Sequences 7-5
Adding Data Generators 7-6

Setting a Column to the Data File Record Number 7-6
Setting a Column to the Current Date 7-7
Setting a Column to a Unique Sequence Number 7-7
Adding Constants 7-9
Using Key Lookups 7-10
Using Set Operations 7-14
Joining Data Sources 7-16
Creating Full Outer Join Conditions 7-20
Splitting Data 7-21
Removing Redundant Data 7-24
Aggregating Data 7-25
Filtering Data 7-28
Ordering Data 7-31
Cleansing Name and Address Data 7-34
Adding a Pre-Mapping Process 7-42
Adding a Post-Mapping Process 7-44
Adding Mapping Input Parameters 7-46
Adding Mapping Output Parameters 7-49
Adding External Processes 7-52
Adding Transformations 7-54
Adding Expressions 7-57

8 Configuring and Generating Mappings

Reconciling Mapping Operators with Repository Objects 8-2
Using Inbound Reconciliation 8-2
Using Outbound Reconciliation 8-6
Configuring a Mapping 8-9
Configuring Mapping Table Operators 8-9
Configuring Flat File Operators 8-13
Configuring Transformation Operators 8-14
Configuring Mapping Operator Attributes 8-14
Setting Attribute Properties 8-15
Setting Loading Properties 8-15
Configuring Physical Properties for a Mapping 8-16
Setting Deployable 8-19

Setting the Step Type 8-19
Setting Runtime Parameters 8-19
Setting Sources and Targets 8-21
Configuring Flat File Physical Properties 8-26
Configuring SAP File Physical Properties 8-32
Setting the Step Type 8-32
Setting the Runtime Parameters 8-32
Validating and Generating a Mapping 8-33
Validating a Mapping 8-33
Generating a Mapping 8-35
Viewing the Generated Code for a Mapping 8-37
Viewing Intermediate Results 8-37
Viewing Code from the Module Editor 8-37

9 Configuring, Generating, and Deploying

Configuring a Physical Instance 9-2
Setting up Configuration Properties 9-3
Guidelines for Configuring Warehouse Modules 9-3
Guidelines for Configuring Dimensions, Facts, and Tables 9-6
Guidelines for Configuring Materialized Views 9-8
Guideline for Configuring Sequences 9-10
Guidelines for Configuring Views 9-11
Creating Database Links, Indexes, and Partitions 9-11
Creating Database Links 9-12
Creating Partitions 9-17
Defining Hash Partitions 9-21
Defining Range Partitions 9-21
Defining a Range with Hash Subpartitions 9-22
Configuring Partition Exchange Loading (PEL) 9-22
Validating Definitions and Generating Scripts 9-23
Validating Definitions 9-23
Valid and Invalid Definitions 9-25
Generating Scripts 9-25
Verifying the Generation Results 9-27
Database Links 9-27

Dimensions 9-28
Views and Materialized Views 9-29
Mappings 9-29
Sequences 9-30
Tables 9-30
Post-Generation Tasks 9-30
Schema Objects Tab 9-31
Transforms Tab 9-31
Deploying Scripts 9-33
Deploying SAP Definitions 9-37
Deploying ABAP Scripts for Non-Transparent Tables 9-38
Deploying ABAP Scripts Manually 9-38
Deploying PL/SQL Scripts for Transparent Tables 9-38
Upgrading the Warehouse 9-39
Generating and Executing Upgrade Scripts 9-40

10 Managing Loads and Updates

Registering Tcl Scripts 10-2
Creating an Oracle Workflow 10-4
Deploying Scripts to the Workflow Server 10-5
Defining the Workflow Process 10-10
Scheduling a Workflow 10-13
Viewing the Job 10-14
Changing Job Parameters 10-15
Viewing the Results 10-16
About the Runtime Audit Viewer 10-16
Viewing the Job Audit 10-17
Viewing the Job Instance Audit 10-18
Viewing the Task Audit 10-19
Viewing the Task Details Audit 10-21
Using the Warehouse Builder Runtime Audit Viewer 10-22
Viewing Specific Results 10-24
Refreshing the Audit Viewer 10-27
Purging Runtime Entries 10-27

11 Administration

About the Warehouse Builder Metadata Loader 11-2
Metadata Export Utility 11-3
Export File 11-3
Exporting Metadata 11-5
Metadata Import Utility 11-7
Importing Metadata 11-9
Metadata Loader Command Line Utility 11-15
Directive Keywords for the Export Utility 11-16
Export a Project 11-18
Directive Keywords for the Import Utility 11-18
Import Selected Modules 11-19
Validation Rules Governing Import 11-20
Splitter for Exporting and Importing OWB Mappings 11-20
About Batch Services 11-24
Using the Batch Services Command Line Syntax 11-25
Running Multiple Command Line Operations 11-30
Locating and Reviewing Log Files 11-33
About the Warehouse Builder Transfer Wizard 11-34
Warehouse Builder Transfer Wizard Overview 11-34
Transfer Considerations 11-35
Importing Metadata into Warehouse Builder 11-35
Exporting Metadata from Warehouse Builder 11-39
About the Archive/Restore Utility 11-44
Setting up Preferences 11-45
Archiving a Project 11-47
Restoring a Project 11-50
Using Discoverer Workbooks for Metadata and Runtime Reporting 11-53
Installing the EUL Template and Workbooks 11-54
Prerequisites 11-54
Setting Database Connections 11-54
Creating Specific .eex Files 11-55
Importing .eex Files into Discoverer 11-56
Accessing the EUL within Discoverer Administration 11-56
Accessing Workbooks using Discoverer User Edition 11-59

Navigating Workbooks 11-60
Definition Workbook 11-60
QA Workbooks 11-67
Runtime Workbooks 11-72
Improving Performance of Workbooks 11-74

12 Metadata Reporting Using Warehouse Builder Browser

Accessing Reports from the Warehouse Builder Client 12-2
Setting up Browser Preferences 12-2
Opening the Reports 12-3
Accessing Reports from the Warehouse Builder Browser 12-4
Starting the Warehouse Builder Browser 12-4
Adding Portlets to the Warehouse Builder Browser 12-6
Available Warehouse Builder Browser Portlets 12-9
Using Warehouse Builder Browser to View Reports 12-14
About Warehouse Builder Reports 12-15
Available Warehouse Builder Reports 12-16
Summary Reports 12-16
Detailed Reports 12-17
Implementation Reports 12-19
Lineage and Impact Analysis Reports 12-20
Lineage and Impact Analysis Diagrams 12-21
Using Warehouse Builder Browser 12-23
Navigating the Warehouse Builder Browser 12-23
General Page Information 12-24
Current Item Information 12-24
Properties Tab 12-25
Contents Tab 12-26
Related Tab 12-27
Reports Tab 12-28
Links Tab 12-29
Browsing Favorites 12-33
General Page Information 12-34
Filter 12-34
Navigation Favorites 12-35

Reports Favorites ... 12-35
Customizing Your Favorites Page ... 12-36
Creating Custom Reports ... 12-38
Creating a Custom Report ... 12-38
Registering a Custom Report ... 12-40
Adding a Custom Report to a Role ... 12-40
Viewing a Custom Report ... 12-40
Creating Other Custom Reports ... 12-40
Using the Administration Pages ... 12-45
Registering a Warehouse Builder Repository ... 12-46
Administering Database Links ... 12-48
Unregistering a Repository ... 12-52
Registering a Custom Report ... 12-52
Purging Stale User Information ... 12-53
Resource Management ... 12-54
Managing Portlet Access ... 12-55
Managing Repositories ... 12-56
Managing Custom Reports ... 12-56
Understanding Roles ... 12-56
Managing Preferences ... 12-57
Saving Preferences ... 12-58
Loading Preferences ... 12-58
Managing the Dependency Index ... 12-60
Setting the Refresh Options ... 12-60
Refreshing the Dependency Index on Demand ... 12-61

A Keyboard Shortcuts

Keyboard Command Access ... A-2
General Windows Keys ... A-2
Tree View Control Keys ... A-2
Accelerator Keys Set for Warehouse Builder ... A-3
Menu Commands and Access Keys ... A-3
Auto-Mapping Mnemonic Key Assignments ... A-4

B Reserved Words

Reserved Words B-2

C Mapping User Interface

Mapping Editor C-1
Mapping Menu Bar C-2
Mapping Editor Toolbar C-5
Toolbox C-5
Operator Property Inspector C-6
Using the Mouse in the Mapping Editor C-7
Left Mouse Click Operations C-7
Tool Tips C-8
Right Mouse Click Operations C-8
Keyboard Operations C-10
Other Dialogs C-11
Add Operator Dialog C-11
Add/Remove Attributes Dialog C-12
Inbound and Outbound Reconcile Dialog C-12
Physical Properties Inspector C-13

D Performance Enhancements

Overview D-2
Overhead-Free INSERT D-3
Fast Index Creation D-4
Fast Constraint Maintenance D-4
Error Handling D-4
Minimum Target Locking D-5
Parallel Direct-Path INSERT D-5
When to Use Partition Exchange Loading D-6
Using Partition Exchange Loading D-7
Configuring the Mapping D-9
Configuring the Target D-10
Create All Partitions D-10
Create All Indexes Using the LOCAL option D-14

Specify the USING INDEX Option for Primary / Unique Keys D-14
Restrictions D-15
Restriction 1: Allow One Date Partition Key D-16
Restriction 2: Allow Natural Calendar System Only D-16
Restriction 3: All Data Partitions Must be in the Same Tablespace D-16
Restriction 4: All Index Partitions Must be in the Same Tablespace D-16
Performance Comparisons D-16
The Internal Workings D-17
Function Example 1: GET_PN D-19
Function Example 2: GET_VC D-20

E Batch Services API

Repository E-2
Open Repository Connection E-2
Close Repository Connection E-2
Projects E-2
Modules E-3
Validation E-3
Generation E-4
Deployment E-4
Deployment into a File System E-4
Deployment in a Database E-4
Metadata Import E-5
Metadata Export E-5

F Using the XML Toolkit

Retrieving Data From Sources F-1
Storing Data in Targets F-2
Using Runtime Controls F-2
How to Call the XML Toolkit F-2
Two Entrances F-3
Typical Control Files F-4
XML Documents Stored in Files F-5
The XML Document F-5
The Target Table F-6

The Control File F-6
About the dateFormat Attribute F-6
A Warehouse Builder Transformation F-7
The Target Table F-8
The Control File F-8
The Style Sheet F-9
A Warehouse Builder Transformation F-9
Multiple Targets F-10
XML Documents Stored as Other Objects F-11
Document Stored as a CLOB F-12
Document Stored as an Object-Based Advanced Queue F-13
Document Stored at a URL F-15
Control File Element Names and Attributes F-16
Source Elements F-16
Target Elements F-18
Runtime Control F-19
Document Type Definition for the Control File F-20
Reference Materials F-21

G OWB Public View Tables Management

Data Model G-3
Implementation Model G-10
Flat File / Record Model G-11
Impact Analysis / Data Lineage Model G-14
Classification Model G-17
Expression & Function Model G-18
Configuration & Deployment Model G-20
Transformation / Mapping Model G-20
Runtime Public Views G-23

H Warehouse Builder Bridges: Transfer Parameters and Considerations

Transfer Parameters H-1
Transfer Considerations H-7
Importing Metadata from an Object Management Group Common Warehouse Metamodel Standard System H-8

Importing Metadata from Computer Associates ERwin 3.5.1 ..... H-9
Importing Metadata from Powersoft PowerDesigner 6.0 ..... H-9
Importing Metadata from Oracle9*i* OLAP server H-10
Exporting Metadata to Oracle Discoverer 3.1 and 4i H-10
Configuring Warehouse Builder for Dimensional Reuse H-12
Defining Dimensions and Facts in Warehouse Builder H-12
Defining the Dummy Tables H-13
Hiding Data Prior to Transfer H-14
Importing Transferred Data into Discoverer H-14
Dimensional Reuse Naming Conventions in Discoverer H-14
Exporting Metadata to an Object Management Group CWM Standard System H-16
Exporting Metadata to Oracle Express H-16
OSA Configuration H-17
Exporting Metadata to Oracle9*i* OLAP server H-18

Glossary

Send Us Your Comments

Oracle9*i* Warehouse Builder User's Guide, Release 9.0.2

Part No. A95949-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail - dwhdoc_us@oracle.com
- Postal service:
  Oracle Corporation
  MS 2op713
  500 Oracle Parkway
  Redwood Shores, California 94065
  U.S.A.

If you would like a reply, please give your name, address, and telephone number below.

If you have problems with the software, please contact your local Oracle World Wide Support Center.

Preface

Oracle9*i* Warehouse Builder is a comprehensive toolset designed for practitioners who develop and implement data warehouses. 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 logical warehouse.
- Validate the set of definitions and their configurations.
- Generate a set of scripts to create and populate the physical database instance.
- Generate data transformation scripts.
- Deploy and initially load the physical instance.
- Maintain the physical instance by conditionally refreshing it with generated scripts.

Audience

This guide is intended for data warehouse practitioners including:

- Warehouse Architects, Designers, and 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 must know SQL and PL/SQL. You need to be comfortable with concepts of Relational Database Management Systems and Data Warehouse design. For information on data warehousing, refer to the *Oracle8i/9i Data Warehousing Guide.* Also, you need to be familiar with Oracle's relational database software products such as Oracle8*i*/9*i*, SQL*Plus, SQL*Loader, Oracle Enterprise Manager, and Oracle Workflow.

How This Guide Is Organized

The *Oracle9i* Warehouse Builder User's Guide contains the following chapters and appendixes.

Chapter 1, "Overview", introduces you to Oracle9*i* Warehouse Builder and describes the design and implementation process for a data warehouse.

Chapter 2, "Getting Started on a Project", introduces the Warehouse Builder repository and how to log on to the repository using the Warehouse Builder client. It also describes how to start a project, and how Warehouse Builder handles multiple users on the same module.

Chapter 3, "Defining Relational Targets", describe how to define relational model targets and how to edit their properties.

Chapter 4, "Defining Dimensional Targets" describes how to define dimensional model targets and how to edit their properties.

Chapter 5, "Defining Source Modules" shows you how to create definitions for data sources using Warehouse Builder wizards. The chapter describes how to import the definitions from Oracle and non-Oracle database systems, and Designer repositories. It shows you how to import data from a flat file source and create formats for the files. This chapter also shows you how to modify definitions, generate and display diagrams for the source objects, and print diagrams.

Chapter 6, "Defining Source to Target Mappings" introduces the concepts of mapping data sources to data targets. It also describes how to create logical definitions that map source operator attributes to target operator attributes.

Chapter 7, "Using Mapping Operators and Transformations" describes the various tools available for transforming data as it moves from the source to the target. The chapter describes the Oracle Transformation Library, the Global Shared (transformation) Libraries, Expression Builder, custom transformations, and SQL operations such as filters, splitters, and aggregation functions.

Chapter 8, "Configuring and Generating Mappings" describes how to configure mapping operators, attributes, and physical properties. It also describes how to define code generation strategies for optimal performance during deployment.

Chapter 9, "Configuring, Generating, and Deploying", describes how to configure and validate the definitions for a specific instance. The chapter then describes how to generate and deploy the scripts that extract, transform, and load data into your data warehouse. These scripts create and populate the data warehouse with dimensions, facts, views, and materialized views.

Chapter 10, "Managing Loads and Updates", describes how to load your data warehouse. It then describes how to register job scripts with Oracle Enterprise Manager to schedule periodic loading into the warehouse.

Chapter 11, "Administration", describes administrative tasks on your warehouse using Warehouse Builder utilities. It describes how to create and drop Warehouse Builder repositories, how to install or drop a Warehouse Builder Runtime Catalog, how to display audit statistics and error messages stored in the Runtime tables, and how to export definitions from a Warehouse Builder repository and import definitions from a Warehouse Builder Metadata file.

Chapter 12, "Metadata Reporting Using Warehouse Builder Browser", describes how to use Oracle Portal to view reports and create your own reports on metadata.

Appendix A, "Keyboard Shortcuts" lists keyboard shortcuts for Warehouse Builder commands.

Appendix B, "Reserved Words", lists the words reserved for Warehouse Builder.

Appendix C, "Mapping User Interface" provides a reference for the Mapping Editor.

Appendix D, "Performance Enhancements" describes performance improvements.

Appendix E, "Batch Services API" describes how to use the Warehouse Builder API commands in batch mode.

Appendix F, "Using the XML Toolkit", describes how to retrieve and store data from XML documents in a target schema.

Appendix G, "OWB Public View Tables Management", 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 H, "Warehouse Builder Bridges: Transfer Parameters and Considerations" describes paramenters required for importing and exporting metadata.

Conventions

The SQL*Plus interface to Oracle8*i* may be referred to as SQL. This interface is the Oracle8*i*/9*i* implementation of the SQL standard ANSI X3.135-1992, ISO 9075:1992, commonly referred to as the ANSI/ISO SQL standard or SQL92.

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 conventions are also 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.
[]	Brackets enclose optional clauses from which you can choose one or none.

Related Documents

For more information, see the following manuals:

- *Oracle 9i Warehouse Builder Installation Guide*
- *Oracle9i Warehouse Builder Release Notes*
- *Oracle8i/9i Data Warehousing Guide*

Oracle provides additional information sources, including other documentation, training, and support services that can enhance your understanding and knowledge of Oracle9*i* Warehouse Builder.

- For more information on Oracle9*i* 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 Oracle9*i* Warehouse Builder, visit MetaLink at:

  http://metalink.oracle.com

- You can order other Oracle documentation at:

  http://oraclestore.oracle.com

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. Standards will continue to evolve over time, and Oracle Corporation 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 additional information, visit the Oracle Accessibility Program Web site at:

http://www.oracle.com/accessibility/

1

Overview

Oracle9*i* Warehouse Builder is a comprehensive and powerful tool that enables you to define and maintain a large scale logical model of a data warehouse and to deploy a physical instance of that model. This chapter describes the process of using Warehouse Builder to design, implement, and load a data warehouse. This guide provides instructions on how to use Warehouse Builder to:

- Create a set of definitions for a logical warehouse schema, its data sources, and the operations that extract data from a source, transform the data, and load the data into a warehouse.
- Configure a physical instance of the logical warehouse, validate the set of definitions for the instance, generate a set of scripts that create and load a physical instance, and deploy a physical instance to an Oracle8*i*/9*i* database system.
- Update and refresh the data warehouse, and set up Oracle Enterprise Manager and Oracle Workflow to manage the warehouse operation.

The Development Phases

There are three development phases in the data warehouse building process:

1. **Definition Phase**

   During this phase, you create logical definitions that describe a warehouse and its sources. These define:

   - The warehouse schema
   - Data sources and targets
   - Mappings of the extract, transform, and load (ETL) operations

2. **Generation Phase**

   A physical instance of the warehouse is defined by configuring a set of logical definitions. The configured logical definitions are validated and then used to generate a set of scripts that create and manage the physical instance. The scripts include:

   - Data definition language (DDL) code to create the warehouse and intermediate schema objects.
   - PL/SQL, SQL*Loader, and Tcl code to extract data, map and transform the data, and then load it into the physical instance.

   Warehouse Builder runs the DDL scripts that deploy the logical definitions as physical objects to create the physical instance. The Tcl scripts can also be deployed to a file system to use in conjunction with Oracle Workflow and Oracle Enterprise Manager to schedule and manage load and refresh jobs.

3. **Load and Manage Phase**

   During this phase, the infrastructure for the load and refresh jobs is implemented:

   - Job dependencies can be defined in the Oracle Workflow process.
   - Oracle Enterprise Manager can be used to schedule mappings, jobs, or an Oracle Workflow process.

Refer to the *Oracle Data Warehousing Guide* for information on data warehouse design and data warehousing strategies.

The Definition Phase

During this phase, you create numerous logical definitions to describe your data warehouse. These logical definitions describe a data warehouse schema, your data sources, optional intermediate staging schemas, and the target warehouse.

After you identify the different sources of data for your warehouse, you create a source module. A module contains logical definitions for the repository objects. You then use the Import Metadata Wizard to import definitions of data sources from:

- Database Sources including Oracle Designer repositories
- Flat File Sources
- Pre-packaged Applications

You then create a complete set of definitions for the target schema of the data warehouse. These definitions are stored in Warehouse Modules. The source and target definitions are used in the design, development, and implementation of routines that extract, transform, and load data into target warehouse schemas.

Finally, you create logical definitions that describe how to extract, transform, and load the data. These definitions are called Mappings and are stored in the warehouse module that defines the target warehouse.

The Generation Phase

After defining the logical warehouse, you configure Warehouse Builder to create and load a physical instance of the warehouse. Warehouse Builder then validates the definitions for the physical instance and generates the scripts to create objects for the instance. This phase consists of the following steps:

1. Configuring the Logical Definitions

   The configuration parameters determine how an object is deployed, the processing characteristics of selected objects, the location of deployed scripts, the physical properties of warehouse objects, and other properties.

2. Validating the Logical Definitions

   After you configure a physical instance, validate the set of definitions to detect script errors such as invalid foreign key references, invalid object references in mappings, and data type mismatches.

3. Generating the Deployment Scripts

   After you have a validated set of configured definitions, generate the scripts.

4. Deploying the warehouse

   After generating the scripts, you can deploy the database objects (database links, tables, dimensions, facts, materialized views, synonyms, and PL/SQL packages) to one or more instances. This creates the empty warehouse.

The Load and Manage Phase

After you deploy scripts, you can register them as jobs with Oracle Enterprise Manager or another scheduling tool. You can use a dependency management tool such as Oracle Workflow to run multiple job processes with dependencies. Schedule these Workflow processes to load or update the warehouse. The Warehouse Builder Workflow Queue Listener monitors the processes and ensures that dependencies are handled in the correct order. After the processes have completed, view the results using the Warehouse Builder Runtime Audit Viewer.

Warehouse Builder enables you to complete several administrative tasks using Warehouse Builder Utilities including the Runtime Assistant, the Run Time Audit Viewer, the Metadata Loader, and the Warehouse Builder Browser reporting tool.

2

Getting Started on a Project

This chapter describes how to start using Warehouse Builder. It also introduces you to the Warehouse Builder Console and some basic Warehouse Builder tasks.

This chapter includes the following topics:

- Logging on to a Repository
- About the Warehouse Builder Console
- Working with Projects
- Creating Objects
- Editing Objects
- Ending a Warehouse Builder Session

Logging on to a Repository

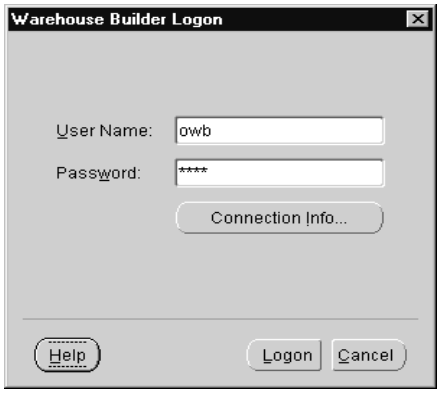
The following instructions show you how to start the Warehouse Builder client.

To log on to a repository:

1. From the **Start** menu, select **Oracle** and then **Warehouse Builder Client.**

   The Warehouse Builder Logon dialog displays.

Figure 2–1 Warehouse Builder Logon



2. Click **Connection Info** and enter the following connection information to connect to an Oracle8*i*/9*i* database instance:
   - Host Name
   - Port Number
   - Oracle SID
3. Click **OK.**
4. Enter the name and password of the repository owner
5. Click **Logon.**

   Warehouse Builder displays the Project dialog.

Figure 2–2 Project Dialog

This repository contains only the shell project My Project.

When a repository contains multiple projects, you can select a project from the drop-down list. This list includes all projects in the repository.

Note: If the Oracle8*i*/9*i* database instance is inactive, Warehouse Builder displays a connection error message.

---

6. Click **OK.**

   Warehouse Builder displays the console window in the Project view. This window remains open during the session.

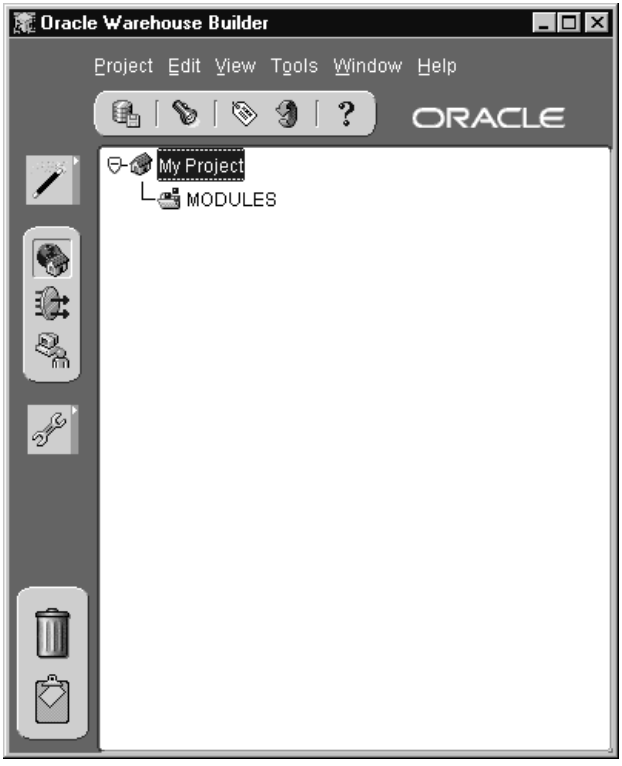
About the Warehouse Builder Console

The Warehouse Builder console has the following components:

- Menu bar
- Console toolbar
- Wizards buttons
- Tree View buttons
- Navigation tree window
- Utilities buttons
- Recycle Bin / Clipboard buttons

Figure 2–3 shows the console in the Project view.

Figure 2–3 Warehouse Builder Console



Console Toolbar

From the console toolbar, you can:

- Commit changes to the repository.
- Search the repository for an object.
- Look at the properties of an object.
- Open the Help system.

Figure 2–4 Console Toolbar

You can also perform these actions from the menus.

Navigation Tree Views

The Warehouse Builder console has the following views:

- Projects
- Transformations
- Administration

Figure 2–5 Warehouse Builder View Buttons

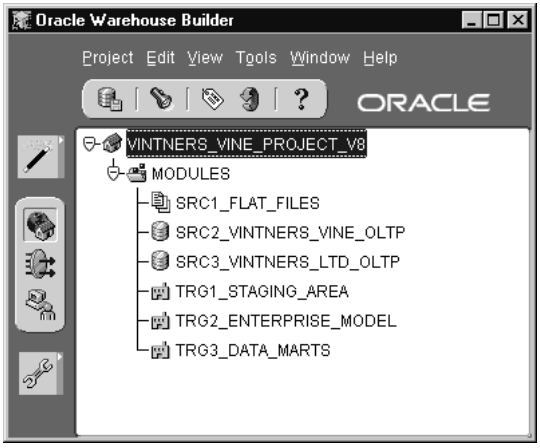
When the Warehouse Builder client starts, the console begins in the Project view and displays the navigation tree for the project selected when you logged on. To change views during a session, click one of the view buttons.

Project Tree

In the Project environment, you can:

- Create and modify definitions for repository objects such as dimensions, facts, relational tables, materialized views, sequences, and views.
- Create, modify, and import definitions for data sources, such as relational schema objects and flat files.
- Create and modify transformation operations.
- Create and modify mappings to direct the flow of data from sources to transformations to target schemas.
- Configure and validate definitions.
- Generate, deploy, and run scripts based on the definitions to create and load a target warehouse or staging area schema.
- Export metadata for a project, its modules, or object definitions.

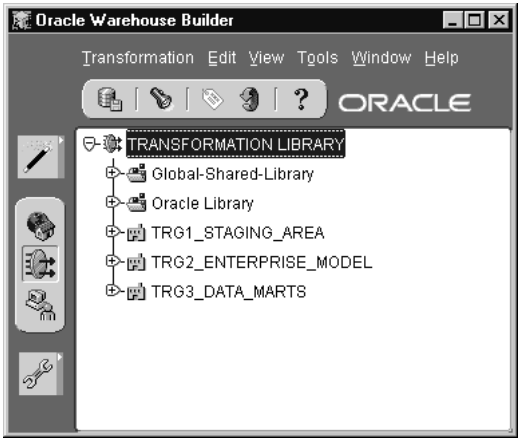
Figure 2–6 Sample Project Tree



Transformation Tree

You can create transformation libraries and transformations within libraries when the console is in the Transformation environment. Figure 2–7 shows the navigation tree for the Transformation Library.

Figure 2–7 Transformation Tree



The tree provides access to the built-in Oracle Library as well as the transformation operations created by developers for your warehouse.

Administration Tree

In the Administrative environment, you can:

- Create, modify, and display the properties of a Project.
- Import metadata from a Warehouse Builder export file.

Figure 2–8 shows the Administration tree. The tree contains Projects and the Integrators supplied with Warehouse Builder. The Integrators can retrieve metadata from applications based on relational databases, flat files, and proprietary systems.

Figure 2–8 Administration Tree

Utilities

Utilities are executables that are separate from Warehouse Builder. You can run a utility from the Warehouse Builder console using the Utilities icon.

To run a utility from the Warehouse Builder console:

1. Click the Utilities icon.

   Warehouse Builder expands the icon into the Utilities drawer.

Figure 2–9 Utilities Drawer

2. Click the utility icon.

   Warehouse Builder opens the utility program.

The Utilities Drawer is configured with the following utilities:

- SQL*PLUS
- Runtime Assistant
- Oracle Enterprise Manager
- Oracle Workflow

You can configure these icons to refer to the utilities installed at your site.

The following instructions describe how to display utility properties, add a new utility, update the configuration of an existing utility, and delete an existing utility.

Displaying Utility Properties

The Preferences sheet contains utility properties.

To display the Preferences sheet:

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

   Warehouse Builder displays the Preferences dialog.

2. Click the **Utilities** tab.

   Warehouse Builder displays the Utility sheet.

Figure 2–10 Utilities Sheet

Adding a Utility

To add a utility to the Utilities Drawer:

1. On the Utility sheet, type over existing information to enter the new utility 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. On the Utilities sheet, select the utility name from the Contents list.

   Warehouse Builder displays the configuration information.

2. Modify the 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 Utilities Drawer:

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

   Warehouse Builder removes the utility from the Utility Drawer.

Recycle Bin and Clipboard

The Recycle Bin and Clipboard buttons are located in the lower left corner of the Warehouse Builder Console. The Recycle Bin stores deleted objects with the option of restoring them at a later time. The Clipboard stores one cut or copied object in Warehouse Builder that can be pasted at a later time.

Use the Clipboard / Recycle tab from the Preferences dialog to specify how to handle the contents of the Recycle Bin and Clipboard when you exit Warehouse Builder. For more information, see "Clipboard / Recycle Preferences" on page 2-13.

Figure 2–11 Recycle Bin and Clipboard Icons

Recycle Bin

The Recycle Bin stores objects you delete from the navigation tree. Use the Recycle Bin dialog to restore deleted objects back to the navigation tree (repository), or to clear all the objects from the Recycle Bin.

> **Note:** When you delete an object, you are prompted to specify whether to save the object in the Recycle Bin.

You can also restore deleted objects into a different repository as a way of moving objects across repositories. In order to do this, your Recycle Bin preferences must be set to persist the Recycle Bin contents across multiple sessions. When you open the new repository, the Recycle Bin contents are the same. Select an object and click **Undo Delete**. The object will be added to the current repository. An object that is restored using the Undo Delete function retains its unique object identifier (UOID) but loses its foreign key references.

Because the Warehouse Builder Recycle Bin is stored on the client machine, it cannot be shared by multiple users. You can set up how to handle the contents of your Recycle Bin across sessions by setting up your Recycle Bin Preferences. For more information, see "Clipboard / Recycle Preferences" on page 2-13. The Recycle Bin is not affected by commit and rollback actions.

Table 2–1 lists the actions available for the Recycle Bin.

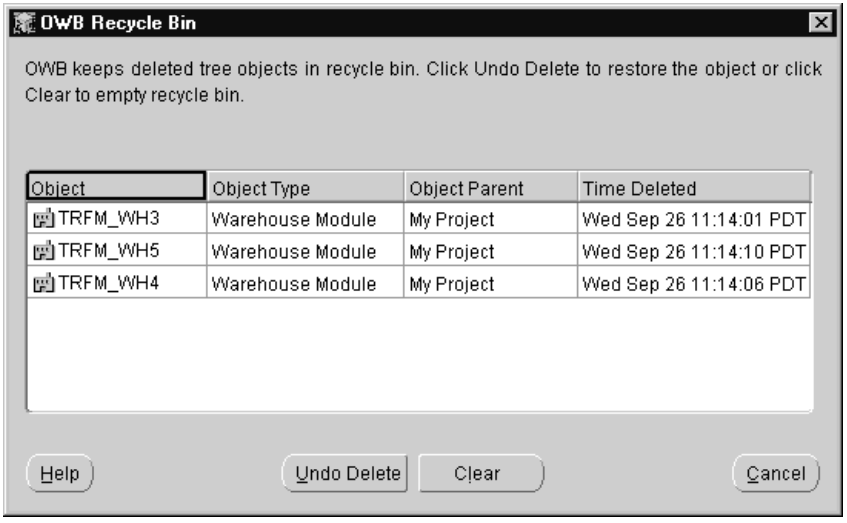
Table 2–1 Recycle Bin Functions

Function	Description
Delete	Deletes the selected object from the navigation tree and the Repository and stores it in the Recycle Bin until it is cleared. When you select this option, you are prompted to specify whether or not to save the object in the Recycle Bin.

Table 2–1 Recycle Bin Functions

Function	Description
Undo Delete	Restores a deleted object in the Recycle Bin in the navigation tree and the repository. If you have the Recycle Bin open, you can select a deleted object to be restored. If you use the edit menu option, only the last deleted object can be restored. To restore additional objects, open the Recycle Bin, select an object and use the Undo Delete button.
Clear	Empties the contents of the Recycle Bin. Once this has been done, you cannot restore any previously deleted objects if your changes have been committed.

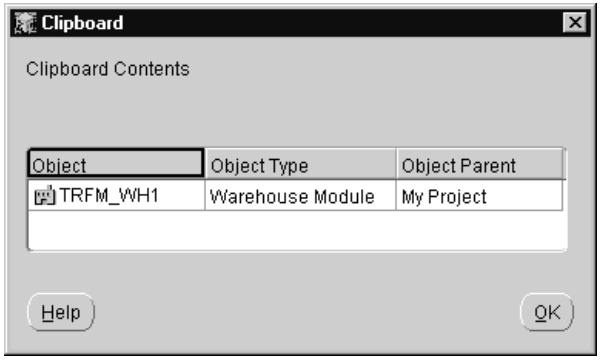
Figure 2–12 Recycle Bin



Clipboard

Warehouse Builder enables you to use cut, copy, and paste to edit objects in your repository. When you select an object and choose **Cut** or **Copy** from the **Edit** menu, the object is stored in the clipboard. You can then choose to Paste that object within the same repository or in a different repository. When you click the clipboard icon, the clipboard contents display in the Clipboard dialog.

Figure 2–13 Clipboard Dialog



You can either use the Edit menu to select these actions or use the quick keys listed in Table 2–2.

Table 2–2 Editing Quick Keys

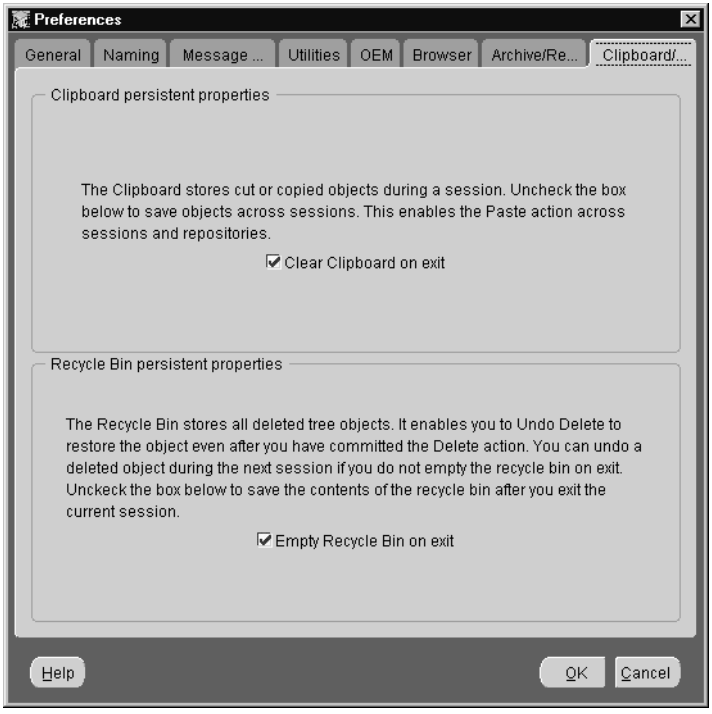
Function	Quick Keys	Description
Cut	Ctrl+X	Removes the selected object from the navigation tree and stores it in the clipboard.
Copy	Ctrl+C	Copies the selected object from the navigation tree and stores it in the clipboard.
Paste	Ctrl+V	Creates a copy of the object in the clipboard and inserts it into the navigation tree.

Clipboard/Recycle Preferences

To set Clipboard/Recycle preferences:

1. From the **Project** menu, select **Preferences.**
2. Select the **Clipboard/Recycle** tab.

Figure 2–14 Clipboard/Recycle Sheet



3. Select the check boxes to determine how you want to handle persistence of the clipboard and recycle bin upon exiting Warehouse Builder.

   Clipboard persistence properties: the clipboard stores the object you last cut or copied from the Warehouse Builder tree. Because the clipboard is specific to your Warehouse Builder client, you can set a preference indicating whether you want the object stored in it for only one Warehouse Builder session or across all Warehouse Builder sessions. The default setting is to empty the clipboard each time you log out of a session.

   If you want to paste the cut or copied object into another repository, or during your next Warehouse Builder session, you must uncheck the **Clear Clipboard on exit** box.

   Recycle bin persistence properties: the recycle bin stores all the objects you delete from the Warehouse Builder tree. Because the recycle bin is specific to your Warehouse Builder client, you can set a preference indicating whether you want the objects stored in it for only one Warehouse Builder session or across all

Warehouse Builder sessions. 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. The recycle bin is unaffected by Commit and Rollback actions. To preserve the objects in the recycle bin across sessions, you must uncheck the **Empty Recycle Bin on Exit** box.

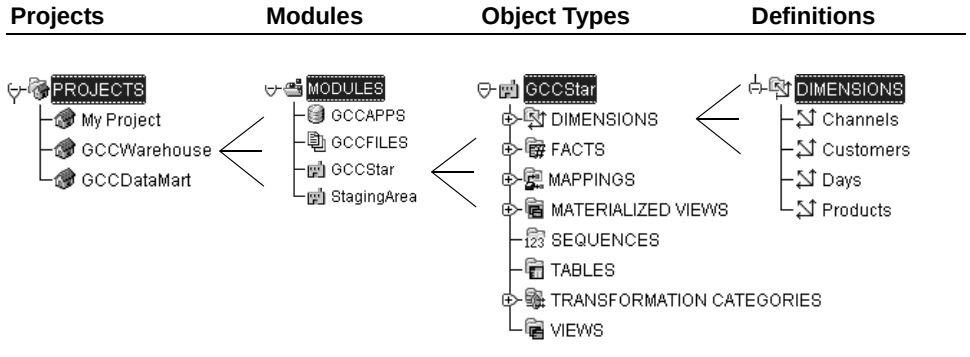
Working with Projects

A Warehouse Builder Project is a repository structure that stores and organizes the definitions of a data warehouse. These definitions include data sources, target warehouse objects, mappings of source data to various targets, transformation operations, and configuration parameters.

About Warehouse Builder Projects

Warehouse Builder projects are divided into containers called modules. Modules contain objects. Figure 2–15 shows the structure of a Project.

Figure 2–15 Warehouse Builder Project Structure



The navigation tree shows the modules within the project. A project contains the following types of modules, represented by different icons:

- Warehouse target modules
- Relational database source modules
- File source modules

Creating a Project

The initial Warehouse Builder repository contains the following:

- The Oracle Library of transformation operations
- A Global Shared Library
- A project shell named My Project

The project shell is required for the initial logon sequence and can be deleted after you create your own projects.

> **Note:** You cannot delete the currently active project.

To create a new project:

1. Click the icon for the Administration view.

   Warehouse Builder displays the Administration tree.

2. Click the plus sign next to ADMINISTRATION.

   Warehouse Builder displays the Projects and Integrators branches in the tree.

3. Right-click **Projects** and select **Create Project.**

   Warehouse Builder displays the welcome page for the New Project wizard.

4. Click **Next.**

   Warehouse Builder displays the Name page.

5. Enter the Project name and a description and then click **Finish.**

   Warehouse Builder creates a project and inserts its name under PROJECTS.

Figure 2–16 Projects Branch in the Administration Tree

Module Contents

Source modules contain definitions of:

- Relational database objects
- Flat files
- Proprietary structures

Target warehouse modules contain definitions of:

- Dimensions
- Facts
- Tables
- Mappings
- Transformations
- Materialized Views
- Views
- Sequences
- Scripts generated by Warehouse Builder

About Multi-User Access

Multi-user access ensures that only one user has write privileges to an object while allowing read-only access on the object to other users. The lock is maintained for the duration of any transaction started by the write-privileged user. The lock is released when the user commits the changes or performs a rollback and closes all editors associated with the object.

> **Note:** If you generate code at any time during your Warehouse Builder session, multi-user locking will remain in effect until you commit your changes.

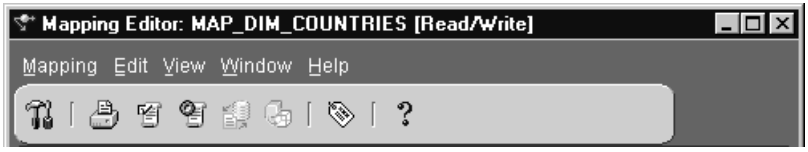
You lock an object when you open the editor, property sheet, or dialog for the object. You acquire a folder-in-use lock when you access an object in a Warehouse Builder hierarchy. This lock prevents other users from deleting a higher object in the hierarchy. For example, if you are editing an object 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.

Figure 2–17 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.
- Exit Warehouse Builder.

Read-Only Mode

If you attempt to open an object that is 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 indicates in the title bar that the object is locked and shows the ID of the user locking the object. 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

Whenever a Warehouse Builder object is locked by a user, other users can view the changes by using synchronize.

Warehouse Builder includes the following synchronization:

- **Automatic Synchronization:** Warehouse Builder automatically synchronizes objects when you access them in read / write mode.

- **Tree Synchronization:** You can synchronize objects by selecting **Synchronize** from the **View** menu. This method displays changes to objects that do not show up with automatic synchronization.

Creating Objects

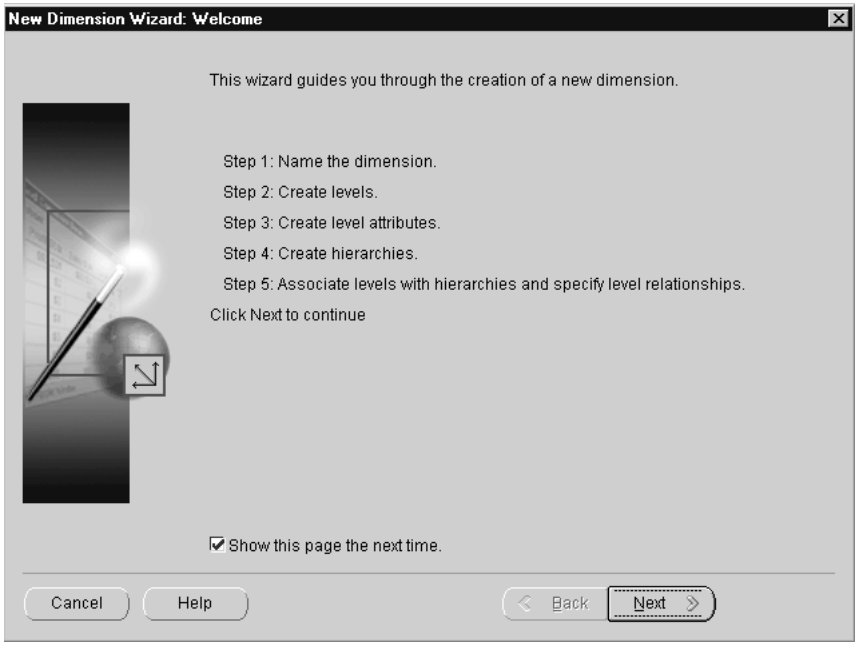
This section describes:

- Using the Warehouse Builder Wizards
- Object Names

Using the Warehouse Builder Wizards

Warehouse Builder provides wizards for creating objects. You open a wizard by right-clicking on an object type and selecting **Create.** Wizards contain pages. The first page is the welcome page. This page lists the number of pages in the wizard and describes the task performed on each page.

Figure 2–18 Wizard Welcome Page



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

To navigate through the pages of a wizard, click the **Back** and **Next** buttons. Click **Cancel** to exit without saving your work. You can move around a page using a 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.

Object Names

Warehouse Builder maintains a logical and a physical name for each object stored in the repository. A logical name is a descriptive business 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 *Oracle8i/9i SQL Reference*.

Because project names are not used in SQL code generation, their names need not conform to the syntax rules for physical names. The length of a project name must not exceed 30 characters.

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:
  - Dimensions
  - Facts
  - Mappings
  - Materialized views
  - Sequences
  - Views
- Transformation names must be unique within a transformation library.

Logical Name Mode

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

A logical name must conform to these rules:

- The length of a name cannot exceed 200 characters.
- The name must be unique within its category.
- All source modules reflect the case of the imported source and are subject to the double-quotes rules as defined in the *Oracle8i/9i SQL Reference*.
- Copy operations from a source to a target in a mapping do not propagate case.

When you create a logical name, Warehouse Builder generates a valid physical name that resembles the logical name. If you create a logical name that duplicates an existing physical name, Warehouse Builder appends an underscore and a number to the logical 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 28 characters. Two characters are used by Warehouse Builder.
- Conform with the basic syntax rules for schema objects defined by the *Oracle8i/9i SQL Reference.*

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 logical name for an object, Warehouse Builder must be in logical 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 logical name.

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

To set the name mode:

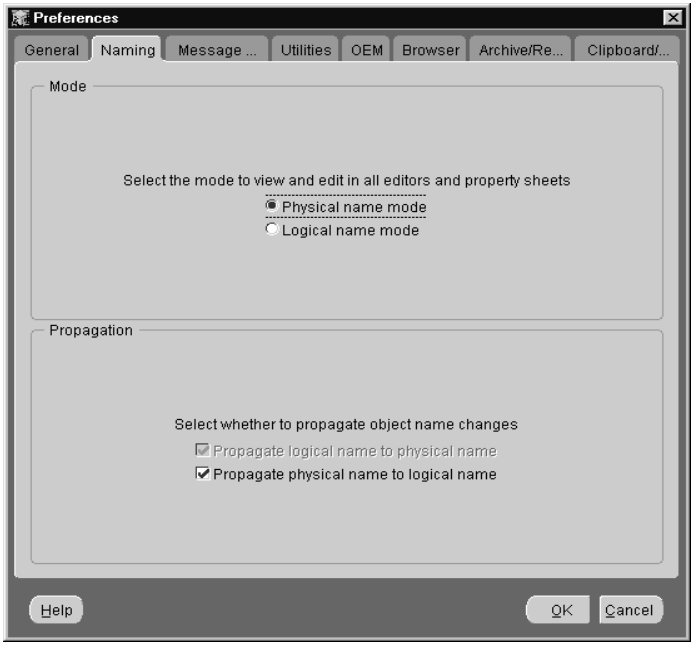
1. From the **Project** menu, select **Preferences.**

   Warehouse Builder displays the Preferences dialog.

2. Select the **Naming** tab.

   Warehouse Builder displays the Naming sheet.

Figure 2–19 Naming Preferences



3. Select physical or logical name mode.

   You can switch the naming mode at any time during a session.

4. Select how the names propagate.
5. 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 will be different.

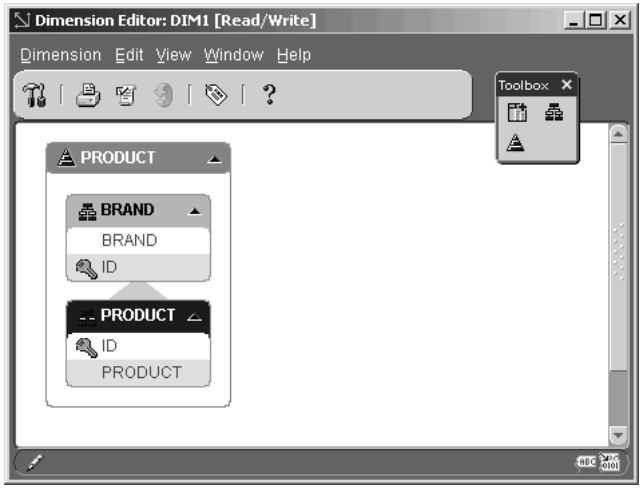
Editing Objects

After you create modules and objects, you can edit them using editors and property sheets.

Object Editors

Object editors provide a separate window with menu items and a toolbox to edit the object. To display an editor, right-click on the object and select **Editor.** Figure 2–20 shows an example of a Dimension Editor.

Figure 2–20 Dimension Editor

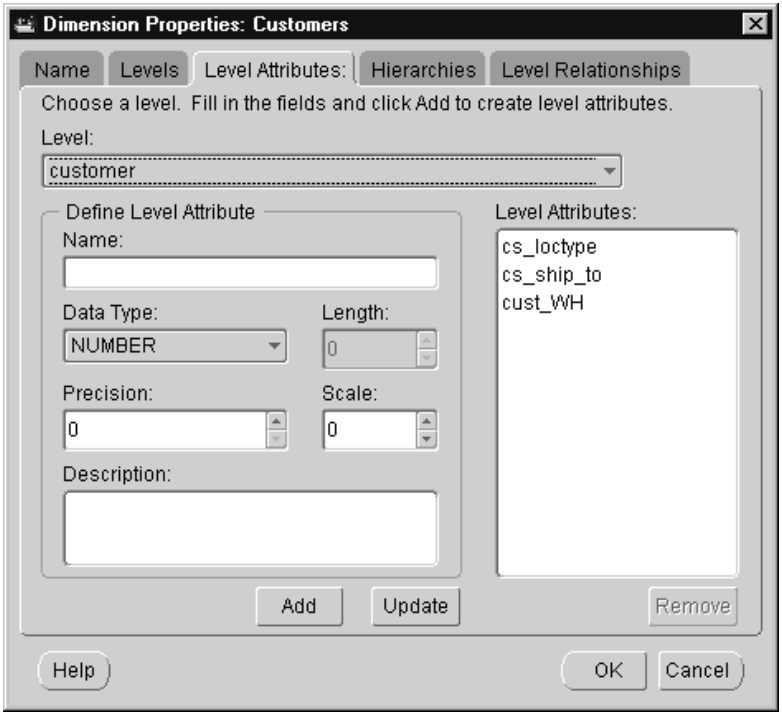


Property Sheets

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. Property sheets vary by object. Figure 2–21 shows an example of a property sheet for a dimension.

Figure 2–21 Property Sheet for a Dimension

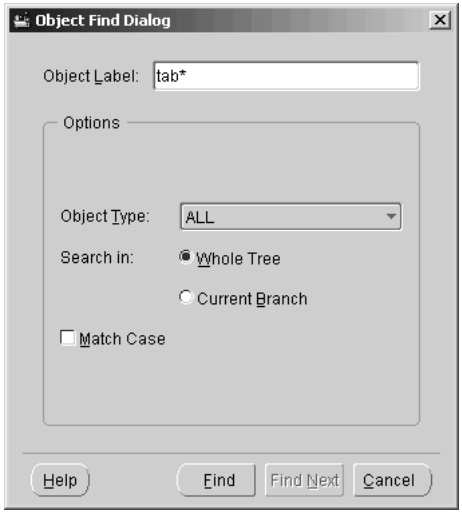


Definitions for objects such as dimensions, tables, facts, and mappings are created using a wizard or imported from an external source. After a definition has been created for an object and stored in a project, you can update the definition by editing its property sheet.

Finding Objects

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 to search for. Use the asterisk (*) as a wildcard character to match zero or more characters.

Figure 2–22 Object Find Dialog



Ending a Warehouse Builder Session

To end a Warehouse Builder session, you commit your work and exit Warehouse Builder.

Committing Your Work

Warehouse Builder does not commit newly created objects or modifications to the repository until you click the Commit icon on the toolbar, or exit Warehouse Builder and click **Yes** on the Commit Confirmation dialog.

You can commit changes at any time during a Warehouse Builder session or when you end your 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.

Exiting Warehouse Builder

To exit Warehouse Builder:

1. Go to the Console window.

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

   If you have not committed your changes when you exit Warehouse Builder, the Commit Confirmation dialog displays. Click **Yes** to commit any changes you have made.

3

Defining Relational Targets

This chapter describes how to define a target module and define relational targets within a module.

This chapter includes the following topics:

- Creating a Warehouse Target Module
- Creating Table Definitions
- Creating Materialized View Definitions
- Creating Conventional View Definitions
- Defining a Sequence Object

Creating a Warehouse Target Module

Warehouse Builder stores definitions for target schemas in target warehouse modules.

To create a target warehouse module:

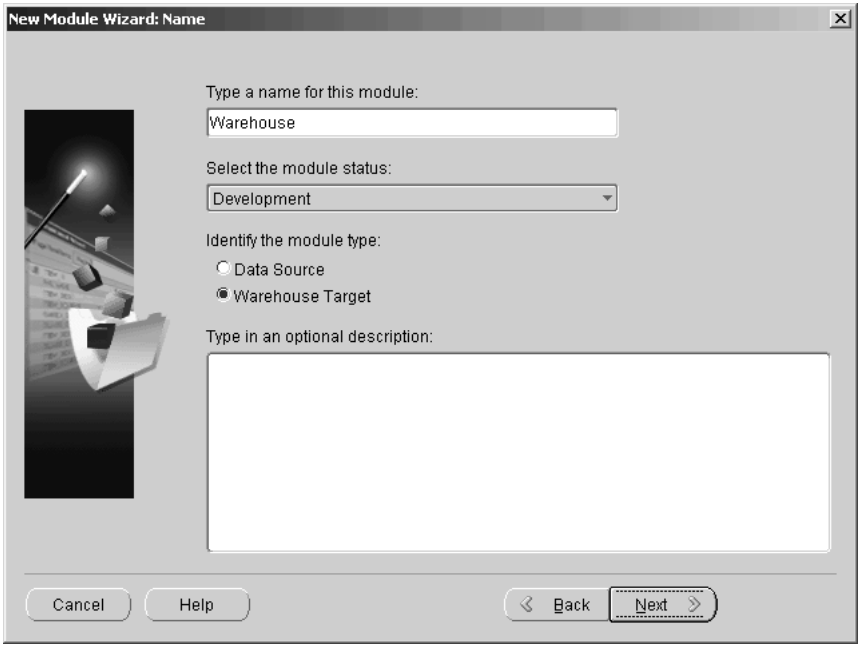
1. In the Warehouse Builder Console, right-click **MODULES** and select **Create Module.**

   Warehouse Builder displays the New Module Wizard welcome page.

2. Click **Next.**

   The wizard displays the Name page.

Figure 3–1 New Module Wizard Name Page



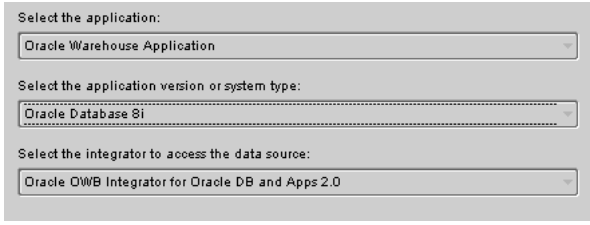
3. Enter the following:
   - A name for the module.
   - The status of the module.

- **Warehouse Target** as the type of module.
- A description of the module.

4. Click **Next.**

   The Target page displays information for Oracle8*i* / 9*i*.

Figure 3–2 New Module Wizard Target Page



5. Click **Next.**

   The wizard displays the Connection Information page.

Figure 3–3 New Module Wizard Connection Information Page

6. If you are not importing any data definitions, click **Next.**

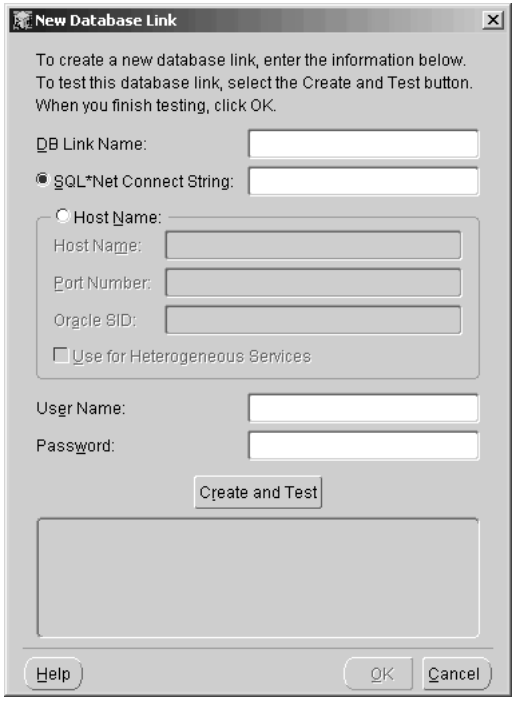
   If you intend to import definitions from a schema or an Oracle Designer repository:

   a. Select Oracle Data Dictionary or Oracle Designer Repository.

   b. If a link to the database has been defined in Warehouse Builder, select the link name from the drop-down list. To define a new link, click **New DB Link.**

      Warehouse Builder displays the New Database Link dialog.

Figure 3–4 New Database Link Dialog



c. Enter the following:

- A name for the new link.
- A SQL*Net connect string or complete host information. The connect string must be defined before you can test the connection.
- A user name and password for the target schema.

d. Click **Create and Test.**

Warehouse Builder connects to the database using the information and displays a status message in the message box below the **Create and Test** button.

Note: A warehouse module can connect only with an Oracle8*i* / 9*i* database instance.

---

e. Click **OK.**

7. Click **Next.**

   Warehouse Builder displays a Finish page summarizing the information you entered. If you need to change any information, navigate back through the wizard pages using the Back button.

   Note: Warehouse Builder displays a warning dialog if the character set of the database instance differs from the character set of the Warehouse Builder host.

8. Click **Finish.**

   Warehouse Builder creates the module in the active project and inserts its name in the MODULES branch of the navigation tree.

Displaying the Warehouse Module

You can view the warehouse module in the Warehouse Module Editor. Use the Warehouse Module Editor to view the structure of a module and create objects within the module.

To open the Warehouse Module Editor:

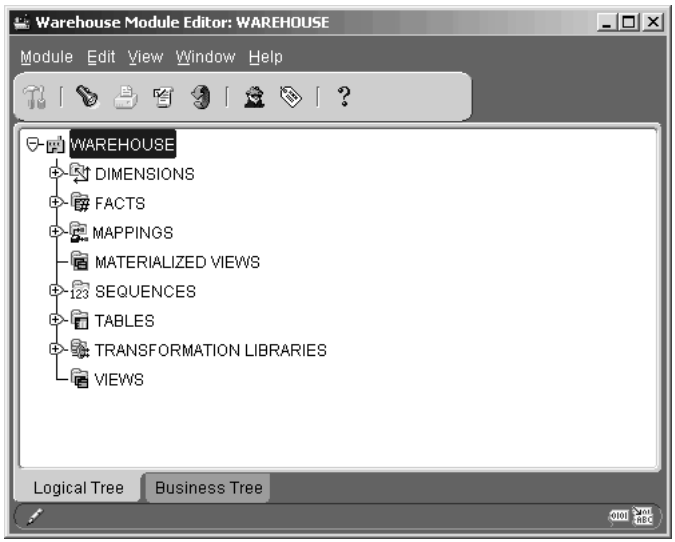
1. Expand the MODULES branch.

   Warehouse Builder lists the project modules.

2. Double-click the warehouse module.

   The Warehouse Module Editor opens displaying the navigation tree.

Figure 3–5 Warehouse Module Editor Navigation Tree



Creating Table Definitions

This section shows you how to create a definition for a table using the New Table Wizard and how to modify the table definition using the property sheet.

Creating a Table Definition

When you create a definition for a table, you enter detailed information about the table into a series of pages in the New Table Wizard.

To create a table definition:

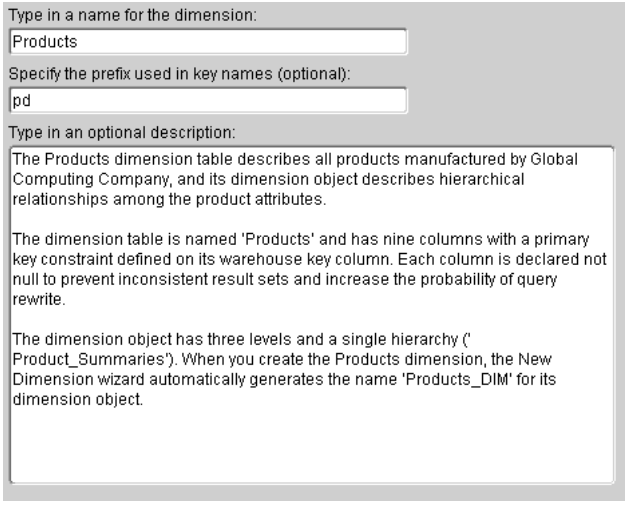
1. In the Warehouse Module Editor, right-click **TABLES** and select **Create Table.**

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

2. Click **Next.**

   The wizard displays the Name dialog.

Figure 3–6 The Name Dialog



3. Enter the following:
   - The name of the table.
   - A description of the table (optional).
4. Click **Next.**

   The wizard displays the Columns page.
5. Define the columns in the table.
   a. Click **Add.**
   b. Select a data type for the column from the drop-down list under Data Type. Warehouse Builder supports the following Oracle8*i*/9*i* data types:
      - CHAR
      - DATE
      - FLOAT
      - NUMBER
      - VARCHAR
      - VARCHAR2

c. Enter the precision, length, or scale as appropriate for the data type.

d. Specify null or not null. The default is null.

e. Type a description of the column in the Note field (optional).

Repeat these steps for each column.

6. Click **Next.**

   The wizard displays the Constraints page. You can define constraints, if necessary.

7. Click **Next.**

   The wizard displays the Finish page. Verify the description.

8. Click **Finish.**

   The wizard creates a definition for the table, stores the definition in the warehouse module, and inserts the name in the navigation tree under TABLES.

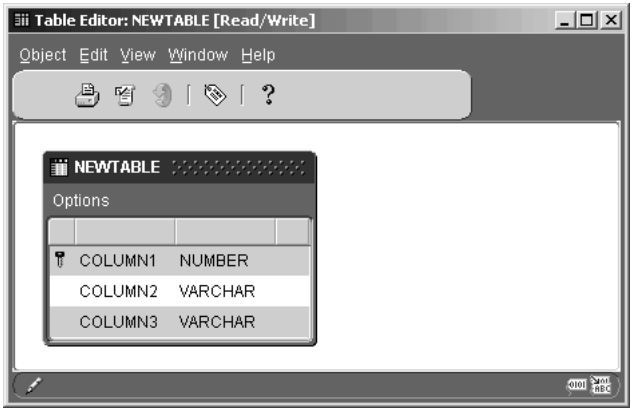
Updating Table Definitions

You can view the definition for a table object in the Table Editor. You use the Table property sheet to edit the table.

Using the Table Editor

To open the Table Editor, right-click the table in the navigation tree and select **Edit.** Warehouse Builder displays the Table Editor.

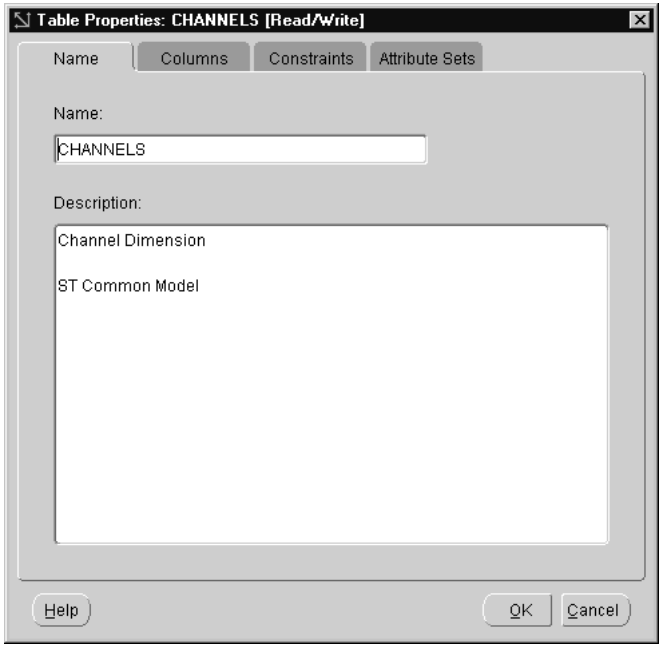
Figure 3–7 Table Editor



Using the Table Property Sheet

You can update the definition for a table by editing entries in the property sheet. In the Warehouse Module Editor, right-click the table and select **Properties.**

Figure 3–8 Table Properties Sheet



The property sheet for the table has the following tabs:

- Name
- Columns
- Constraints
- Attribute Sets

The following examples show you how to order columns within a table, change a UK to a PK constraint, create a check constraint, and create attribute sets.

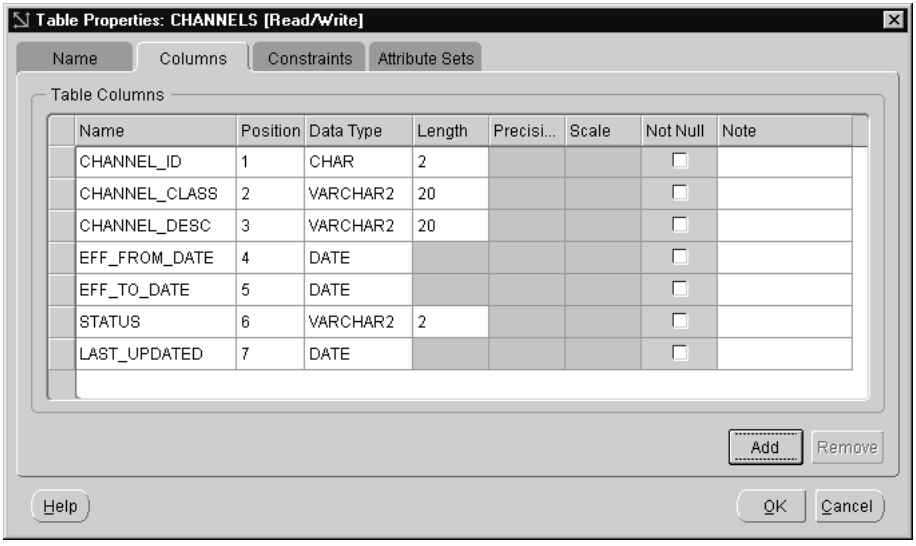
Ordering the Columns By default, the column names within a table are sorted in the order they were created. The order of column names in the property sheet is propagated to the DDL script that Warehouse Builder generates to create the table. The default ordering can cause problems if an application is sensitive to the order of columns. You can order columns so that a generated concatenated key for a fact table corresponds to expected query usage. This ordering can greatly influence query performance.

To order the columns:

1. Open the table Property Sheet.
2. Select the **Columns** tab.

   Warehouse Builder displays the information on the columns in the table.

Figure 3–9 Columns Tab of Table Properties Sheet



3. Select the gray box to the left of the column name and drag the row up or down the list.
4. Click **OK.**

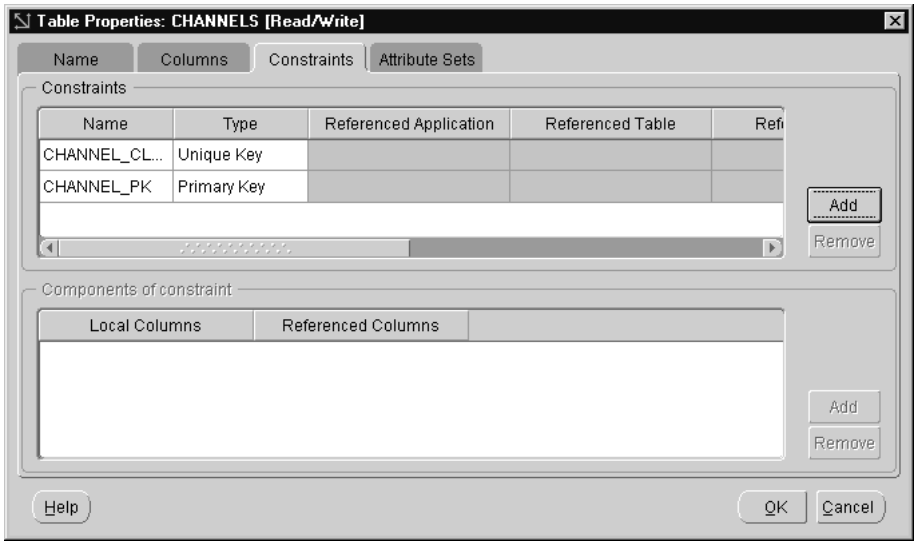
You can order the following warehouse objects:

- Dimensions
- Facts
- Tables
- Materialized views
- Conventional views

Changing a Constraint Warehouse Builder automatically generates a UK constraint for each Level defined within a dimension table. The generated constraint is always on

the column defined as the Level ID column, and except for the lowest-level, these constraints are purely logical constraints that are not used in the DDL to create the table. You can edit these constraints but you cannot remove them. If you attempt to remove one of these constraints, Warehouse Builder displays an error message.

Figure 3–10 Generated UK Constraints



When you declare a PK or UK constraint, you cannot later change it to a FK or CK constraint. You must drop the constraint and create a new one. Similarly, if you create an FK constraint, you cannot change it to any other kind of constraint. You must drop the constraint and create a new one.

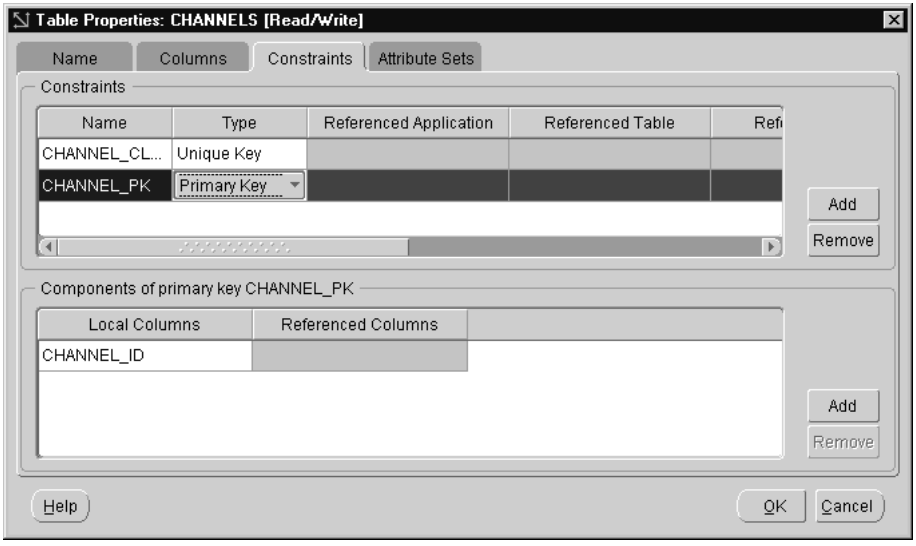
For a dimension table, Warehouse Builder automatically generates a UK rather than a PK constraint on the column that defines the lowest level of aggregation.

To change a constraint type:

1. Open the table property sheet and select the **Constraints** tab.

   Warehouse Builder displays the information on all constraints defined in the table.

Figure 3–11 Constraints Tab of Table Properties Sheet



2. Click the drop-down arrow under the Type column and select Primary Key.
3. Change UK in the constraint name to PK.

   The editor only allows you to select a UK or PK constraint.
4. Change the name of the constraint to reflect this change.
5. Click **OK.**

   Warehouse Builder updates the definition of the underlying table.

> **Note:** You cannot remove a generated UK constraint or column for a dimension Level ID.

Adding Check Constraints You can add a check constraint to any table column. These constraints enforce business rules on values stored in a dimension, fact, or table. They can also be used to define a not null constraint. However, using the Add Check Constraints feature can slow your load performance.

To add a check constraint:

1. Open the table property sheet and select the **Constraints** tab.

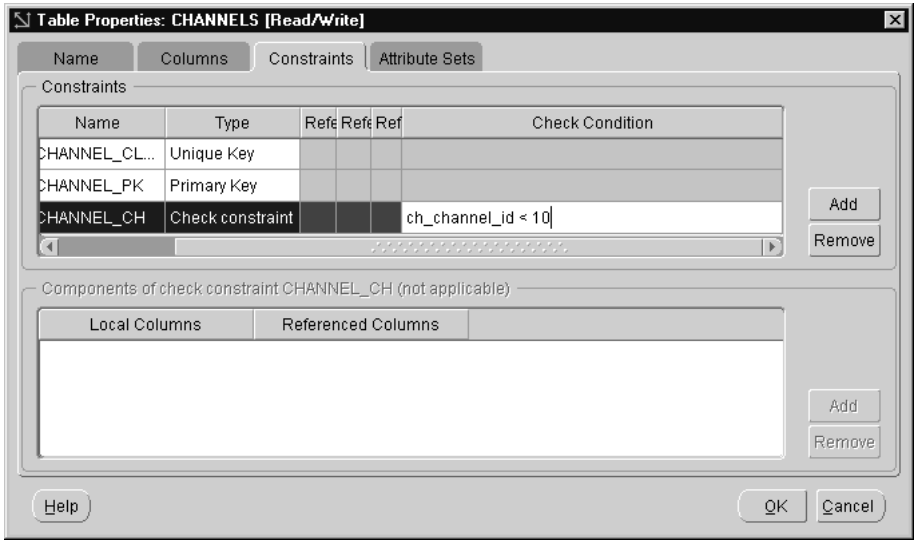
Warehouse Builder displays information on all the constraints defined in the table.

2. Click **Add.**

   Warehouse Builder inserts a row in the top pane.

3. Enter a name for the constraint, select Check Constraint, and enter a condition.

Figure 3–12 Constraints Tab of Table Properties Sheet



4. Click **OK.**

   Warehouse Builder stores the CHECK constraint in the property sheet.

   A CHECK integrity constraint requires that a condition be true or unknown for every row of the table. If a statement causes the condition to evaluate to false, then the statement is rolled back.

   The condition of a CHECK constraint has the following limitations:

   - The condition must be a Boolean expression that can be evaluated using the values in the row being inserted or updated.
   - The condition cannot contain subqueries or sequences.
   - The condition cannot include the SYSDATE, UID, USER, or USERENV SQL functions.

- The condition cannot contain the pseudocolumns LEVEL, PRIOR, or ROWNUM.

For additional information on CHECK constraints, see the *Oracle8i* / 9*i Application Developer's Guide - Fundamentals.*

Notes:

- The column name referenced in the condition of the CHECK constraint must exactly match a physical name defined for the table in its property sheet.
- Warehouse Builder does not check the syntax of the condition. This can result in problems during the code generation phase when Warehouse Builder generates a create table statement to deploy the table.

Creating Attribute Sets

Columns in a table are called attributes. Each table in the warehouse module has a predefined attribute set consisting of all the table columns and another attribute set for each constraint. You can create additional named attribute sets containing any number of the table columns in the order you specify.

You can add the following types of attribute sets:

- **User-defined**: Optional attribute set that you can create, modify, or delete in the Table Properties sheet.
- **Bridge-type**: Optional attribute set that can be transferred across a bridge for viewing in another software program. You can create, modify, or delete a bridge-type attribute set in the Table Properties sheet. A table can have only one bridge-type attribute set.

Attribute sets are used in mappings.

To add an attribute set:

1. Open the table property sheet and select the **Attribute Sets** tab.

   Warehouse Builder displays information on the attribute sets defined in the table.

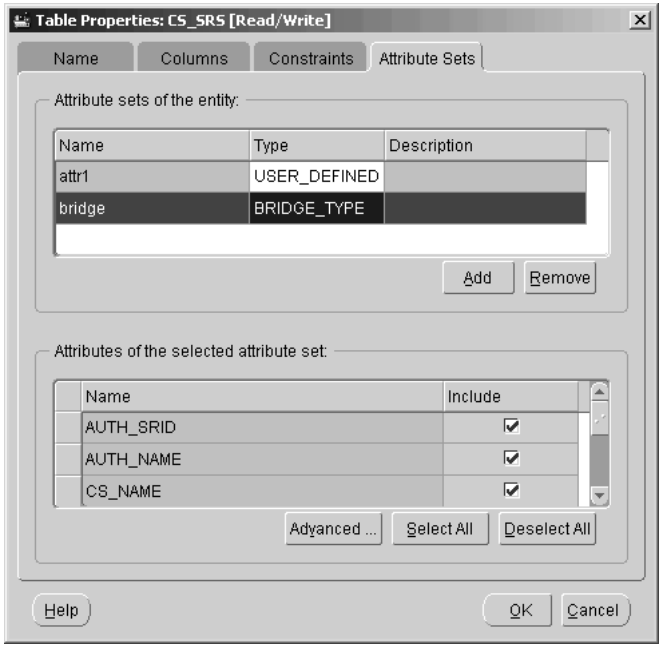
2. Click **Add.**

   Warehouse Builder inserts a row in the top pane.

3. Type a name for the attribute set under the Name column.
4. Select USER_DEFINED or BRIDGE_TYPE from the drop-down list.
5. Type a description.
6. Click the check box for each attribute that you want to include.

   The order in which you select the columns determines their initial order in the attribute set.

Figure 3–13 Table Properties

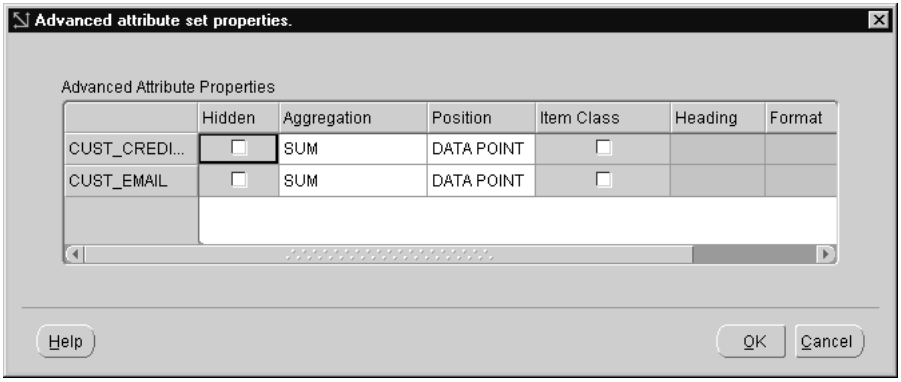


> **Note:** Changing the order of non-selected attributes has no effect after you close the Table Properties sheet.

7. If you selected BRIDGE_TYPE, click **Advanced** to specify additional properties of the attribute set.

   Warehouse Builder displays the Advanced Attribute Set Properties dialog.

Figure 3–14 Advanced Attribute Set Properties Dialog



a. For each attribute in the bridge-type attribute set, specify the properties:

   - **Hidden:** Click the check box to hide the column. In the Discoverer Administration Edition, hidden columns are greyed out. In the Discoverer User Edition, hidden columns are not shown.
   - **Aggregation:** Select SUM, MIN, MAX, AVG, COUNT, or DETAIL for no aggregation. The default is SUM.
   - **Position:** Select 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:** Enter the heading text.
   - **Format:** Enter the text for the format field.

b. Click **OK.**

8. Click **OK** to close the Table Properties dialog.

Importing Definitions

You can import definitions for tables, Views and Sequences into a warehouse module using the Import Metadata Wizard. For instructions, see "Importing Definitions from a Database" on page 5-19.

Creating Materialized View Definitions

This section shows you how to create and update definitions for materialized views. You create a definition for a materialized view using the New Materialized View Wizard. You modify a view definition by editing a property sheet. In the New Materialized View Wizard, you enter:

- A name and description
- Aliases for columns
- Queries that define the view
- Constraints (only logical)

Creating a Materialized View Definition

When you create a definition for a materialized view, you enter detailed information about it into a series of pages for the New Materialized View Wizard.

To create a materialized view definition:

1. In the Warehouse Module Editor, right-click **Materialized View** and select **Create Materialized View.**

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

2. Click **Next.**

   The wizard displays the Name page.

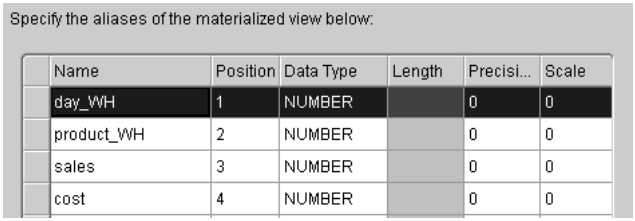
3. Enter the following:

   - The name of the view
   - A description of the view (optional)

4. Click **Next.**

   The wizard displays the Columns page.

Figure 3–15 Columns Page



To define a column:

a. Click **Add.**

b. Type the column name.

c. Select the data type.

Repeat this procedure for each column.

5. Click **Next.**

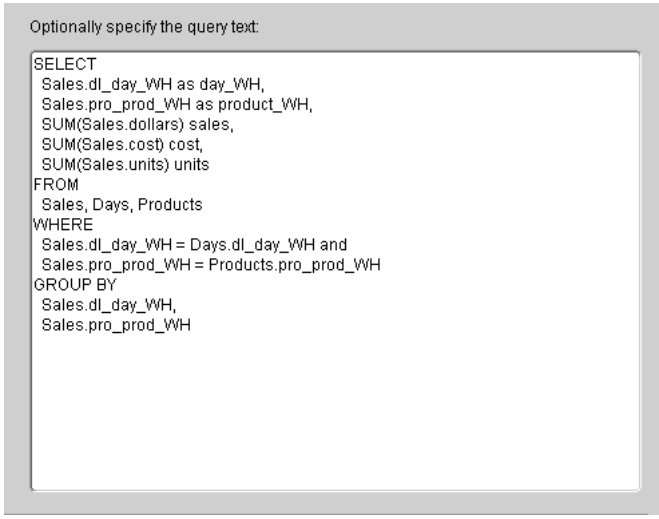
   The wizard displays the Query Text page.

6. Do one of the following:

   - Enter text for a query that defines the view.
   - Leave the Query Text box empty. Later, create a mapping that defines the necessary query. See Chapter 7, "Using Mapping Operators and Transformations" for information.

   Figure 3–16 shows sample query text.

Figure 3–16 Sample Query Text



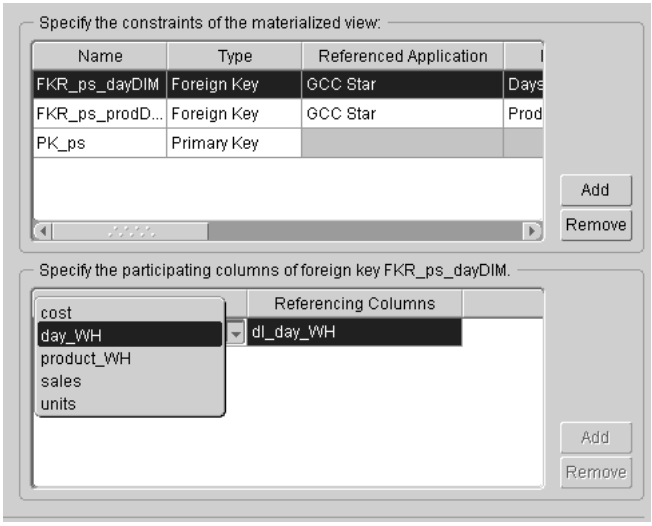
Note:

- Warehouse Builder generates code for a view only if query text is entered and columns are defined.
- Warehouse Builder generates a Create Materialized View statement to deploy the view even if its syntax is invalid. Warehouse Builder does not check the syntax of the select statement used to define a view.
- Entered queries are automatically completed with a semi-colon. Adding one will cause a syntax error.

7. Click **Next.**

   The wizard displays the Define Constraints page.

Figure 3–17 Define Constraints Page



8. Define the key constraints:

   a. Click **Add** next to the upper box and enter the constraint name.

   b. Select the constraint type.

   c. Select the constraints:

      – Referenced application

      – Referenced table

      – Referenced key

   d. Click **Add** next to the lower text box. Select the columns local to the view that are in the constraint.

   Repeat these steps to define the other constraints.

9. Click **Next.**

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

10. Click **Finish.**

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

tree. Confirm this by fully expanding the editor navigation tree. The name of the materialized view now occurs under the MATERIALIZED VIEWS subtree.

Updating a Materialized View Definition

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 **Edit.** 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 "Updating Table Definitions" on page 3-9.

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

Creating Conventional View Definitions

This section describes how to create and update definitions for conventional views. You create a definition for a conventional view using the New View Wizard. You update the view definition by editing its property sheet.

Creating a View Definition

Although this view has similar structure to a materialized view, the views differ as follows:

- **Function:** A view restricts access to a single family of products. A materialized view makes queries run faster.
- **Visibility:** A view is visible to a class of users. A materialized view is transparent to users.
- **Physical Storage:** A view occupies no storage space. A materialized view does.

To create a view definition:

1. Open the warehouse module and expand its navigation tree.
2. Right-click **View** and select **Create View.**

   Warehouse Builder displays the welcome page for the wizard.

3. Click **Next.**

   The wizard displays the Name page.

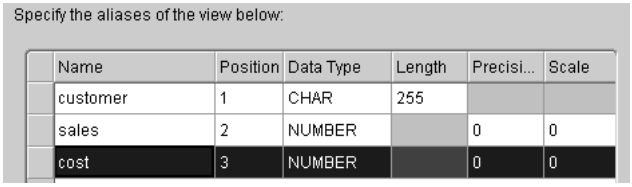
4. Enter the following:

   - The name of the view
   - A description of the view (optional)

5. Click **Next.**

   The wizard displays the Columns page.

Figure 3–18 Columns Page



6. Define the columns:

   a. Click **Add.**

   b. Type the column name.

   c. Select the data type.

   Repeat this procedure for each column.

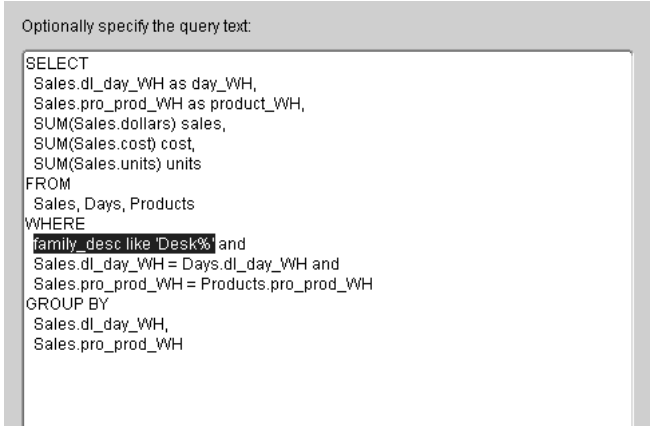
7. Click **Next.**

   The wizard displays the Query Text page. When you create the definition for a view, you can:

   - Enter the text for a query that defines the view in the Query Text box.
   - Leave the Query Text box empty. Later, create a mapping that defines the necessary query. See Chapter 7, "Using Mapping Operators and Transformations" for information.

   Figure 3–19 shows sample query text.

Figure 3–19 Query Text



Note:

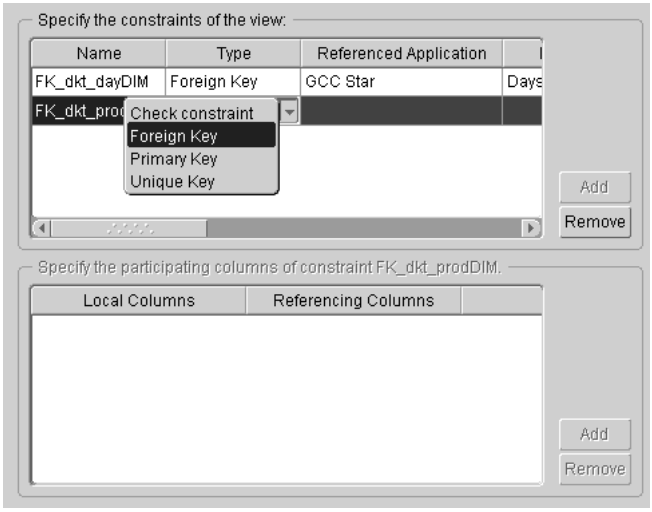
- Warehouse Builder does not generate code for a view if its query text is not included in its Property Sheet or if it has no columns defined.
- Warehouse Builder generates a Create View statement to deploy the view even if its syntax is invalid. Warehouse Builder does not check the syntax of the select statement used to define a view.

8. Click **Next.**

The wizard displays the Define Constraints page.

Use this page to define logical constraints for a 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.

Figure 3–20 Define Constraints Page



Define foreign key reference constraints as described in step 8 in "Creating a Materialized View Definition".

9. Click **Next.**

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

10. Click **Finish.**

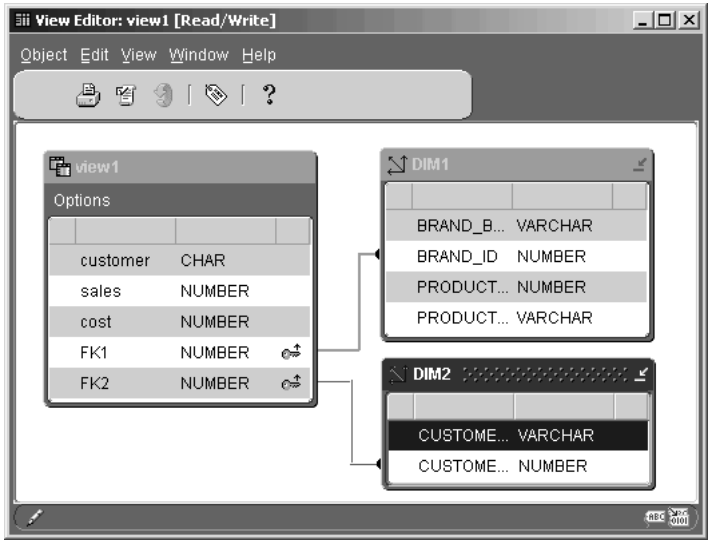
    Warehouse Builder creates a definition for the view, stores the definition in the warehouse module, and inserts its name in the warehouse module navigation tree under VIEWS.

Updating a View Definition

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.

Figure 3–21 View Editor



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 "Updating Table Definitions" on page 3-9.

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

Defining a Sequence Object

A sequence object populates the warehouse key column for a dimension. To define a sequence object, you use the New Sequence Wizard. You configure the sequence settings when you use the sequence in a mapping. See "Adding Mapping Sequences" on page 7-5 for more information.

To define a sequence object:

1. Right-click **SEQUENCE** and select **Create Sequence.**
2. Click **Next.**
3. Enter a name and description for the sequence in the appropriate fields and click **Next.**
4. Click **Finish.**

4

Defining Dimensional Targets

This chapter describes how to define dimensional objects within a warehouse target module.

This chapter includes the following topics:

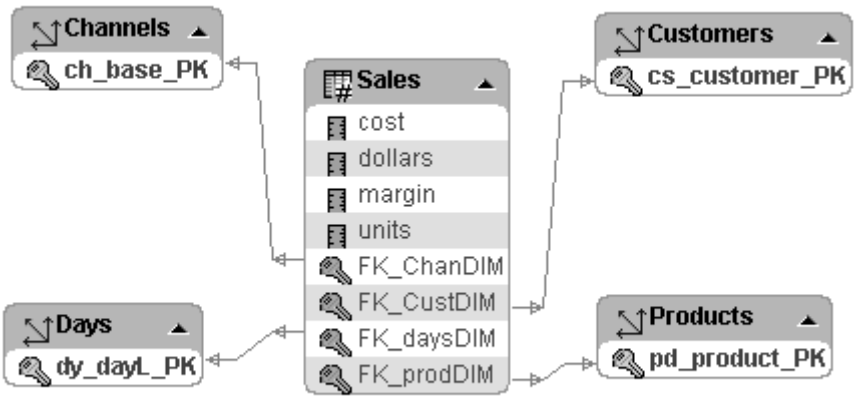
- About Star Schemas
- Creating Dimension Definitions
- Creating Fact Table Definitions

About Star Schemas

A typical dimensional model used as the target schema for a data warehouse contains a fact table and dimensions. This is known as a star schema. It can also contain materialized views and a conventional view.

Figure 4–1 shows a fact table linked to a set of dimensions over foreign key constraints.

Figure 4–1 Sample Star Schema



For information on star schemas and dimensional models, see the *Oracle8i/9i Data Warehousing Guide.*

You should create schema objects in the following order:

1. Dimensions
2. A fact table
3. Views

Dimensions are created first and objects that reference the dimensions second. You can create the definition for the fact table first by omitting the foreign key reference columns and adding them later after the dimensions have been defined. When a fact table relies on the server to verify foreign key references, this same order must be followed.

About Dimension Tables

Dimension tables contain descriptive details that give meaning to the numbers that populate fact tables.

Query performance can be improved because users often analyze data by drilling down on known hierarchies. Examples of hierarchies are a temporal hierarchy of Year, Quarter, Month, Day and a brand hierarchy of Category, Brand, Product. Oracle8*i*/9*i* makes use of defined hierarchies by rewriting queries to retrieve data from summary rather than detail tables. The rewritten queries run much faster.

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 fact tables.
- 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 Oracle8*i*/9*i* server.

About Fact Tables

Fact tables contain measures and link to one or more dimension tables. Most fact table measures are additive. Common additive measures include dollars, units, and cost. Fact tables can include non-additive and semi-additive measures such as margins, averages, and balances. Event fact tables do not contain measures.

Fact tables 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. Referential integrity constraints can cause the data loading to slow down because they require an enormous number of look-up operations.

When dimensions are designed with warehouse keys, the fact table row length is usually reduced because warehouse keys are shorter than their natural counterparts. The result is improved query performance.

A typical fact table 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 fact table is a data list, the foreign key reference columns do not uniquely identify each row in the fact table.

- A set of foreign key reference columns that link the table with its dimensions.

When you create a definition for a fact table, 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.

About Materialized Views

Materialized views are summary tables used by the Oracle8*i/9i* server to rewrite queries so they execute faster. A query that retrieves product totals from a monthly summary table will run much faster than a query that retrieves data from a detail table and then calculates the monthly subtotals. Because the first query retrieves far fewer rows and performs fewer calculations, it executes much faster.

End users do not know whether the materialized views exist or how to rewrite queries. The server automatically does this work. The Oracle8*i/9i* server can also refresh materialized views as the detail data changes.

When you create materialized views, you can enable query rewrite. To maximize the number of query rewrites, create dimension and fact tables so they include the physical characteristics of dimensions and facts described previously. See the *Oracle8i/9i Data Warehousing Guide* for more information about query rewrites.

About Conventional Views

Conventional views simplify and secure data within a data warehouse without requiring additional physical storage space. For example, a view could calculate and return standard deviations for brand summaries rather than a list of details. This simplifies access for users. Another view could restrict access to the sales values of specific products. This type of view simplifies management of the data warehouses for the database administrator.

Although the procedure for creating a definition for a view is similar to creating a materialized view, the objects themselves are different. A materialized view occupies storage space; a view does not. A materialized view speeds queries; a view has a different purpose. Materialized views are designed for the server and users do not know about their existence; views are displayed to users.

Creating Dimension Definitions

This section describes how to create dimensions using the New Dimension Wizard. Use the New Time Dimension Wizard to create a time dimension. This section also describes how to modify the definitions using property sheets.

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 4–1 summarizes the rules for dimensions.

Table 4–1 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's value. For example, a city determines a 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 generates 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 > DayL

H2: YearL > 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.

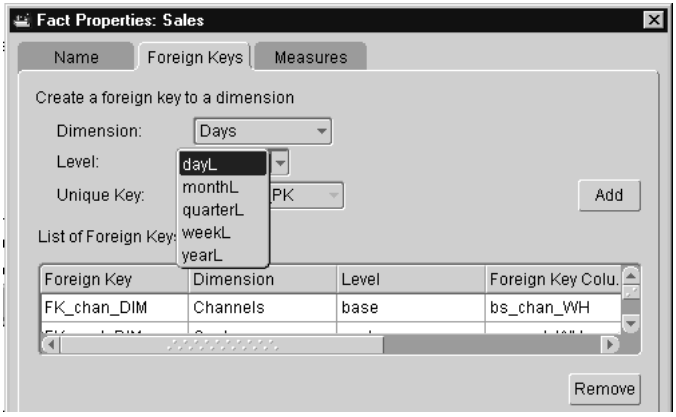
About Unique Key Constraints

When you create a definition for a hierarchy, Warehouse Builder creates an identifier key for each level of the hierarchy and a unique key constraint on the

lowest level. Warehouse Builder uses the identifier keys during the generation phase to build a DDL script to create the dimension object.

When you create a foreign key reference constraint on a fact table that points to a dimension, Warehouse Builder shows the unique key constraint and the other identifier keys as candidates for the referenced column. A fact table 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 4–2 Fact Properties Dialog Showing the Foreign Keys Tab



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 > WeekL > 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 columns.

Creating a Dimension Definition

To create definitions for a dimension, use the New Dimension Wizard. You name the dimension table and define a primary key constraint on its warehouse key column. 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 4–2 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 4–2 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. Right-click **Dimensions** and select **Create Dimension.**

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

2. Click **Next.**

   The wizard displays the Name page.

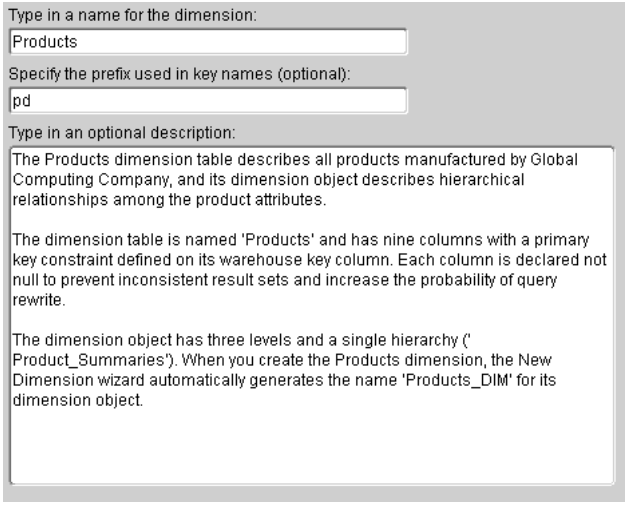
3. Type the following:

   - 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 table name is used.

   - A description of the dimension (optional).

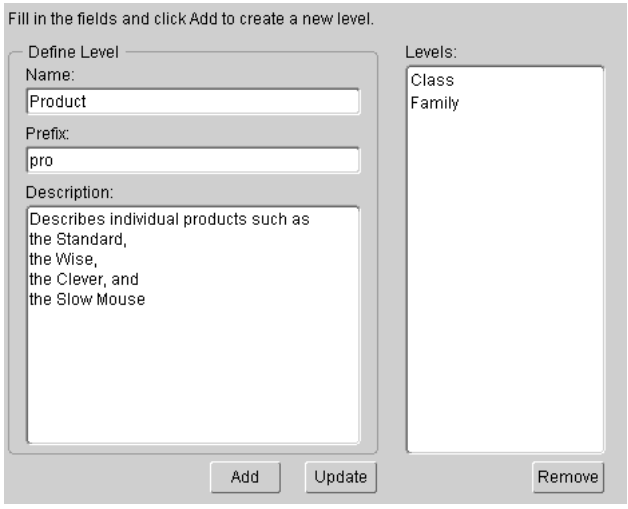
Figure 4–3 Name Page



4. Click **Next.**

   The wizard displays the Levels page.

Figure 4–4 Levels Page



> **Note:** Dimensions contain at least one level of aggregation. You can define a default level of aggregation to satisfy this requirement and include additional levels as required.

5. 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.
6. 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 automatically 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 fact foreign keys may already refer to the generated level. Unique key names generated after you edit the prefix use the new prefix.

7. 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 level's logical key column. This attribute is used in the CREATE DIMENSION statement to define the level, and the defined level is used in the statement's DETERMINES clause to specify other columns within that level (dependent columns). See the *Oracle8i/9i SQL Reference* and the *Oracle8i/9i Data Warehousing Guide* for more information.

8. Provide the following information:

   a. Select a level of aggregation from the drop-down list.

   b. Type a name for the attribute.

   c. Select a data type for the attribute from the drop-down list under Data Type. Warehouse Builder supports the following Oracle8*i*/9*i* data types:

      – CHAR

      – DATE

      – FLOAT

      – NUMBER

      – VARCHAR

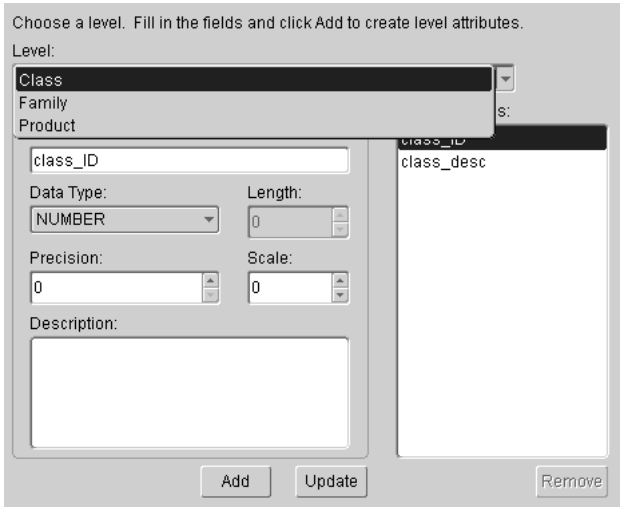
      – VARCHAR2

   d. Enter the precision, length, or scale as appropriate for the data type.

   e. Type a description of the attribute.

9. Click **Add.**

Figure 4–5 Level Attributes Page



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.

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

a. Type a new name in the Name text box.

b. Click **Update.**

10. Click **Next.**

The wizard displays the Hierarchies page.

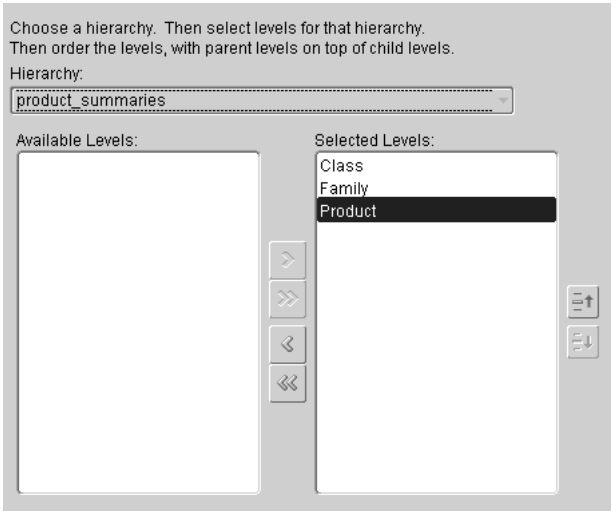
11. Define the hierarchy:

- Type a name and prefix for each hierarchy.
- Type a description of the hierarchy.

12. Click **Next.**

The wizard displays the Level Relationships page.

Figure 4–6 Level Relationships Page



13. 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.
14. Click **Next.**

    The wizard displays the Finish page. Verify the description.
15. 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 dimension's base level of aggregation. 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. See "Changing Key Constraints" on page 4-25 for information.

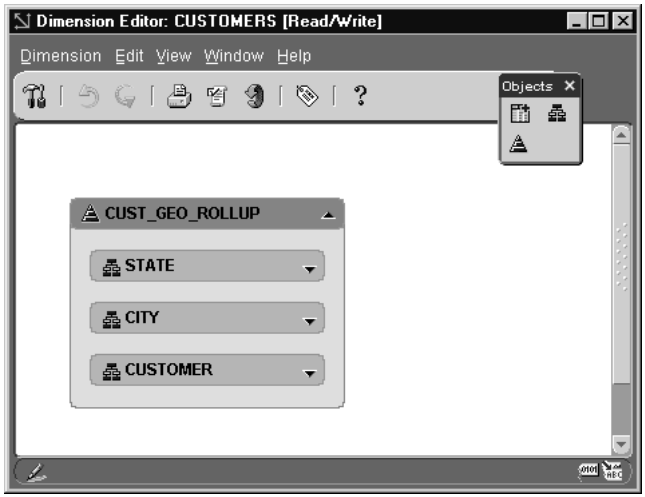
Updating Dimension Definitions

You can update 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 **Edit.**

Figure 4–7 Dimension Editor



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:

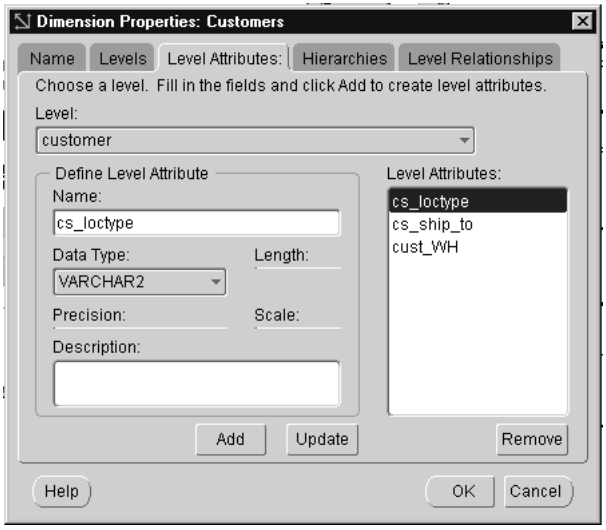
1. Fully expand the Level where you want to add the new attribute.
2. Drop the Attribute icon on the Level.

   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.
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 4–8 Level Attributes Tab of Dimension Properties Sheet



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

Using the Property Sheets

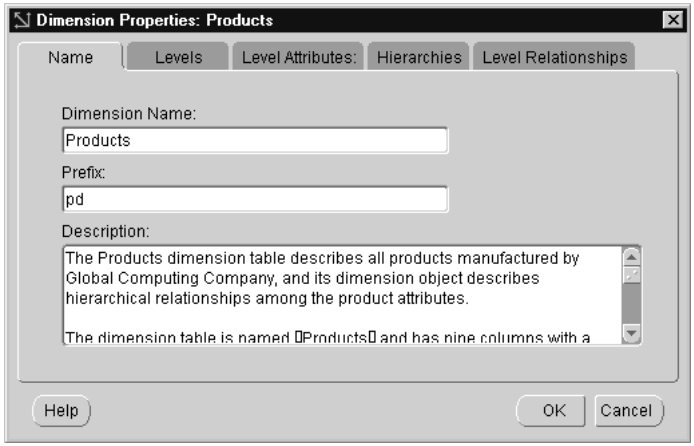
The dimension object and the dimension table both have property sheets. In the dimension object property sheet, you edit the levels and hierarchies. In the dimension table property sheet, you edit the columns and constraints.

To display the dimension object property sheet:

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

Warehouse Builder displays the property sheet for the dimension object.

Figure 4–9 Dimension Object Properties Sheet



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 "Using the Table Property Sheet" on page 3-10.

Creating a Time Dimension Definition

This section describes how to use the New Time Dimension Wizard. The Time Dimension Wizard generates a SQL insert statement to populate the time dimension. You do not need to extract data from a data source. You must set the start and end dates for the statement that generates the data. See "Guidelines for

Configuring Dimensions, Facts, and Tables" on page 9-6 for information on configuring the dates.

The Time Dimension Wizard uses predefined names and prefixes for levels, hierarchies, and attributes. The attributes have predefined data types. After you create the definition, you can update the property sheet.

Note: Time dimensions differ considerably and many designs depend on multiple time dimensions. The New Time Dimension Wizard covers a limited number of cases. If the wizard does not meet your needs, use the New Dimension Wizard instead.

To create a time dimension definition:

1. In the Warehouse Module Editor, right-click **Dimensions** and select **Create Time Dimension.**

   Warehouse Builder displays the welcome page for the New Time Dimension Wizard.

2. Click **Next.**

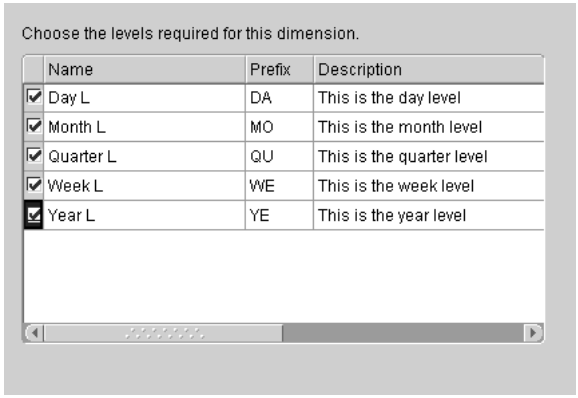
3. Type the following:

   - A name for the dimension.
   - A prefix.
   - A description of the dimension.

4. Click **Next.**

   The Levels page contains predefined levels of aggregation and a prefix for each level.

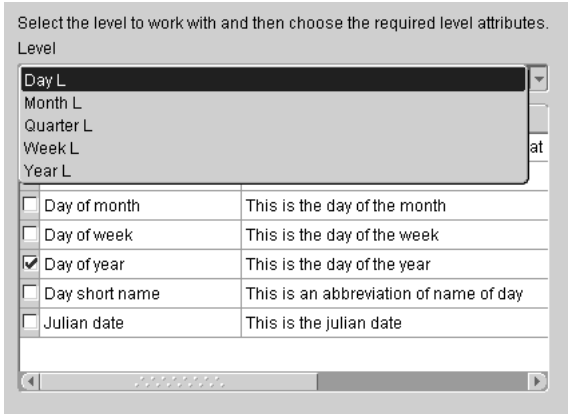
Figure 4–10 Levels Page



5. Check the levels of aggregation required to support your dimensional model.
6. Click **Next.**

   The wizard displays the Level Attributes page. This page contains a set of predefined attributes for each level.

Figure 4–11 Level Attributes Page



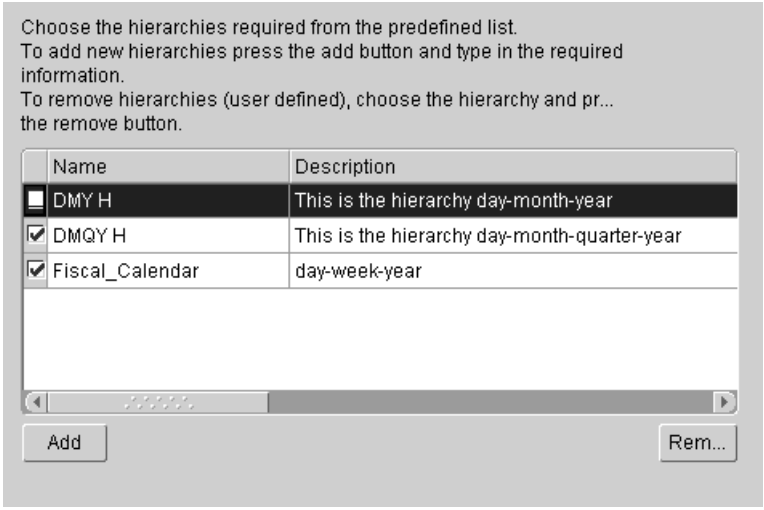
- Select a level of aggregation from the drop-down list.
- Check the required attributes for that level.

Continue this process to select attribute sets for each level.

7. Click **Next.**

   The Hierarchies page contains a set of predefined attributes for each level.

Figure 4–12 Hierarchies Page



8. Check the predefined hierarchies required by the time dimension.
9. Add custom hierarchies required by the time dimension:
   a. Click **Add.**
   b. Name the hierarchy.
   c. Describe the hierarchy.
10. Click **Next.**

    The wizard displays the Level Relationships page.

11. Define the levels for each custom hierarchy by moving the level from Available Levels to Selected Levels. The wizard automatically orders the levels.
12. Click **Next.**

    The wizard displays the Finish page.

13. Click **Finish.**

    The wizard stores the definition in the warehouse module.

The New Time Dimension Wizard generates the attributes you select plus additional attributes and constraints. In addition to the selected attributes, the wizard generates:

- An ID attribute for each level.
- A Smart_key attribute for the base level.
- A UK constraint on the ID attribute for each level, which you can modify but not remove. See "Changing a Constraint" on page 3-12.
- A PK constraint on the Smart_key attribute.

You can edit the names of constraints, levels, attributes, and hierarchies in the dimension property sheet. For additional information on editing a dimension object, see "Updating Dimension Definitions" on page 4-12.

Creating Fact Table Definitions

This section describes how to create and update a definition for a fact table. You create a definition for a fact table using the New Fact Table 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.

You use the New Fact Table Wizard to create definitions for a fact table. This information includes details regarding foreign key references, measures, and the data types of all the table's columns.

Creating a Definition for a Fact Table

This section describes how to create a fact table definition.

To create a fact table definition:

1. Right-click **FACTS** and select **Create Fact.**

   Warehouse Builder displays the Welcome page for the New Fact Wizard.

2. Click **Next.**

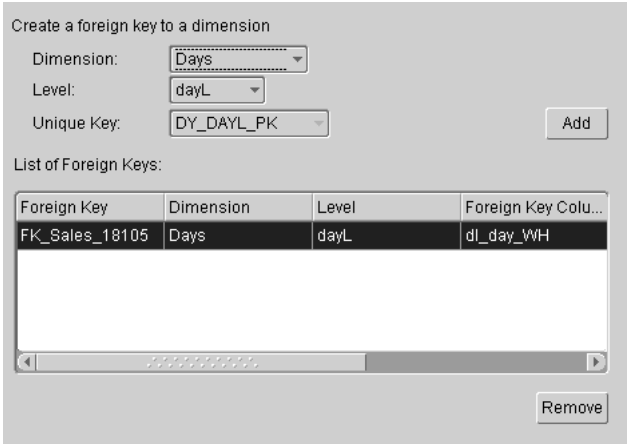
   The wizard displays the Name page.

3. Enter the following:
   - The name of the fact table
   - A description of the fact table (optional)

4. Click **Next.**

   The wizard displays the Define Foreign Keys page.

Figure 4–13 Define Foreign Keys Page



5. Define the foreign key references:

   a. Select the name of a dimension from the Dimension drop-down list.

   b. Select the base key level from the Level drop-down list.

   c. Select the primary key column constraint defined on the dimension from the Unique Key drop-down list.

   d. Click **Add.**

      The wizard inserts the foreign key reference constraint in the text box that lists the foreign keys.

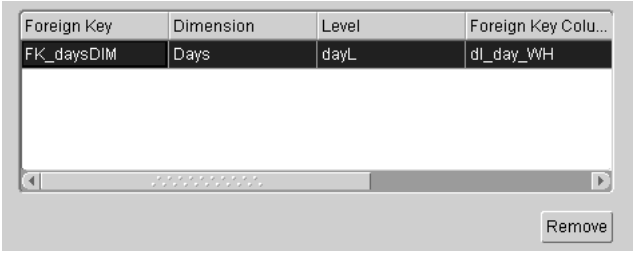
   e. Repeat these steps for each foreign key constraint. Select the lowest level of aggregation for the foreign key reference target.

Note:

- The Fact Wizard displays the name for each generated PK constraint on a dimension's 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 fact table foreign key references.

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.

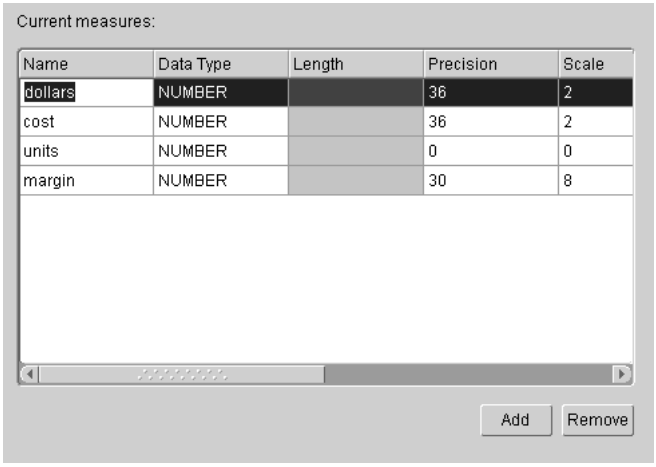
Figure 4–14 Foreign Keys



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

   The wizard displays the Define Measures page.

Figure 4–15 Define Measures Page



8. Define the measures for the fact table:
   a. Click **Add.**
   b. Type the name of the measure.
   c. Select the data type of the measure.
   d. Repeat these steps for each measure in the fact table.
9. Click **Next.**

   The wizard displays the Finish page. This page summarizes the fact table. Click **Back** to modify any of the elements.
10. Click **Finish.**

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

Updating a Fact Table Definition

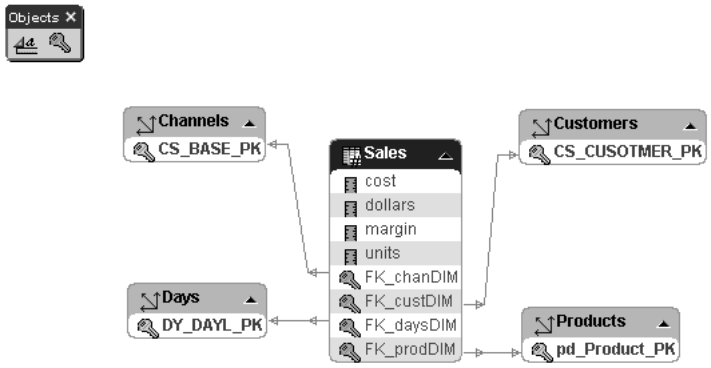
A fact object has two property sheets: one for the fact object and another for the fact table. You can update a fact object's properties by editing the property sheets. In addition, you can add foreign key references or measures to a fact object using the Fact Editor. You can also use the Fact Editor to change fact properties and foreign key relationships with dimensions.

Using the Fact Editor

To open the Fact Editor, right-click a fact table in the Warehouse Module Editor and select **Edit** from the pop-up menu.

Warehouse Builder displays the Fact Editor containing a tool palette and a diagram of the fact table and the related dimensions.

Figure 4–16 Fact Editor



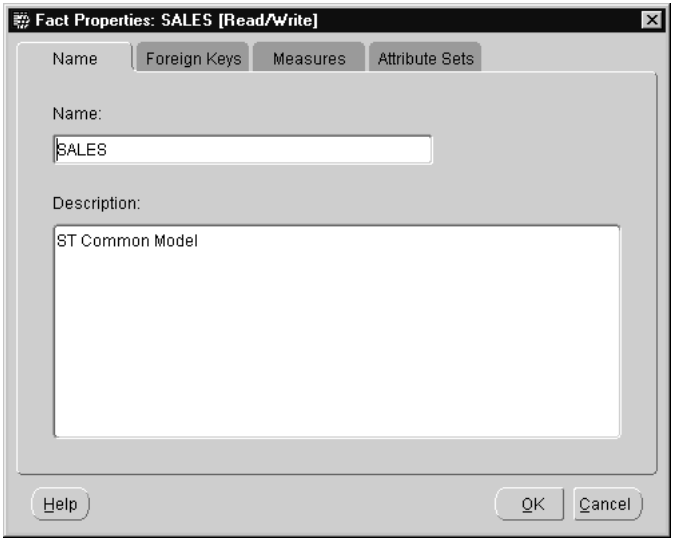
To print the diagram, click the printer icon on the Fact Editor toolbar.

To display the fact object property sheet:

- From the **Fact** menu, select **Fact Properties** or click the **Properties** icon.
- Right-click the fact and select **Properties.**

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

Figure 4–17 Fact Properties Sheet



From the Fact Properties sheets, you can:

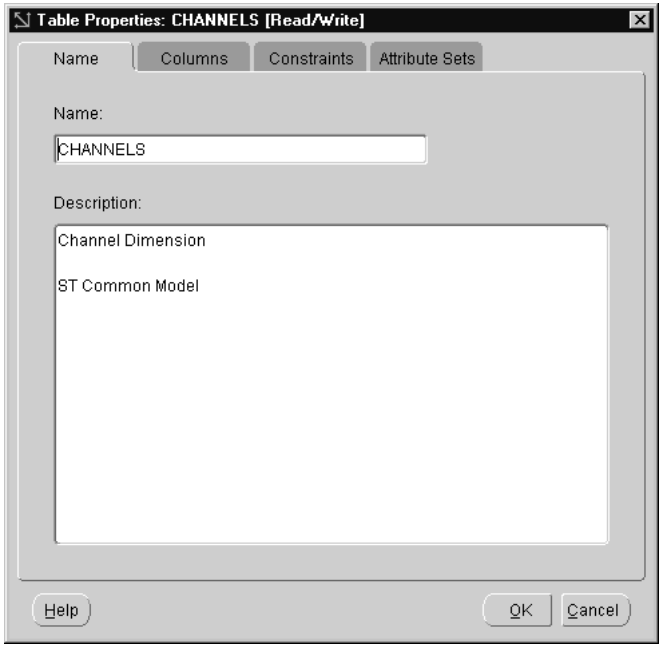
- Change the name and description of the object.
- Add or Remove a foreign key reference constraint.

  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).

Edit the table property sheet if you want to change the physical properties of a fact.

To display the fact table property sheet, right-click the fact and select **Table Properties** from the pop-up.

Figure 4–18 Table Properties Sheet



From the Table Properties sheets, you can:

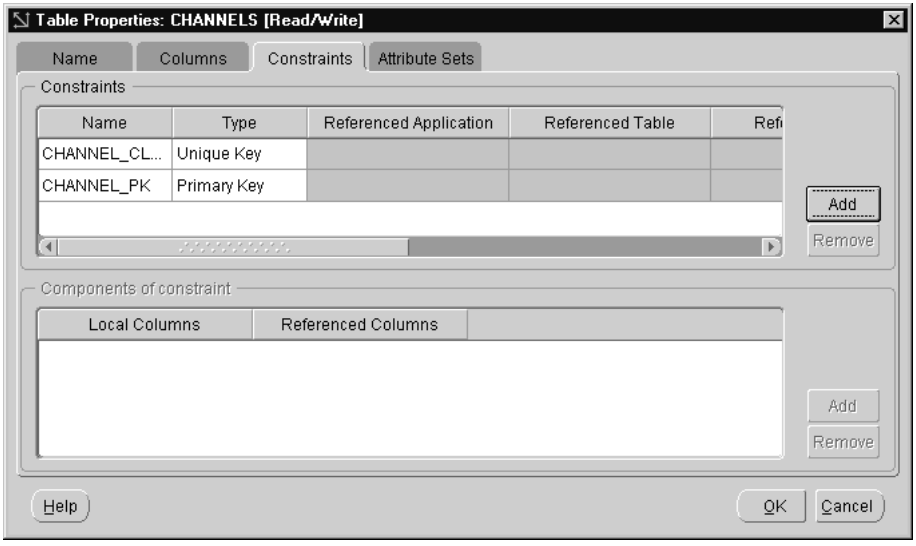
- Change the name and description of the object.
- Add a new column, edit the name, data type, column position, and description of an existing column, or remove a column. For a description on positioning columns, see "Ordering the Columns" on page 3-11.
- Add, remove, or edit a foreign key reference constraint.
- Add, remove, or edit attribute sets.

Note: Although you can change a UK constraint to a PK constraint, you cannot change an FK constraint to another kind of constraint. You must first remove the constraint and then create a new one.

Changing Key Constraints The Oracle8*i*/9*i* server can automatically refresh materialized views provided that a materialized view log file is created for the

underlying fact table. To build the log file, the fact table must have a PK constraint. You can change the composite UK constraint defined on a fact table to a PK constraint.

Figure 4–19 Constraints Tab of the Table Properties Sheet



To change key contraints:

1. Open the Table properties sheet.
2. Select the **Constraints** tab.
3. Select the generated segmented key name.
4. Select **Primary Key** from the Type drop-down list.
5. Change the name of the constraint to reflect its new property.
6. Click **OK.**

Importing Definitions

You can import definitions for tables, views and sequences using the Import Metadata Wizard. See "Importing Definitions from a Database" on page 5-19 for instructions.

Adding Transformations

Transformations are pre-built PL/SQL functions, procedures, package functions, and package procedures. They take input data, perform operations on it, and produce output data. Custom transformations are used to define an operation outside of the Oracle Library. You create a custom transformation using the New Transformation Wizard.

The following sections describe the transformation libraries and how to create custom transformations.

About Transformations

Warehouse Builder supports the following transformation types:

- **User Transformation Package:** This category contains package functions and procedures that you define.
- **Predefined Transformations:** These categories exist in the Oracle Library and consist of built-in and seeded functions and procedures.
- **Functions:** The functions category is automatically created in every warehouse module. This category contains any standalone functions used as transformations. These functions can be defined by the user or imported from a database. A function transformation takes 0-n input parameters and produces a result value.
- **Procedures:** The procedures category is automatically created in every warehouse module. 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.
- **Imported Package:** This category is created by importing a PL/SQL package. 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.

About Transformation Parameters

Most transformations have parameters. The input parameter specifies a source of data, the output specifies a result. More complex transformations often have multiple input, input/output, and output parameters.

About Oracle Transformation Libraries

Each time you create a warehouse module, Warehouse Builder creates a Transformation Library for that module containing transformation operations. This library contains the standard Oracle Library and an additional library for each warehouse module defined within the repository.

Transformation Libraries consist of the following types:

- **Global Shared Library:** a collection of re-usable transformations categorized as functions and procedures defined within your repository.
- **Oracle Library:** a collection of pre-defined functions from which you can define procedures for your Global Shared Library.

When you create a custom transformation, add it to the Global Shared Library to share across warehouse modules. If the transformation is specific to one module, add it to the transformation library within that module.

Global Shared Library

The Global Shared Library stores transformations that are shared across a repository. The default categories are:

- **Functions:** This category stores standalone functions.
- **Procedures:** This category stores standalone procedures.

Oracle Library

The Oracle Library includes a set of standard transformations organized into categories including:

- Administration
- Character
- Conversion
- Date
- Numeric
- Other
- XML

Accessing Transformation Libraries

You can access the Transformation Libraries from Expression Builder, the Add Transformation dialog, or the New Transformation Wizard. You can also access Transformation Libraries from the navigation tree in the Warehouse Builder Console.

Creating Transformation Libraries

You can create transformation libraries to organize transformations. The following steps show you how to create transformation libraries within the warehouse module using the New Transformation Library Wizard.

To create a transformation library:

1. Expand the navigation tree for the active warehouse module.
2. Right-click **Transformation Libraries** and then select **Create Transformation Library.**

   Warehouse Builder displays the New Transformation Library Wizard welcome page.
3. Click **Next.**

   The wizard displays the Name page.
4. Enter a name and description for the library.

   A library name can have from 2 to 40 alphanumeric characters but no spaces.
5. Click **Finish.**

   The wizard inserts the name of the library in the module's navigation tree.

Defining Custom Transformations

Custom transformations are used to define an operation outside of the Oracle Library. Definitions for custom transformations are stored in a warehouse module Global Shared Library in the Transformations branch that can be subdivided into categories in the Module Editor.

Create new transformations using the New Transformation Wizard. To create a transformation, enter the SQL or PL/SQL code that the transformation executes.

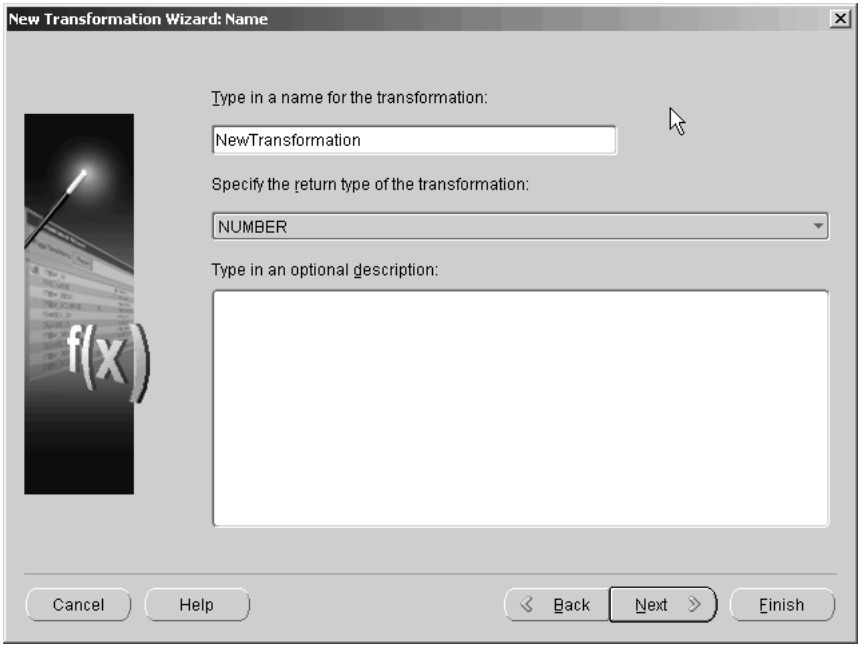
To define a custom transformation:

1. Open the Transformation node on the Navigation Tree in the Warehouse Module Editor

2. Select a transformation category for the type of transformation you want to create.
3. Right-click on the category and select **Create Transformation** from the pop-up menu.

   Warehouse Builder opens the New Transformation Wizard.
4. Enter a name in the Name field and a description documenting what the transformation does.
5. Select a transformation type from the drop-down list.
6. Click **Next.**

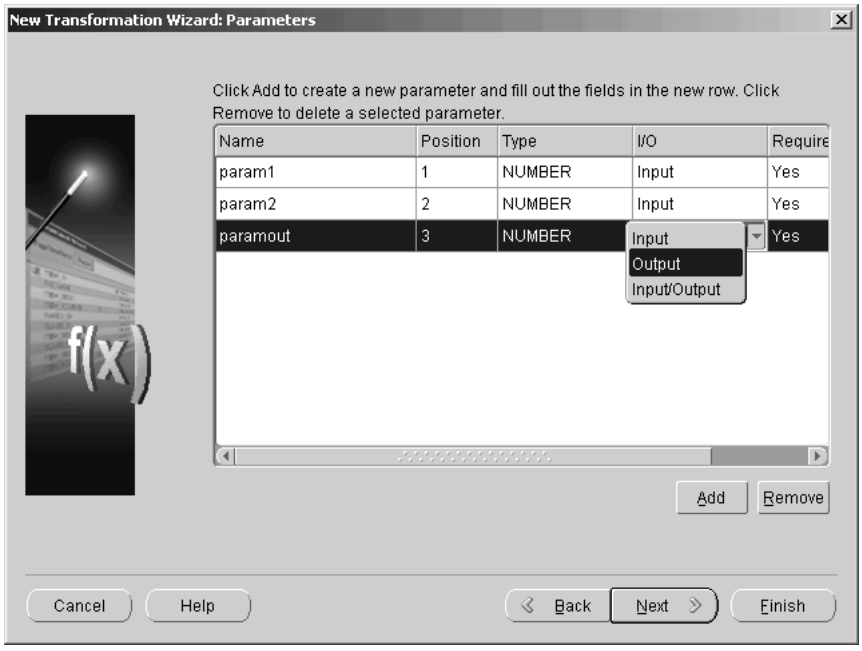
Figure 4–20 Transformation Name Page



7. Define each parameter for the transformation on the Parameters page.
   a. Click **Add.**
   b. Enter 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.

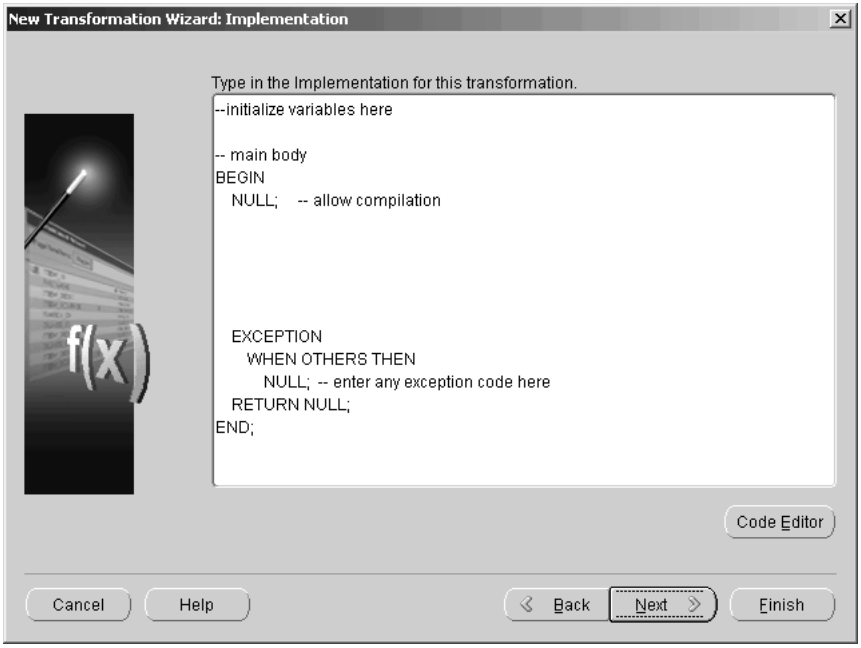
Figure 4–21 Transformation Parameter Page



8. Click **Next.**

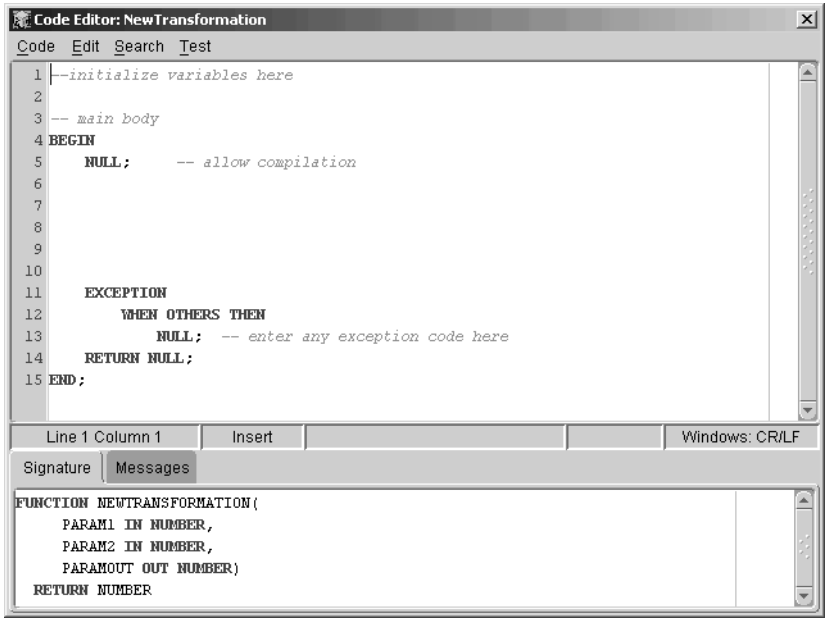
9. Enter the PL/SQL code for the parameter on the Implementation page.

Figure 4–22 Transformation Implementation Page



- Click **Code Editor** to display the code editor. The code editor has line numbers, find, deploy, and syntax checking.

Figure 4–23 Code Editor for a New Transformation



- After values have been specified for each parameter, if necessary, click **Back** to make corrections.

10. Click **Finish.**

Editing Transformation Properties

You can edit a transformation on the transformation properties sheet. Make sure you edit properties consistently. For example, if you change the name of a parameter, then you must change the name in the implementation code.

Importing PL/SQL Packages

Using the Import Wizard, you can import PL/SQL functions, procedures, and packages into a Warehouse Builder project. When Warehouse Builder generates a script for the extract and load job, it generates the added constraint within the PL/SQL routine that implements the mapping. At runtime, you can accept the default value or supply a different one.

When you submit the script using Oracle Enterprise Manager, you can then modify the runtime parameter value.

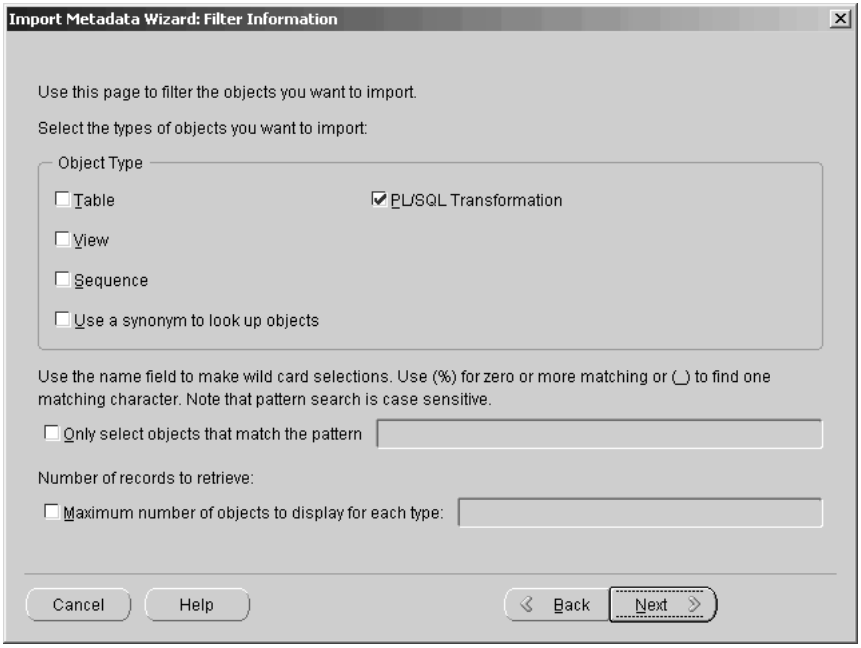
The following steps describe how to import PL/SQL packages from other sources into Warehouse Builder.

To import a PL/SQL function, procedure, or package:

1. Open the Warehouse Module Editor.
2. From the **Module** menu, select **Import.**

   Warehouse Builder displays the Import Metadata Wizard Welcome page.
3. Click **Next.**
4. Select **PL/SQL Transformation** in the Object Type field of the Filter Information page.

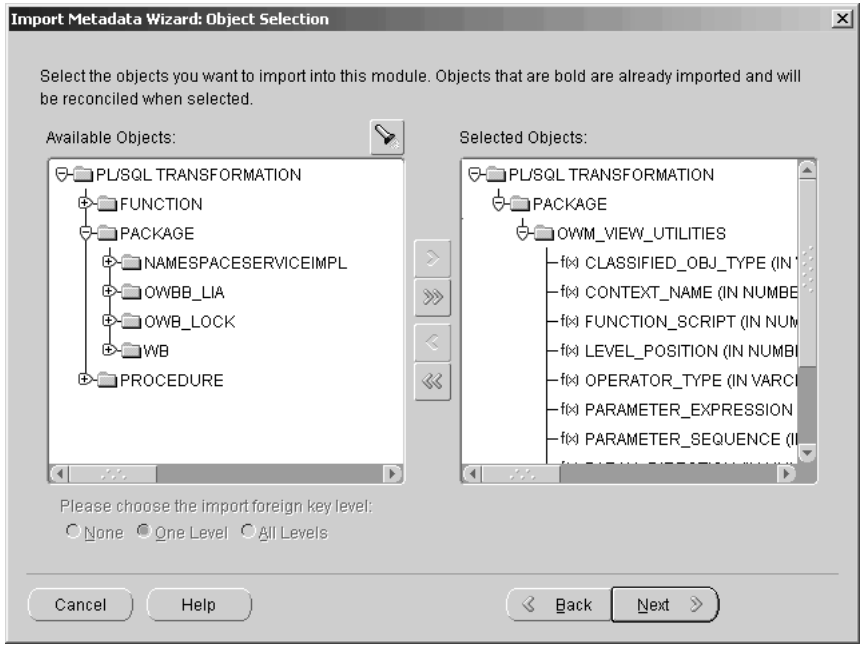
Figure 4–24 PL/SQL Transformation Selection



5. Click **Next.**

   The Import Metadata Wizard displays the Object Selection page.

Figure 4–25 Object Selection Page

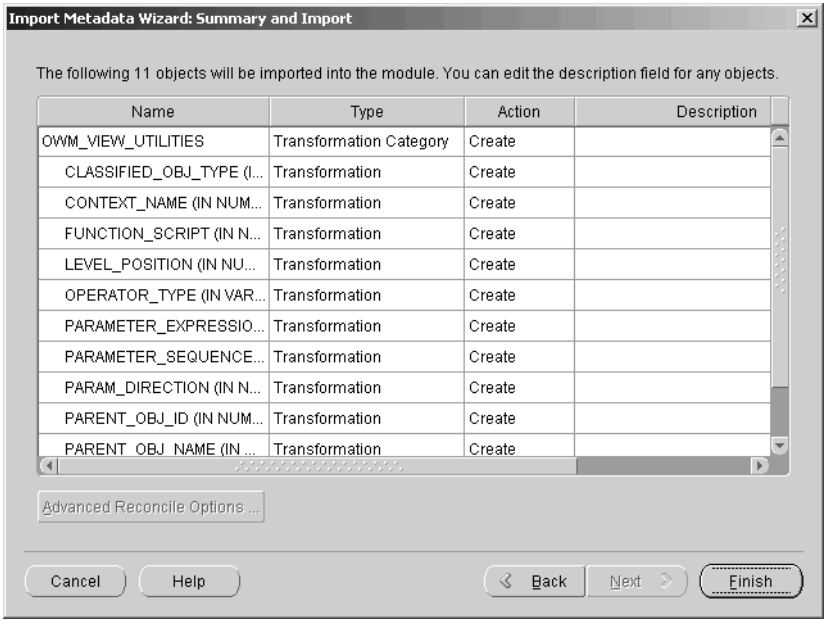


6. Select a function, procedure, or package from the Available Objects list. Move the objects to the Selected Objects list by clicking the single arrow button to move a single object or the double arrow button to move multiple objects.

7. Click **Next.**

   The Import Metadata Wizard displays the Summary and Import page.

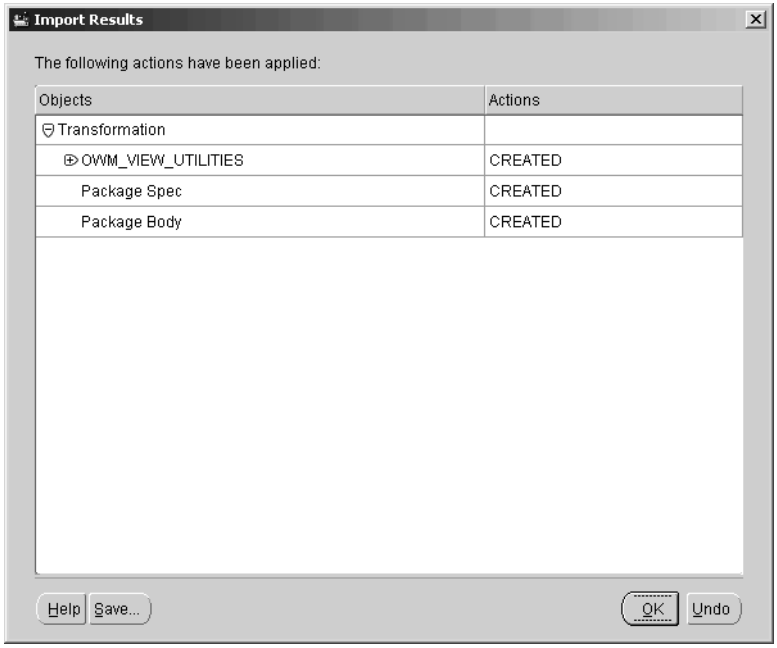
Figure 4–26 Summary and Import Page



8. Verify the import information. Click **Back** to revise your selections.
9. Click **Finish** to import.

   The Import Results dialog displays.

Figure 4–27 Import Results

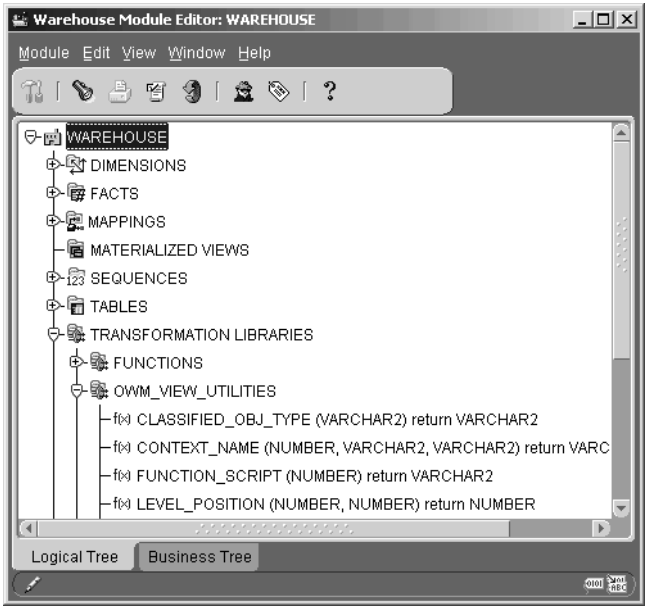


10. Click **OK.**

Click **Undo** to cancel the import process.

The imported PL/SQL information appears in the Module Editor under Transformation Libraries.

Figure 4–28 Warehouse Module with Imported Transformations



When you use the imported PL/SQL:

- You can edit, save, and deploy the imported PL/SQL functions and procedures.
- You cannot edit imported PL/SQL packages.
- Wrapped PL/SQL objects are not readable.
- Imported packages can be viewed and modified in the category property sheet.
- You can edit the imported package body but not the imported package specification.

Defining Business Areas

A business area is a logical grouping of data within a warehouse module. These areas define links to a subset of the module's objects. You can define multiple business areas within a warehouse module. The business tree displays a warehouse module's business areas.

Business areas are useful when the logical warehouse contains a large number of objects. Define business areas when you need to examine subsets of warehouse

objects or export subsets of objects to a decision support tool such as Oracle Discoverer or Oracle Express.

Oracle Discoverer and Oracle Express use business areas to provide their users with access to the data they need for ad hoc queries, decision support, and presentation of results. You can use the Warehouse Builder Transfer Wizard to export business areas from Warehouse Builder to Discoverer and Express.

You can create multiple business areas that share the same links. After you create a business area, you can update its definitions using the standard Warehouse Builder editors, such as the Dimension, Fact, Table, and Mapping editors. You can also update an object definition by editing its property sheet. For additional information on updating definitions, see the previous examples in this chapter.

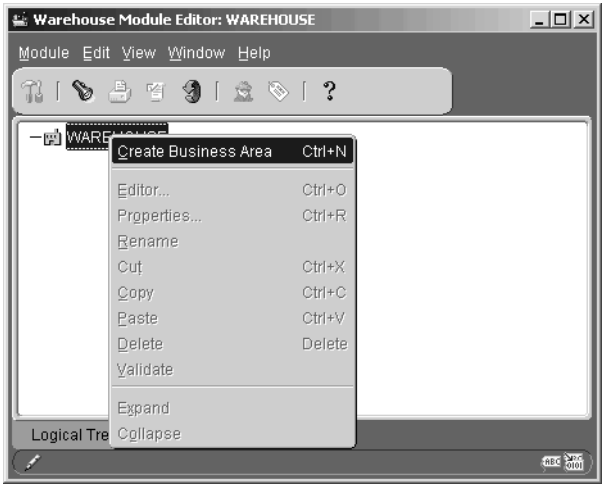
Note: When you create a business area, you do not create any new objects. The business area only organizes existing objects into identifiable subsets.

To create a definition for a subset of warehouse objects in a business area:

1. Open the warehouse module.
2. Click the **Business Tree** tab.

   Warehouse Builder displays the navigation tree for the business areas.

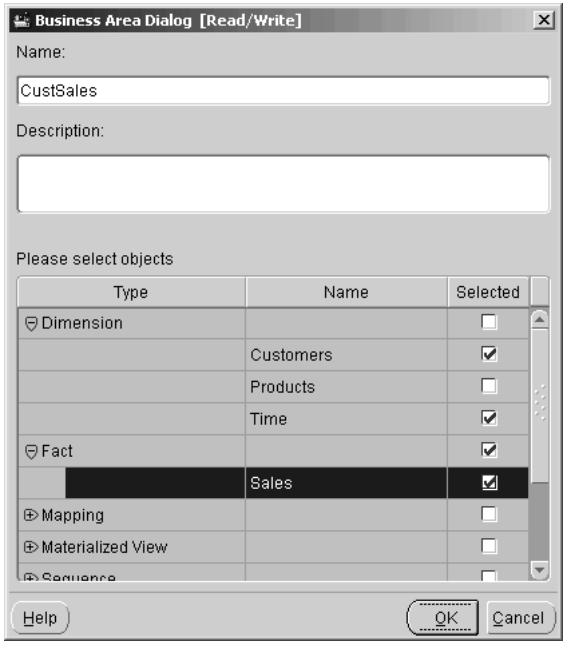
Figure 4–29 Create Business Area



3. Right-click on the warehouse module and select **Create Business Area** from the pop-up menu.

   Warehouse Builder displays the Business Area dialog.

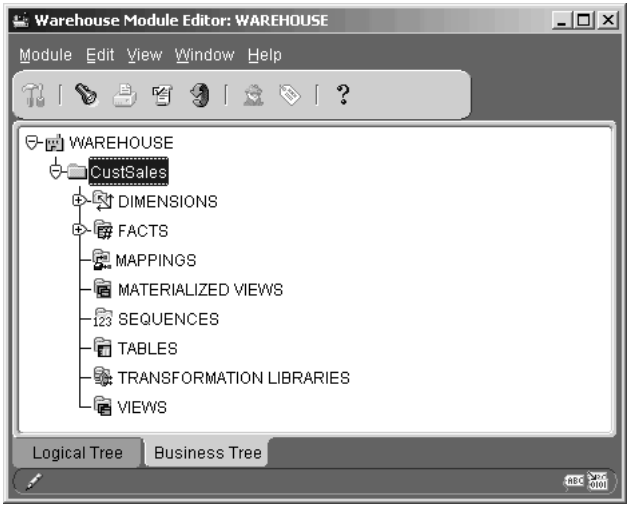
Figure 4–30 Business Area Dialog



4. Type a name and description for the business area.
5. Select the objects to include within the business area. You can select the check box for a type to select all objects of that type.
6. Click **OK.**

   Warehouse Builder creates the business area.

Figure 4–31 Business Tree



5

Defining Source Modules

This chapter describes how to create definitions for data sources. This chapter also shows you how to modify definitions, generate and display diagrams for the source objects, and print diagrams.

This chapter includes the following topics:

- Creating Source Modules
- Importing Definitions from Database Sources
- Creating Definitions for Flat File Sources
- Importing Definitions for Pure Extract and Pure Integrate
- Importing Definitions for SAP Data Sources

Creating Source Modules

To access a source, you create a source module. You can create source modules for:

- Oracle databases
- Non-Oracle databases
- Designer repositories
- Flat files
- SAP

Warehouse Builder uses software integrators to read definitions and extract data from source systems. The New Module Wizard determines the correct integrator for a specific case from the source type and the version or system type. You can configure a module for an Oracle system, a non-Oracle database, or an Oracle Designer repository.

Table 5–1 summarizes application types and their corresponding integrators.

Table 5–1 Applications and Corresponding Software Integrators

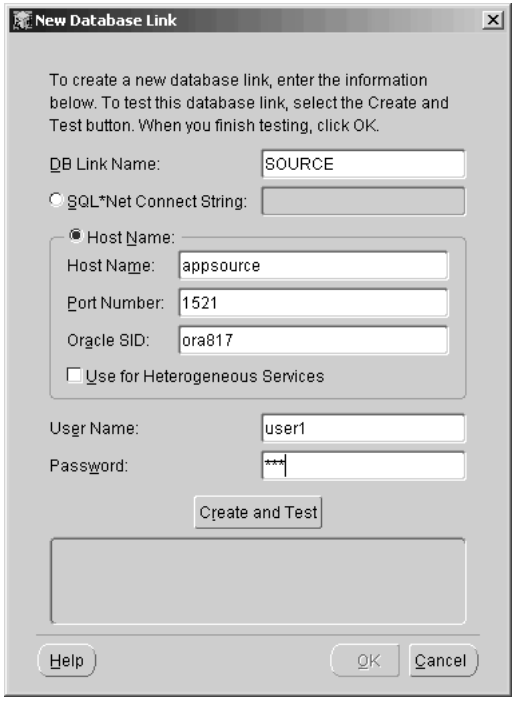
Type of Source	Application Version or System Type	Integrator
Oracle Database	Oracle Database 7.3, 8.0, 8*i*, 9*i*	OWB Integrator for Oracle DB & Apps 3.0
Non-Oracle Database	Oracle Generic Gateway Connectivity	OWB Integrator for Oracle DB & Apps 3.0
SAP R/3 3.x, 4.x	SAP Application Server	OWB Integrator for SAP Applications 3.0
Flat File System	Generic File System	OWB Integrator for Flat Files

Configuring Connection Information 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. Warehouse Builder uses the link to access the data dictionary of the source.

You 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. Create a new database link using the New Database Link dialog. Database administrators usually prefer to create and control database links to avoid security issues.

Figure 5–1 New Database Link Dialog



To create a new database link, specify the information in Table 5–2. You can specify the connection information as a connect string or as individual values.

Table 5–2 New Database Link Parameters

Parameter	Description
DB Link Name	A database link name can be a maximum of 128 bytes and can include periods (.) and the at sign (@).
SQL*Net Connect String	A connect string for the database system. For a non-Oracle database: include '(HS=OK)' in the connect_ data clause.
Host Name Information	Alternate specification of values for database link parameters.

Table 5–2 New Database Link Parameters (Cont.)

Parameter	Description
Host Name	Alias for the IP address of the host machine.
Port Number	Configured port for the Oracle Listener.
Oracle SID	SID for an Oracle Instance or an Oracle Transparent Gateway.
Heterogeneous Services	For a non-Oracle database: check this box. This includes systems accessed via ODBC, OLE DB or an Oracle Transparent Gateway.
User Name & Password	User name and password for the database system. Case sensitive names and passwords need to be double-quoted.

For more information on database links and connect strings, see *Oracle8i / 9i Distributed Database Systems* and *Oracle8i / 9i SQL Reference.*

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, see "Updating a Source Module" on page 5-18.

Warehouse Builder communicates with non-Oracle systems using Oracle8*i* / 9*i* Heterogeneous Services and a complementary agent. Heterogeneous Services makes 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 Oracle8*i* / 9*i*.

- 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 Oracle8*i* / 9*i* database server. You must still create and customize an initialization file for your generic connectivity agent.

For additional information on distributed processing systems, see *Oracle8i / 9i Distributed Database Systems.*

Note: The init<sid>.ora parameter OPEN_LINKS of an Oracle database determines the number of simultaneously open links during a session. If the value is too small, the Oracle8*i*/9*i* instance returns an ORA-2020 error.

Creating a Database Source Module

This section describes how to create a source module that connects with an application based on a database system.

To create a database source module:

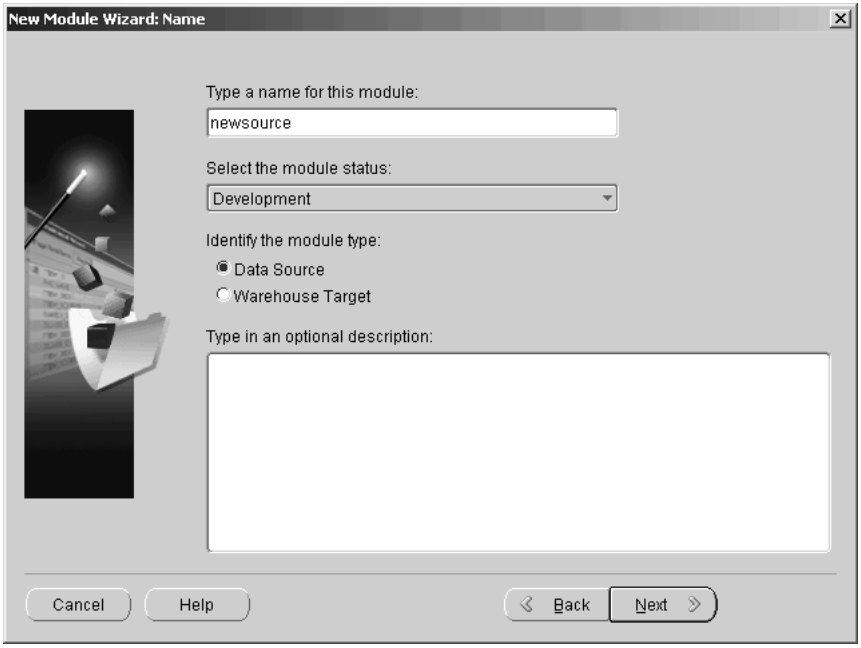
1. Select **MODULES** and then **Create Module** from the Warehouse Builder Console menu or right-click **MODULES** and select **Create Module.**

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

2. Click **Next.**

   The wizard displays the Name page.

Figure 5–2 Name Page



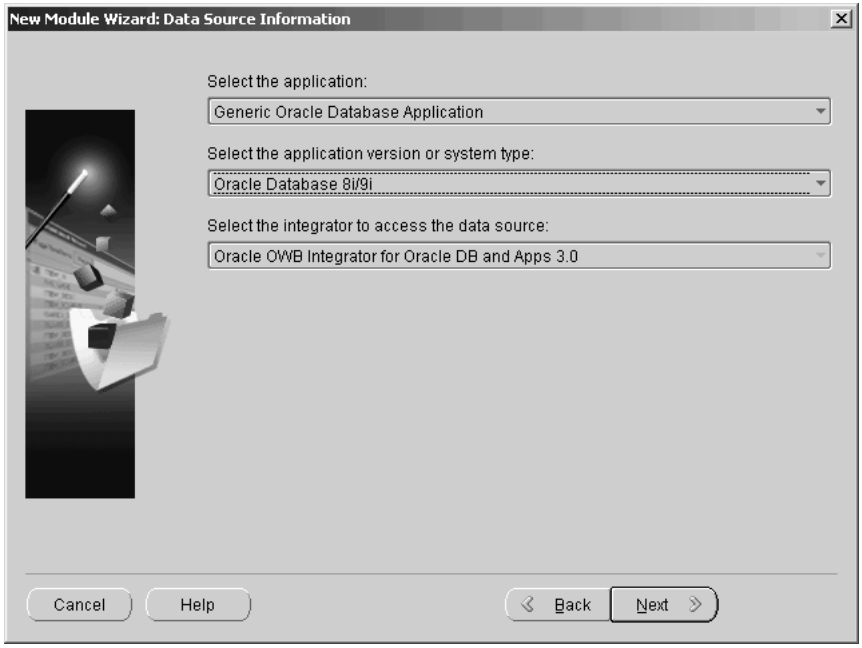
3. On the Name page, type:
   - Name of the module
   - Status of the module

     Specify the status as Development, Quality Assurance, or Production. This status is for descriptive purposes only.
   - **Source** as the Module Type
   - Description (optional)
4. Click **Next.**

   The wizard displays the Data Source Information page.
5. 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.

Figure 5–3 Data Source Information Page



Note: For a non-Oracle Database, select Oracle Generic Gateway Connectivity as the Database Version.

6. Click **Next.**

   The wizard displays the Connection Information page.

7. Select Oracle Data Dictionary as the metadata source.

8. 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 Connection Information for Database Sources" on page 5-2.

   When the database link points to a heterogeneous services agent, the page contains an additional box for the gateway agent.

   The wizard displays the link information and the name of the schema owner.

Figure 5–4 Connection Information Page

9. To change the schema owner name, click **Change Schema.** The wizard displays a list of users.
10. Select a schema and then click **OK.**

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

Creating a Source Module for an Oracle Designer Repository

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 itself.

A property on the New Module Wizard's Connection page determines whether a module can import definitions from an Oracle Database or from an Oracle Designer Repository.

Figure 5–5 Oracle Designer Repository Option

Check the button and select a database link that connects with the host where the Oracle Designer Repository resides. Otherwise, create the source module using the procedure described for the previous example.

Creating a Source Module for Designer 6*i*

This section describes how to create a source module that connects with an Oracle Designer 6*i* repository.

Designer 6*i* includes versioning for accessing repository objects. Designer 6*i* uses workareas to control versions of an object. By selecting a workarea, you specify a version of a repository object. The workarea acts as an object qualifier that you specify before you can access it.

With Designer 6*i*, you can 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* Application Systems are controlled by workareas, they have version control. Consult the Designer 6*i* documentation for more information about workareas and application systems.

All visible objects of an Application System are available for use as data sources in Warehouse Builder. To select Designer 6*i* objects as Warehouse Builder sources:

- Specify a workarea
- 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*, 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 within 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 create a Designer 6i source module:

1. Select **MODULES** and then **Create Module** from the Warehouse Builder Console menu or right-click **MODULES** and select **Create 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, 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.
   - **Source** as the Module Type
   - Description (optional)
4. Click **Next.**

   The wizard displays the Data Source Information page.

5. 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.

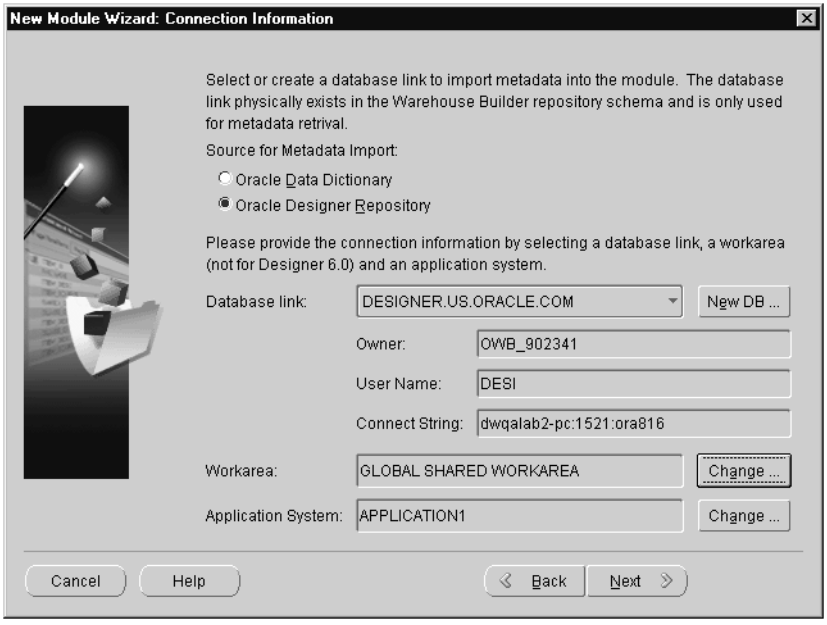
6. Click **Next.**

The wizard displays the Connection Information page.

7. Select **Oracle Designer Repository** as the metadata source.
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 Connection Information for Database Sources" on page 5-2.

Figure 5–6 Connection Information Page

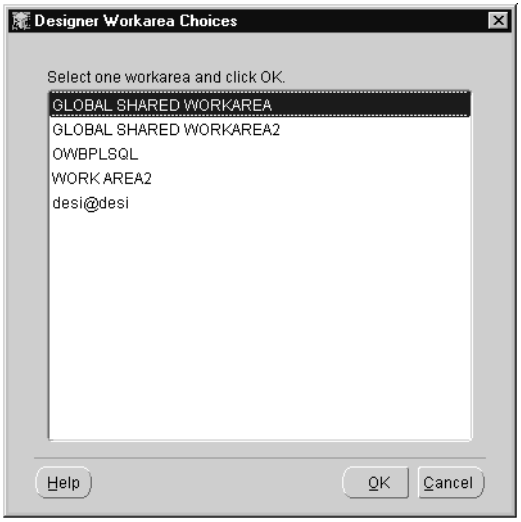


If the New Module Wizard detects a Designer 6*i* repository, the Workarea field, Application System field, and Change button is enabled. If the New Module Wizard detects a Designer 6.0 repository, an alert displays and the Workarea field and Change button are disabled.

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.

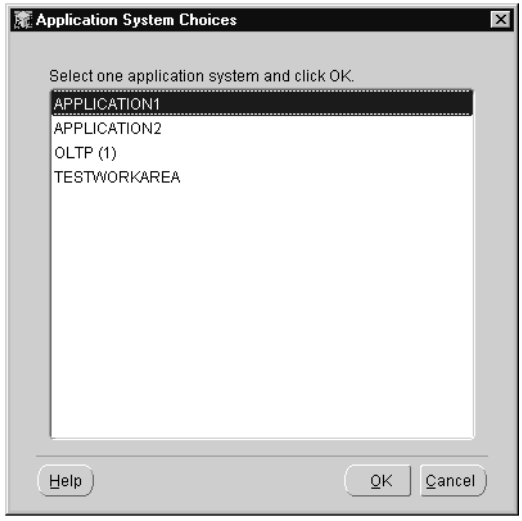
Figure 5–7 Workarea Selection List



10. To specify the Application System, click **Change** and choose one from the selection list.

If you have not set a workarea, a dialog box prompts you to do so. You must set a workarea before you can select an application system.

Figure 5–8 Application Systems Selection List



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.

Creating a Flat File Source Module

A project may need to extract data from flat files. This section describes how to create a source module for a flat file source. Before you can create a file source module, the files must be local to the computer where the Warehouse Builder client is installed.

When the files reside on a UNIX operating system, the directory containing the files must be mapped to the Warehouse Builder host, which requires third-party software on the host to support a Network File System (NFS) connection.

The general steps for mapping a UNIX directory to a Warehouse Builder host are:

1. Set up the UNIX host as an NFS server.

2. Set the source directory on the UNIX machine as sharable.
3. Install the third-party NFS-software package on the Warehouse Builder host.
4. Map the UNIX directory to the Warehouse Builder host.

When you create a source module using the New Module Wizard, you can point the module to the NFS directory using the Browse button or by entering the complete path name to the directory.

To create a file source module:

1. Select **MODULES** and then **Create Module** from the Warehouse Builder Console menu or right-click **MODULES** and select **Create 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:
   - Name of the module
   - Status of the module

     Specify the status as Development, Quality Assurance, or Production. This status is for descriptive purposes only.
   - **Source** as the Module Type
   - Description (optional)
4. Click **Next.**

   The wizard displays the Data Source Information page.

5. Select ***Generic File Based Application*** for the Application.

   The wizard determines the other selections.

6. Click **Next.**

   The wizard displays the Connection Information page.

   The Oracle9*i* Warehouse Builder integrator for flat files reads files that have fixed-length or delimited single record formats. Fixed-length files can contain logical records that have multiple physical records. Multiple record types can be classified in either fixed-length or delimited files.

Figure 5–9 Connection Information Page

7. Enter the name of the drive that contains the source directory. This drive must be mapped to the Warehouse Builder host.
8. Enter the directory that contains the file. Click **Browse** to select a directory.
9. Click **Next.**

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

    The wizard creates the source module and inserts its name in the project navigation tree.

Create a Source Module for SAP Definitions

This section describes how to create a source module for an SAP data source. After you create the source module, you can import the metadata definitions from SAP tables using the Import Metadata Wizard.

To create the SAP source module:

1. Expand the navigation tree for your project.
2. Right-click **MODULES** and select **Create Module.**

   Warehouse Builder displays the Welcome page for the New Module Wizard.
3. Click **Next.**

   The wizard displays the Name page.
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. These options can be used to document the warehouse design version.
   - **Module Type:** Select **Data Source.**
   - **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: SAP R/3 3.x or SAP R/3 4.x.

   Note: If you select the wrong SAP application, the SAP Integrator will display an error when you attempt to log on. You must then click **Back** to return to this page and select the correct application.

When you select the Application Type (SAP), the wizard automatically selects the System Type and Integrator Type (SAP) for the application.

7. Click **Next.**

   The wizard displays the Connection Information page for SAP applications.
8. Provide the following information in the Connection Information page:
   - **Connection Type:** Select remote function call (RFC) or SAP remote function call (SAPRFC.INI File).
   - **Connection Information:** The fields within this box depend on the connection type you choose.

     RFC Connection type requires the following connection information:

- Application Server: the name of the SAP application server
- System Number: the SAP system number for SAP GUI login
- Client: the SAP client number
- User Name: the SAP GUI user name
- Language: EN for English or DE for German (If you select DE, only the description text will be in German; all other text will be in English.)

SAPRFC.INI File requires the following connection information:

- RFC Destination: the alias for the SAP connection information
- Client: the SAP client number
- User Name: the SAP GUI user name
- Language: EN for English or DE for German (If you select DE, only the description text will be in German; all other text will be in English.)

Note: To use the SAPRFC 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.

---

9. Click **Next.**

   The wizard displays the Logon dialog for the SAP application.

10. Enter the SAP GUI password for your SAP GUI user and click **Logon.**

    The wizard attempts to log onto the SAP application. If unsuccessful, the wizard displays an error message; otherwise, it displays the Finish page.

    If the application server version information entered on the Data Source Information page of the wizard does not match the actual application version for the source system, you are prompted with an error message. If necessary, click **Back** to return to the Data Source Information page and correct the application version, then logon again.

    After you log on to the SAP source, the wizard displays the Finish page.

11. To proceed directly to the Import Metadata Wizard, check the box at the bottom of page. You can also choose to import metadata at a later time and leave the box unchecked.
12. Review the module information, then click **Finish.**

    The wizard creates the new SAP source module and inserts its name in the project navigation tree.

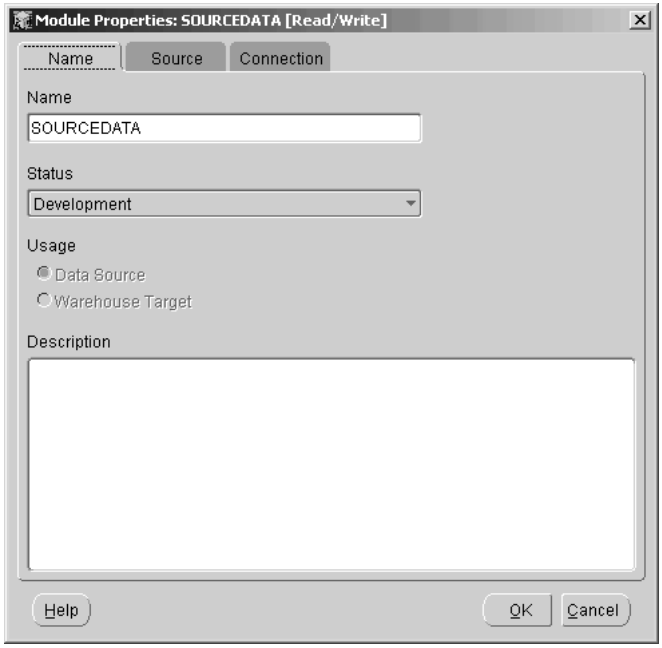
> **Note:** To import metadata at a later time, right-click the source module in the project navigation tree and select **Import** from the pop-up menu.

Updating a Source Module

You can update the definition of a source module by editing its property sheet. To display the property sheet, right-click on the module and select **Properties**. The property sheet for a module contains the following tabs:

- **Name:** Describes the type of module (Data Source or Warehouse Target) and its function.
- **Data Source:** Describes the Application Type, Application Version or System Type, and the Integrator used to access the source.
- **Connection:** Describes the configured connection.

Figure 5–10 Property Sheet for a Source Module



Importing Definitions from Database Sources

This section describes how to import definitions from a database application and store them in a source module.

Importing Definitions from a Database

You 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, and a Designer repository.

Designer Repository Notes: You cannot import Dimension Objects. Position will be renumbered as 10, 20, 30, ….

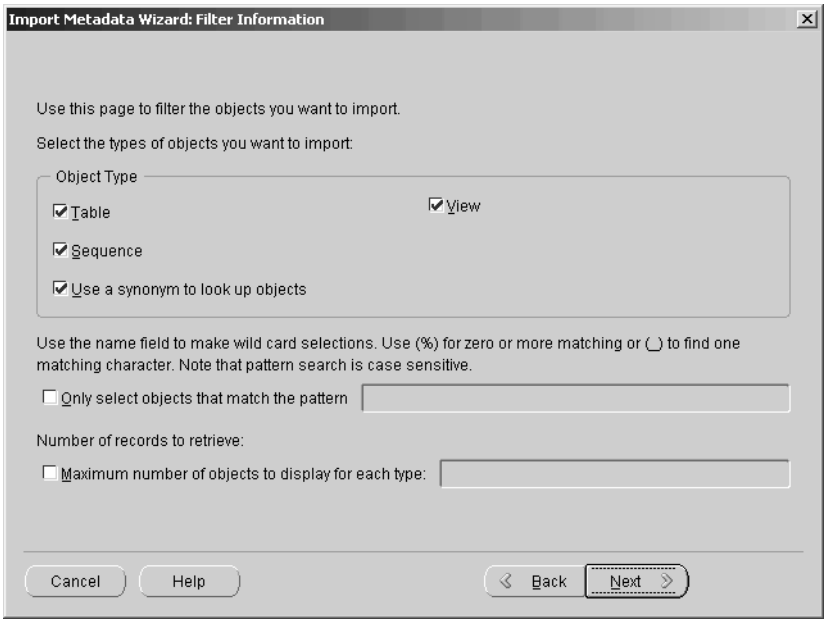
To import definitions from an Oracle database application into a source module:

1. Right-click on 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.

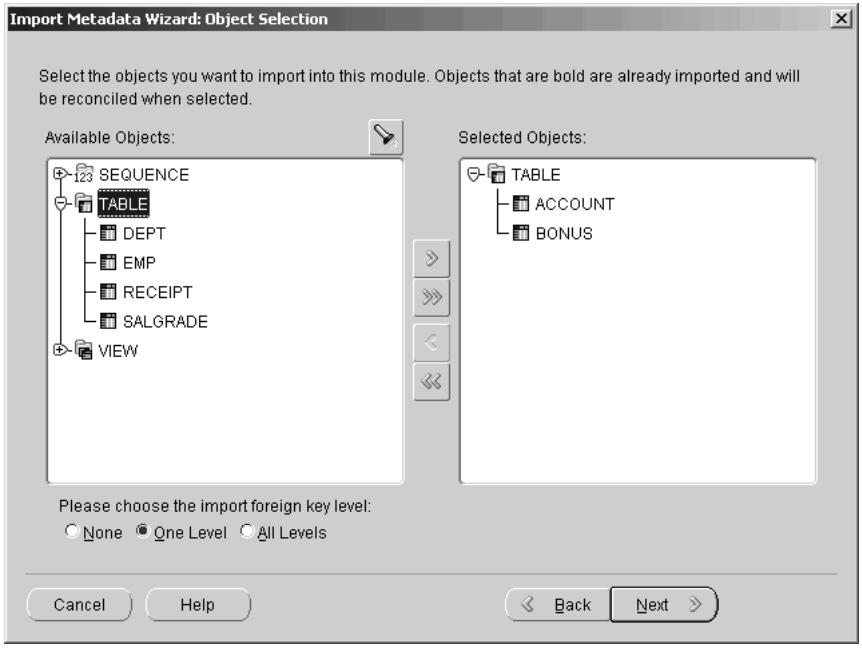


3. Limit the search of the data dictionary. Enter any of the following:
   - Select tables, views, or sequences.
   - Type a search pattern, for example, a warehouse project name followed by a %.
   - Type a maximum number of objects to retrieve.
4. Click **Next.**

   Warehouse Builder retrieves names that meet the filter conditions from the data dictionary and displays the Object Selection page.
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 one name and the names of objects it references, select the name and check **One Level**.
- To move a single name and names of the objects it references directly or indirectly, select its name and check **All Levels**.

Figure 5–11 Import Metadata Wizard Object Selection Page

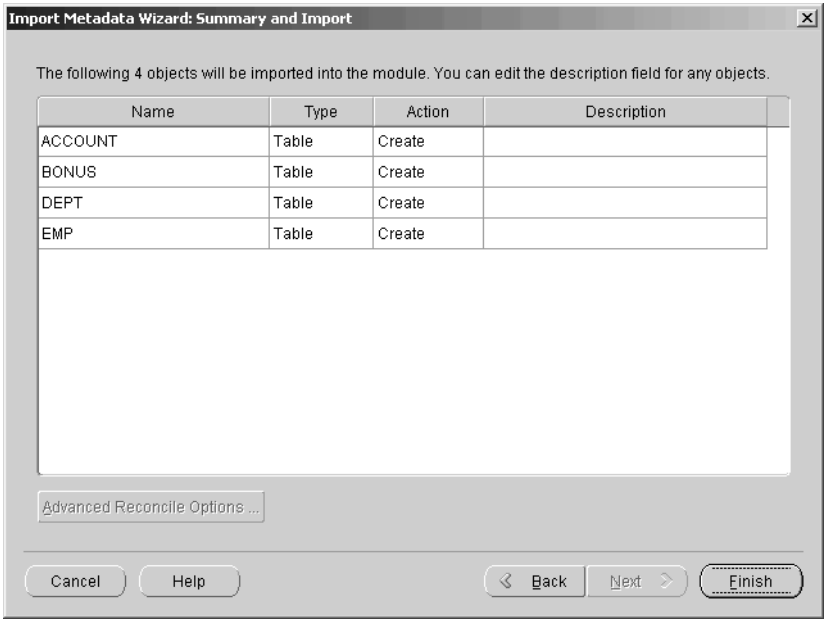


If you are re-importing definitions, previously imported objects appear in bold.

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.

Figure 5–12 Import Metadata Wizard Summary and Import Page



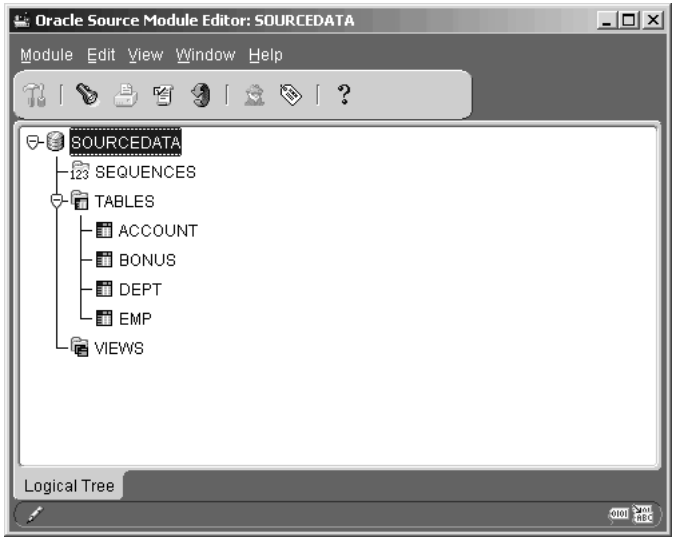
7. Click **Finish.**

   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 module.

Figure 5–13 Source Module with Imported Objects



Re-Importing Definitions from an Oracle Database

Re-importing your source database definitions enables you bring in changes added to the source since your original import. You do not have to remove the original definitions from the repository. You are also given advanced options that enable you to preserve some of the changes you have made to the objects since the original import. This includes any new objects, foreign keys, relationships, and descriptions you have created.

To re-import definitions:

1. Right-click on 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.**

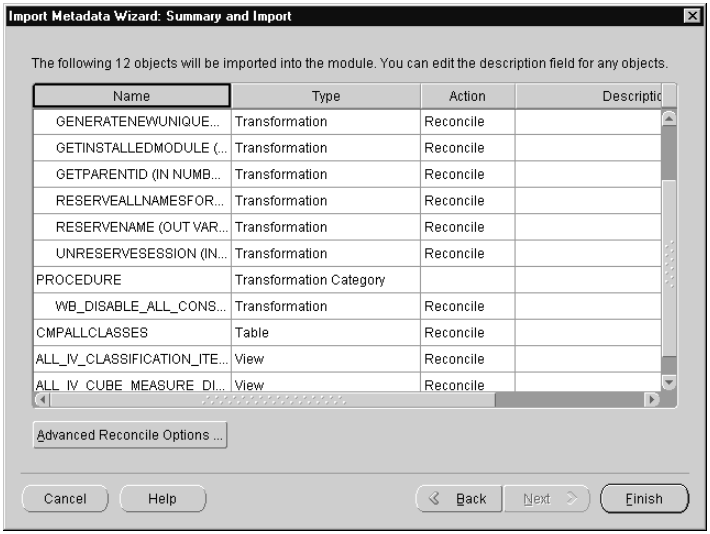
   The Object Selection page displays. The objects that were originally imported display in bold.

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. The Reconcile action is displayed for the objects you are re-importing.

Figure 5–14 Summary and Import Page Showing Reconcile Action



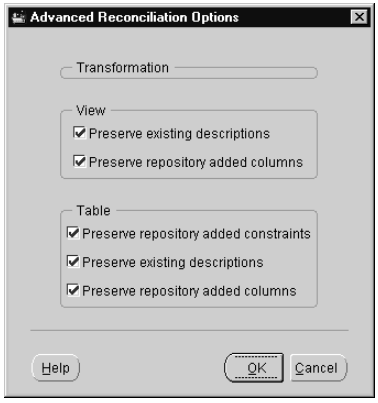
Note: If the source contains new objects related to the object you are re-importing, the wizard requires that you import the new objects at the same time. The Create action displays for these objects.

---

7. Click **Advanced Reconcile Options** to select advanced reconciliation options.

   The Advanced Reconciliation Options dialog displays.

Figure 5–15 Advanced Reconciliation Options Dialog



Select options for reconciling views:

- Preserve existing descriptions
- Preserve repository added columns

Select options for reconciling tables:

- Preserve repository added constraints
- Preserve existing descriptions
- Preserve repository added columns

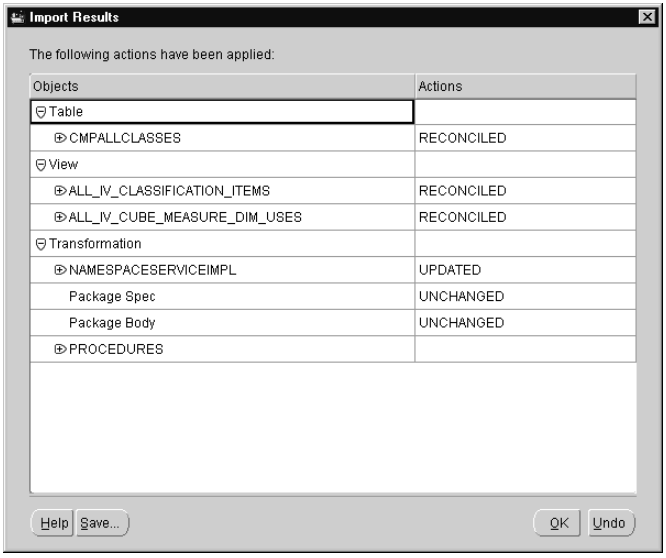
Note: By default, all options are checked. Uncheck boxes to have these repository objects replaced and not preserved.

---

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.

Figure 5–16 Import Results Dialog



The report lists the actions performed by Warehouse Builder for each object.

Click **Save** to save the report. Make sure to use a naming convention that is specific to the re-import.

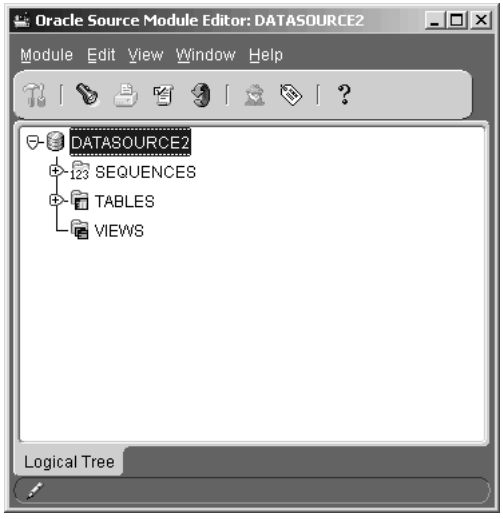
10. Click **OK** to proceed.

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

Updating Oracle Database Source Definitions

You can update source definitions, diagram individual definitions and their references, and print the diagrams using the Source Module Editor. To display this editor, double-click the module name.

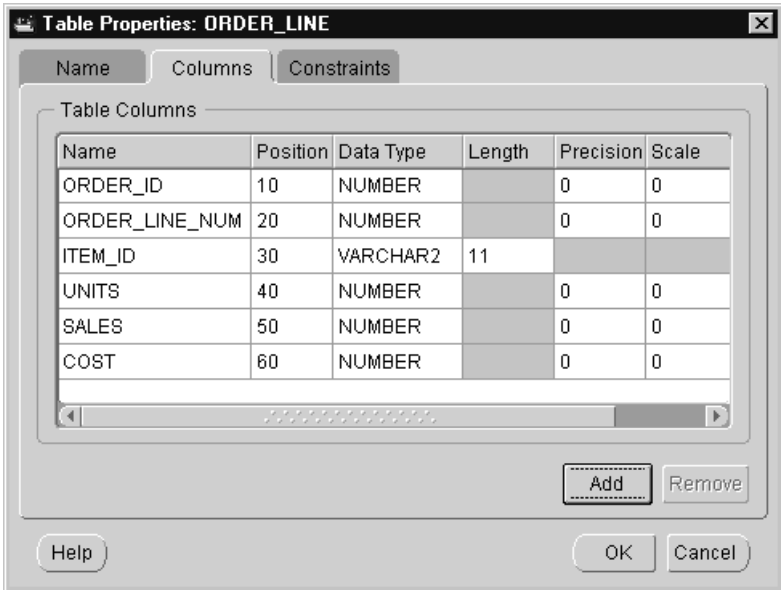
Figure 5–17 Source Module Editor



Update a Source Definition

You can update a source definition by editing entries in its Property Sheet. To display a definition's property sheet, select its name in the navigation tree and then select **Properties** from the pop-up list.

Figure 5–18 Columns Tab of the Table Properties Sheet

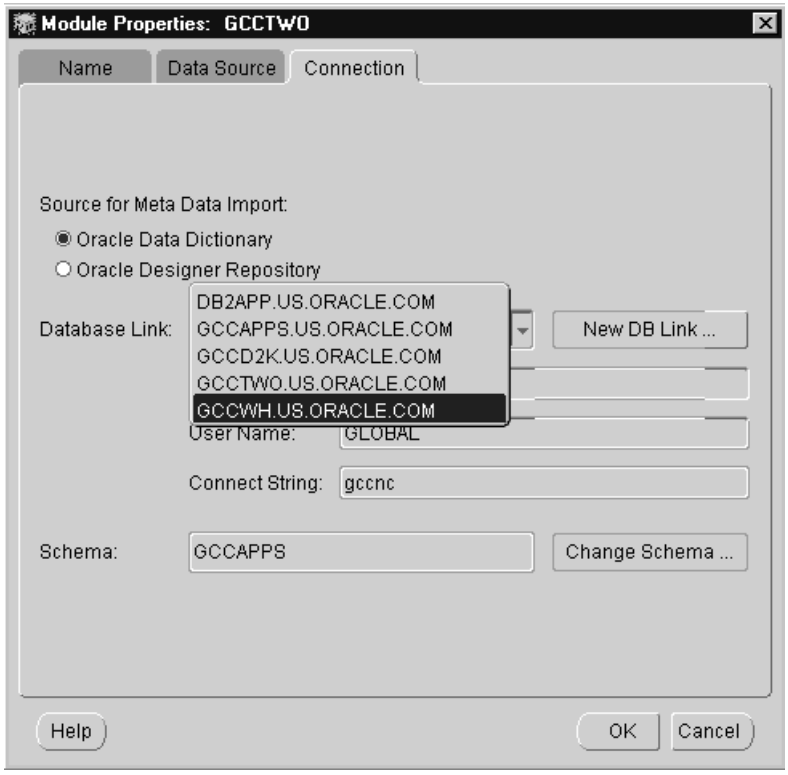


To update an existing entry, select the entry and enter the new information. Some entries have drop-down lists that limit the range of selections. For example, when you change the data type of a column, you must select an entry from a drop-down list. You can also add new entries or remove existing ones.

Update the Connection

You can update the connection information for a data source by selecting another database link from the drop-down list.

Figure 5–19 Connection Tab of the Module Properties Sheet

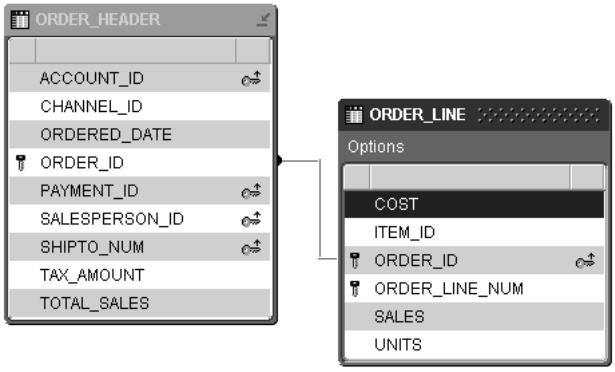


When you change the connection information, Warehouse Builder displays a warning message that you may compromise the existing definitions in the source module. To change the connection, click **OK.**

Diagram a Source Definition

You can display a diagram for a definition and its references using the Source Module Editor. To display a diagram for a definition, select its name in the navigation tree and then select **Edit** from the pop-up list. The Source Module Editor displays a diagram of the definition.

Figure 5–20 Table Editor



Each object in the diagram has a toolbar that you can use to sort the column names. The toolbar is divided into three rectangles. To sort the column names click one of the rectangles:

- Left rectangle determines the position and order of the primary key columns.
- Middle rectangle sorts all the column names.
- Right rectangle determines the position and order of foreign key columns.

The sort order is *only* for display purposes and has no bearing on the ordering of column names within the definition.

Print a Source Definition To print a diagram of a source definition, display the diagram and then click the Print icon on the editor's toolbar.

Creating Definitions for Flat File Sources

This section shows you how to create a definition for a flat file. Using the Flat File Sample wizard you can:

- Create format definitions for delimited and fixed-length files
- Create format definitions for logical records within a fixed-length file that consists of one or more physical records within a file
- Identify and create format definitions for multiple record types within a file

After you create and store a format definition, you can use it to describe other flat files that have the same properties.

About the Flat File Sample Wizard

When you create a definition, the Flat File Sample Wizard opens the file, displays a sample of data, and requests detailed information about the file format. The Flat File Sample Wizard is structured as shown in Table 5–3.

Table 5–3 Flat File Sample Wizard Pages

Page Name	Information Required
Setup	File format: Fixed-length or delimited
	Field delimiter and enclosure characters for delimited format
	Terminator or physical record length for fixed-length format
	Character set
Record Organization	Single or multiple record types
	Number of rows to sample and rows to skip before sampling
	Logical record definition (only available for fixed-length files that have a single record type)
Record Types (only appears for multiple record types)	Column positions specifying the record type
	Name of each record type
Column Definition (only appears for fixed-length files)	Width of each column in a fixed-length file
Properties	Name, type, mask, NULLIF, DEFAULTIF, field length
	Header row

Character set: Warehouse Builder's default NLS character set is the same as its host. If it differs from the source file's character set, the data sample might be unintelligible. You can display the data sample in the source's native character set by selecting it from the drop-down list. For complete information on NLS character sets, see the *Oracle8i/9i National Language Support Guide.*

Physical record length: The length of a fixed-format record can be specified as length in characters or set to a user-defined terminator. The length specification results in greater efficiency.

Logical records: Warehouse Builder can manage a source file of logical records. The number of physical records in a logical record can be fixed or variable. For a fixed number, you specify the number of physical records per logical record. For a variable number, you specify a continuation character at either the end or beginning

of each physical record. See "Specifying Logical Records" on page 5-42 for more information.

Multiple record types: The Flat File Wizard can interpret a source file that contains a variety of record types. You must specify the column within the source file that contains unique record types. You then scan the column to identify unique record type values, then define the characteristics of that record type. The record types can be renamed.

Field type: Describes the data type of the field for the SQL*Loader. Warehouse Builder supports the following set of portable data types:

- CHAR
- DATE
- DECIMAL EXTERNAL
- FLOAT EXTERNAL
- INTEGER EXTERNAL
- ZONED EXTERNAL

The native numeric data is a number in character form; it is not a binary representation. The numeric data types are identical to CHAR except with respect to the DEFAULTIF and NULLIF constraints. See the discussion below on field constraints.

You can represent FLOAT EXTERNAL data either in scientific or regular notation. The representations 5.33 and 533E-2 are both valid.

For complete information on SQL*Loader field and data types, refer to *Oracle8i / 9i Utilities*.

Field 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/DEFAULTIF conditions: You can override the default action of the SQL*Loader by placing a DEFAULTIF or NULLIF condition on a field.

- When a numeric or DATE field contains all blanks, SQL*Loader rejects the entire record. To override this action, include a DEFAULTIF = BLANKS condition on the field. When SQL*Loader evaluates this condition, it sets the field to zeros and loads the record.

- When a character field contains all blanks, you can direct SQL*Loader to mark the column as null rather than storing the blanks by including a NULLIF = BLANKS condition on the field.

When you describe the field using the Flat File Sample Wizard, you can choose one of these constraints.

Creating a Definition for a Fixed-Length File

The following procedure describes how to create a definition for a file using the Import Metadata and Flat File Sample Wizards. Each logical record of this file consists of a single physical record. The location of the file was configured in the warehouse source module.

To create a definition for a flat file format:

1. Right-click on a file module and select **Import.**

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

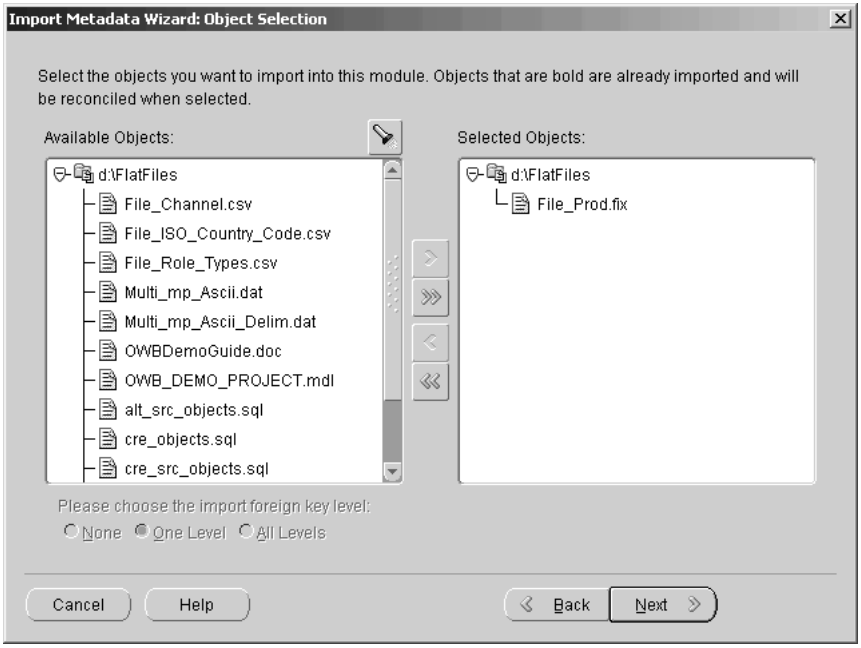
2. Click **Next.**

   The wizard displays the Filter Information page. Use this page to filter file names.

3. Click **Next.**

   The wizard displays the Object Selection page.

Figure 5–21 Object Selection Page



4. Move the name of the file to be described from the Available to the Selected Objects window pane.

5. Click **Next.**

   The wizard displays the Summary and Import page. The left-most column of this page contains a status ball which can be red or green. If green, then Warehouse Builder already has a definition of the file's format—proceed to step 15; if red, then you must create a format for the file using the Flat File Sample Wizard.

6. Select a file that has a red status ball and click **Sample** at the bottom of the Summary and Import page.

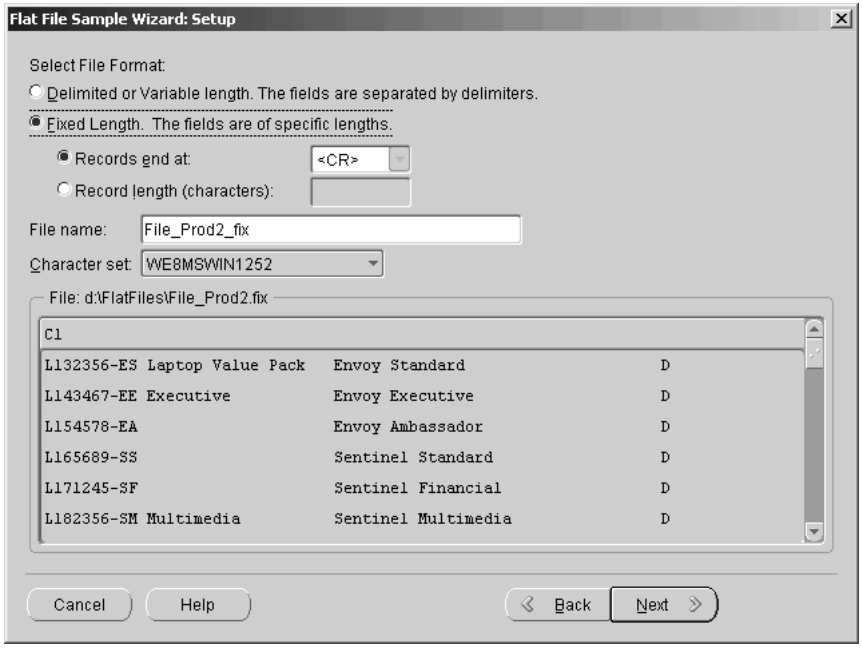
   The wizard displays the welcome page for the Flat File Sample Wizard.

7. Click **Next.**

   The wizard opens the file, reads a sample of data, and displays the File Setup page. This page displays the sample of data in a template with a few initial values set for the global properties.

8. Verify and select the global properties:
   - File format: Fixed Length
   - Record ends at: <CR> by default.
   - Record size can be specified by length
   - NLS Character set: WE8MSWIN1252

Figure 5–22 Flat File Sample Wizard Setup Page

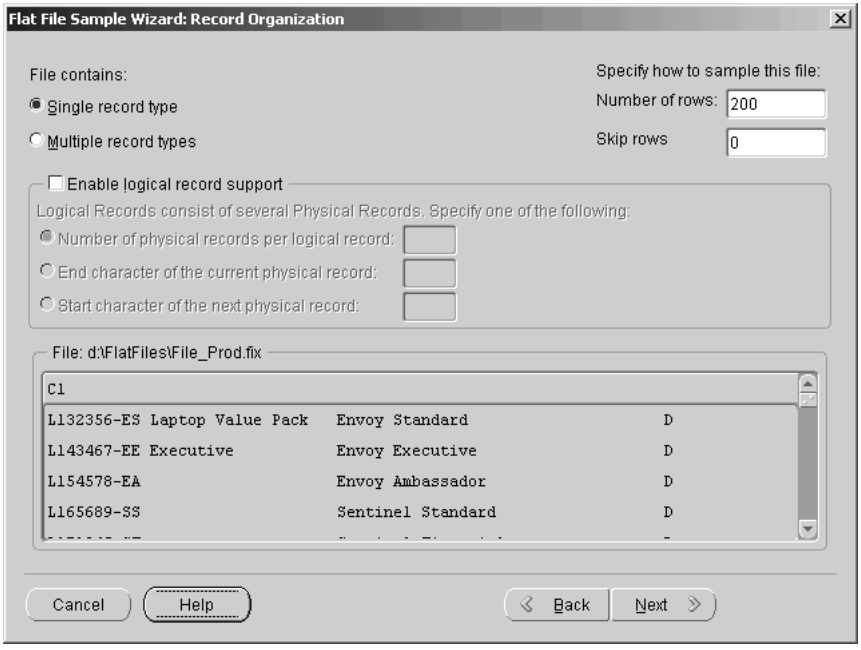


> **Note:** If the Warehouse Builder character set differs from the source, the sample might not be readable. If so, select the source character set from the drop-down list and Warehouse Builder translates the sample.

9. Click **Next.**

The wizard displays the Record Organization page. Use this page to specify whether the file contains single or multiple record types, or if it requires a logical record structure. You can also select how many rows of the file to sample.

Figure 5–23 Record Organization Page



10. Click **Next.**

The wizard displays the Column Definition page. Use this page to specify the column widths.

Figure 5–24 Column Definition Page

Define a column using one of two methods:

- Locate where the column ends in the sample and click that position on the ruler. The wizard displays a red tick mark on top of the ruler and marks the boundary with a red line.
- Specify the column width in the Field Widths space.

If you make a mistake, double-click the marker to restart.

Use the vertical and horizontal scroll bars to navigate.

11. Click **Next.**

    The wizard displays the Properties page. This page defines each field of the logical record just as you marked it off in the previous step.

12. Use the Properties page to define each of the logical record's fields:

    – **Name:** Can be changed.

      If your flat file supplies or contains field names, select the check box **Use the first record as the field names** to automatically transfer header information.

    – **Type:** Describes the source to SQL*Loader. Click the field data type and select a type from the drop-down list.

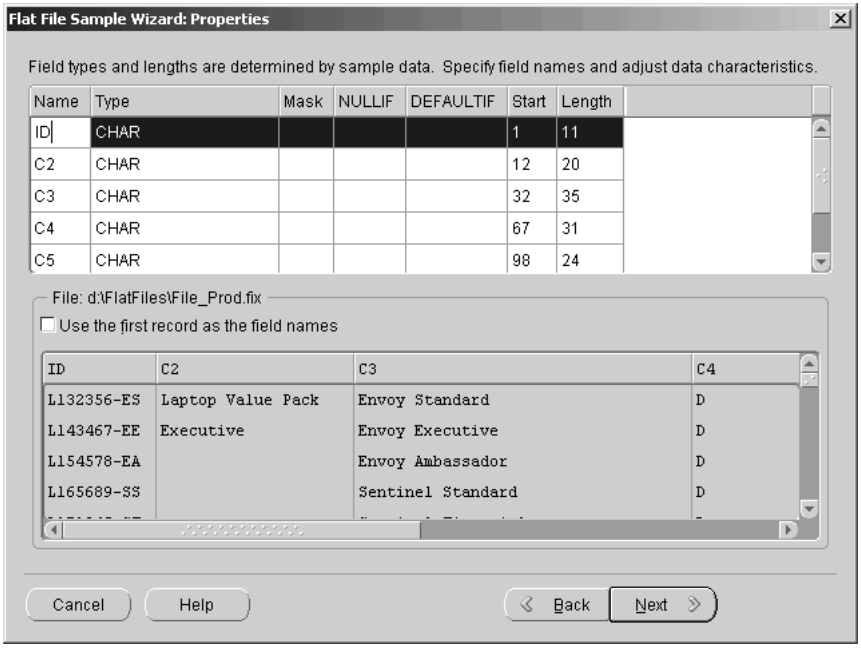
    – **Mask:** Overrides the default for DATE formats.

    – **NULLIF or DEFAULTIF:** Overrides the default SQL*Loader action for fields that contain all blanks. Specify DEFAULTIF or NULLIF and Warehouse Builder generates a corresponding DEFAULTIF=BLANKS or NULLIF=BLANKS condition.

    – **Length:** Specifies the length of the field.

    For more information on these fields, see "Creating Definitions for Flat File Sources" on page 5-30.

Figure 5–25 Properties Page



13. Click **Next.**

    Warehouse Builder displays the Summary page. Verify that the definition is correct. If not, click **Back** to navigate the wizard pages.

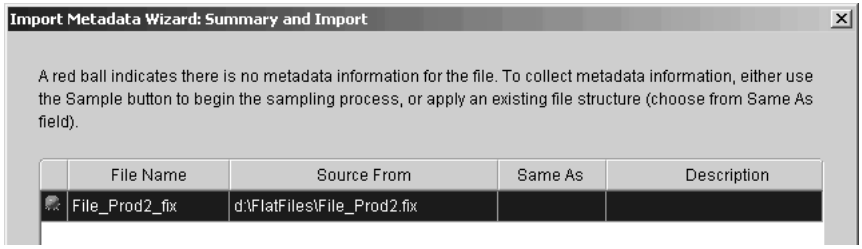
14. Click **Finish.**

    Warehouse Builder exits the Flat File Sample Wizard and returns to the Summary and Import page of the Import Object Wizard.

    The Summary and Import page has been updated:

    - The status ball is green.
    - The File Structure Name column now has an entry.

Figure 5–26 Summary and Import Page



If you open this wizard at a later time with a file that has the same format as the one you have just set, then you can select this entry from the Same As field instead of creating a new definition with the Flat File Sample Wizard.

15. Click **Finish.**

    Warehouse Builder creates a definition for file, stores the definition in the source module, and inserts the format's name in the source module's navigation tree.

Defining Multiple Records in a Fixed-Length File

You can define different record types within a file using the Flat File Sample Wizard.

To define multiple record types:

1. Open the Import Metadata Wizard, select a file, and then click **Sample** to open the Flat File Sample Wizard. See "Creating a Definition for a Fixed-Length File" on page 5-33.

   The wizard displays the welcome page for the Flat File Sample Wizard.

2. Click **Next.**

   The wizard displays the Setup page.

3. Select Fixed Length records.

4. Select a way to specify where each fixed-length record ends.

   - If you select **Records end at,** specify the code that terminated each record.
   - If you select **Record length (characters),** specify the number of characters in each record.

5. Optionally, select a different character set.

6. Click **Next.**

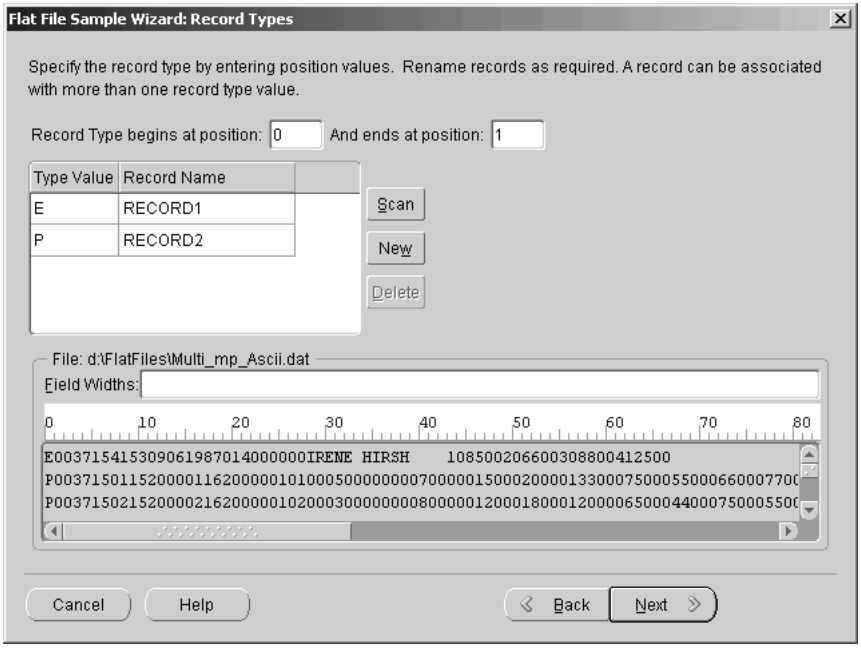
The wizard displays the Record Organization page.

7. Select **Multiple record types** and specify the number of rows to sample. Optionally, specify the number of rows to skip before sampling.
8. Click **Next.**

   The wizard displays the Record Types page.
9. Identify the column or columns that identify the record type in the file by using the Record Type begins at position field and the And ends at position field.

   In the following example, the first column defines the record type, so the first column begins in position 0 and ends in position 1.

Figure 5–27 Record Types Page



10. Click **Scan.**

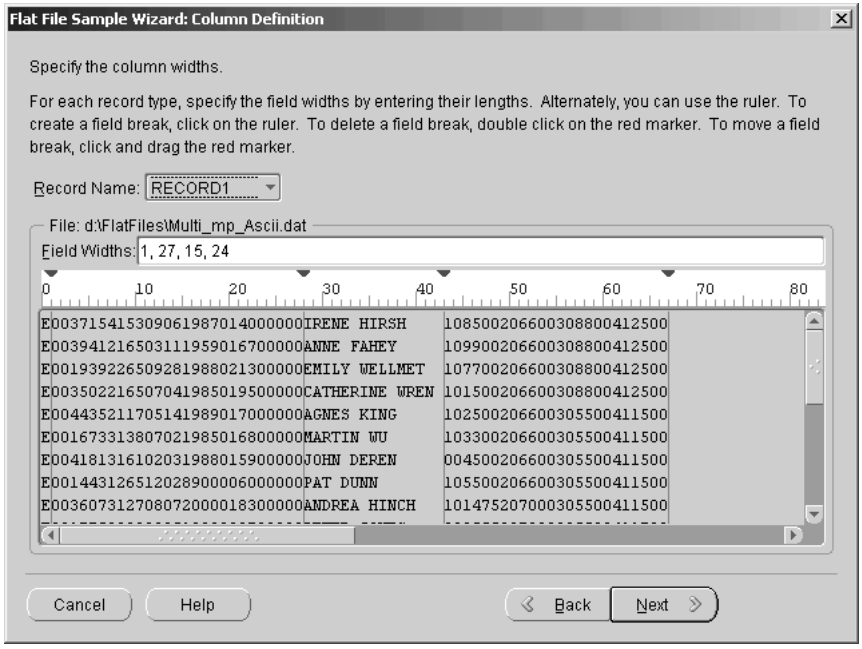
    A list of distinct type values appears with the default record names RECORD1, RECORD2, and so on. You can edit the record names, select a different record name for any type value, and add or delete type values.
11. Click **Next.**

The wizard displays the Column Definition page.

12. Select a record name and use the ruler to specify the length of a field, or specify the field length in numbers in Field Widths.

    The Field Width settings you define refer to the positions that contain the record type value. When you have adjusted the information for each column, either select another record name and define its columns or click **Next** to go to the Properties page.

Figure 5–28 Column Definition Page



You now need to define the data characteristics for each field, and adjust masking and constraints.

13. Adjust the data type, mask, and constraints (NULLIF, DEFAULTIF) as needed for each record type.

14. Click **Next.**

    The wizard displays the Summary page showing what is to be imported by the wizard.

15. Click **Finish.**

The source module now contains a definition for the file format, and within it, definitions for the individual records.

Specifying Logical Records

When the logical record for a source file contains multiple physical records, you must specify the physical records and describe the assembly method. Only fixed-length files with a single record type can have logical records that contain multiple physical records.

To describe multiple physical records:

1. Open the Import Metadata Wizard, select a file to describe, and then click **Sample** to open the Flat File Sample Wizard. See steps 1 - 5 in "Creating a Definition for a Fixed-Length File" on page 5-33.

   The wizard displays the welcome page for the Flat File Sample wizard.

2. Click **Next.**

   The wizard displays the Setup page. Use this page to define the global properties.

3. Select the **Fixed Length** radio button and complete the other selections as required.

4. Click **Next.**

   The wizard displays the Record Organization page.

Figure 5–29 Logical Record Support

5. Select the radio button that describes how the logical record is assembled:

   - Fixed number of physical records per logical record.

- Variable number of physical records with a continuation character at the end of each physical record that signifies that the record belongs with the next physical record.
- Variable number of physical records with a continuation character at the beginning of each physical record that signifies that the record belongs with the previous physical record.

The wizard updates the display of the logical record in the lower panel to reflect your selection. The default selection is one physical record per logical record.

6. You can now define the breaks for each field, click **Next,** and complete the definition for the format.

Creating a Definition for a Delimited File

The following procedure describes how to create a definition for a file using the Import Metadata and Flat File Sample Wizards. The location of the file was configured in a warehouse source module.

To create a definition for a delimited file format:

1. Select the warehouse module.
2. Right-click the module name and select **Import.**

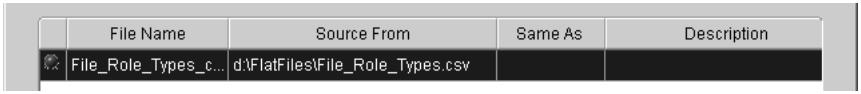
   Warehouse Builder displays the welcome page for the Import Metadata Wizard.
3. Click **Next.**

   The wizard displays the Filter Information page which you can use to filter the file names.
4. Click **Next.**

   The wizard displays the Object Selection page.
5. Move the file name from the Available to the Selected Objects list using the single arrow key.
6. Click **Next.**

   The wizard displays the Summary and Import page. The left-most column of this page contains a status ball that can be red or green. If green, then Warehouse Builder already has a definition of the file format. If red, you must create a definition for the file using the Flat File Sample Wizard.

Figure 5–30 Summary and Import Page Showing File Status



If the ball is red and you cannot describe the file with the format, you must create a definition for the file's format.

To create a format:

a. Click **Sample** at the bottom of the page.

The wizard displays the welcome page for the Flat File Sample Wizard.

b. Click **Next.**

The wizard opens the file, reads a sample of data, and displays the Setup page. This page displays a sample of data in a template with a few initial values set for the file's global properties.

- The Field Delimiter default is the comma (,).
- The Enclosures defaults are double quotation marks (") for both the left and right enclosures.

c. Select the file format as delimited and an NLS character set. See the discussion on NLS character sets on page 5-31.

Use the text box for the left and right enclosure characters to define text strings in the data. You can enter an enclosure character in the text box or select one from the drop-down list.

d. Click **Next.**

The wizard displays the Record Organization page.

e. Specify single record type and the number of rows to sample. You cannot enable logical record support for a delimited format file. For a file that has multiple record types, see "Defining Multiple Record Types in a Delimited File" on page 5-45.

f. Click **Next.**

The wizard displays the Properties page. Use this page to describe each field.

See the discussion for each field property in Step 12 on page 5-37.

g. Click **Next.**

The wizard displays the Summary page. Verify that the format definition is correct. If not, navigate the wizard pages by clicking **Back** and correct the definition.

h. Click **Finish.**

The Flat File Sample Wizard returns to the Summary and Object page of the Import Object Wizard.

The Summary and Object page has been updated. The status ball is now green and the File Structure Name column now has an entry.

Figure 5–31 Summary and Import Page Showing File Status

You can open this wizard at a later time and use the file format you created to describe any flat that has the properties described by this format. Instead of sampling the new file you can select this format from the **Same As** field.

7. Click **Finish.**

Warehouse Builder creates a definition for file, stores it in the source module, and inserts its name in the source module navigation tree.

Defining Multiple Record Types in a Delimited File

When a flat file contains several different types of records, you can use the scanning feature within the Flat File Sampling Wizard to search and label record types.

To associate multiple record types within a flat file:

1. Open the Import Metadata Wizard, select a file to describe, and then click **Sample** to open the Flat File Sample Wizard.

   The wizard displays the welcome page for the Flat File Sample Wizard.

2. Click **Next.**

   The wizard displays the Setup page. Use this page to define the file's global properties such as whether records are delimited by a character or space.

3. Click **Next.**

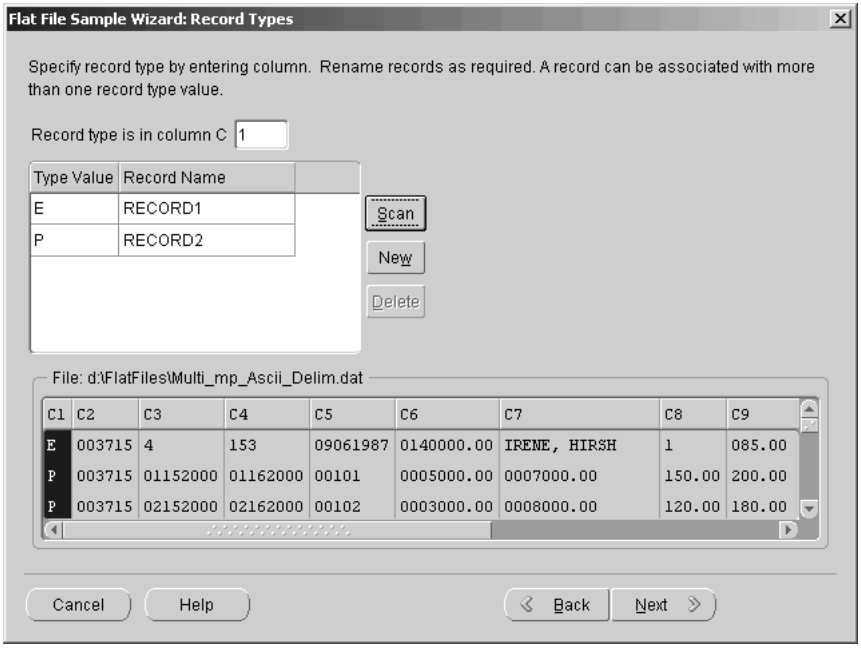
Define whether the file contains single or multiple record types.

Figure 5–32 Record Organization Page

4. Click **Next.**

   Selecting multiple record types opens the Record Types page.

Figure 5–33 Record Types Page



5. Identify the column that contains unique record information. For delimited files, the program assumes the column to scan is the first column of the record unless you specify a different column. Click **Scan.** All unique values appear.

   Records identified are named RECORD1, RECORD2, and so on. You can rename them by typing the new name in the field.

When you select a record type in the list, the lower panel shows data only for that record type.

6. Click **Next.**

   The wizard displays the Properties page.

7. For each record type, select its record name and adjust the data types, mask, and constraint information.

   See the detailed discussion for each field property on page 5-37.

8. Select the remaining record type and adjust the date types and other information for that record type. When you have the record type definition in the proper structure, click **Next.**

   The wizard processes your records.

9. The Summary page displays all record information to be imported.

10. Click **Finish** to import the file or click **Back** to return to a previous page to make changes.

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.

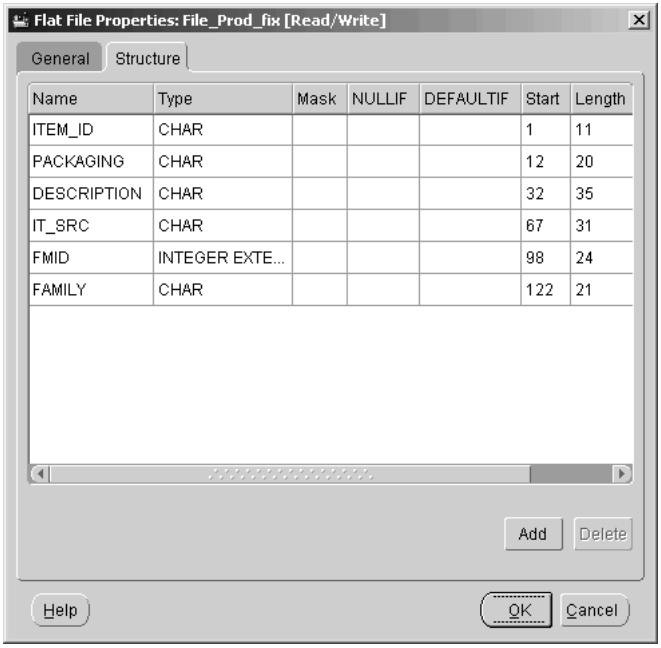
2. Right-click the file name and select **Properties.**

   Warehouse Builder displays the General tab of the Flat File property sheet. You can edit the name and description of the definition. You can also change the global properties ascribed to the file, such as the physical record size, the number of physical records per logical record, and the delimit and enclosure characters.

3. Select **Structure.**

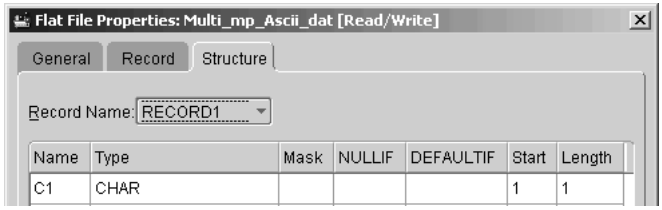
   Warehouse Builder displays the Structure page.

Figure 5–34 Structure Page



4. Use this page to:
   - Edit a field name, data type, mask.
   - Add a field mask.
   - Add a NULLIF condition.
   - Add a DEFAULTIF condition.
   - Add or delete a field.
5. After completing your changes, click **OK.**

Figure 5–35 Structure Page



6. If the file contains multiple record types, select **Record.**

   Warehouse Builder displays the Record sheet. You can edit the record type information, for example, you can delete a record type or add a new one.

Importing Definitions for Pure Extract and Pure Integrate

You can import definitions that describe source data that can be extracted using Pure Extract and Pure Integrate. These imported definitions can then be the subject of mappings defined within your project.

You can also export definitions with Pure Extract and import them into a Warehouse Builder project. For a complete description of how to use the Oracle Warehouse Builder MetaData Loader Utility, refer to "About the Warehouse Builder Metadata Loader" on page 11-2.

Importing Definitions for SAP Data Sources

You can import definitions from an SAP application and store them in an SAP source module in your project. See "Create a Source Module for SAP Definitions" on page 5-15 for instructions on how to create an SAP source module.

Business Components

SAP application systems logically group numerous data tables under different business components. Examples of business components are FI - Finance and CO - Controlling.

The business component filter on the Import Metadata Wizard Filter Information page enables you to filter metadata from a business area of your interest within the SAP application. To select the appropriate business component, you can navigate to the SAP business component tree, locate a specific business component, and view a list of tables organized under that component.

Import Metadata from SAP sources

To import metadata from an SAP application using the Import Metadata Wizard:

1. Fully expand the navigation tree for your project.
2. Right-click the name of your and select **Import** from the pop-up menu.

   Warehouse Builder displays the welcome page for the Import Metadata Wizard.
3. Click **Next.**

   The wizard displays the Filter Information page.
4. Use this page to select how you want to filter tables to be displayed in the Object Selection page.
   - Choose Business Component option if you want to filter the tables by business areas within the SAP application.

     When you click **Browse**, the wizard displays the SAP R/3 Business Component Hierarchy Dialog, a hierarchical navigation mechanism for locating data objects in SAP applications. After you select a business component from this dialog, your selection displays on the blank Business Component field on the import wizard page.
   - Choose the **Name matches** entry field or the **Description matches** entry field to enter a string and obtain matching tables from the SAP data source. The following rules apply:
     - Although the **Name matches** field is not case sensitive, the **Description matches** field is case sensitive.
     - You must enter a text string in the selected Text String entry field; it cannot be left empty.
     - You can create a filter for object selection by using the wildcard characters (%) for zero or more matching characters and (_) for a single matching character.
     - The SAP Integrator allows you to import metadata for transparent tables, cluster tables, or pool tables.
5. Specify the number of tables you want to import in the **Maximum number of objects displayed** field.
6. Click **Next.**

   The wizard displays the Object Selection page with a description of each table.

7. Select tables from the Available Objects list and move them to the Selected Objects list using the arrow buttons.

   For a description of the keys, see the example for a database source module in this chapter.

   Note: The SAP Integrator does not currently support importing definitions for views.

8. If the radio button for the foreign key level is set to One Level or All Levels, the Confirm Import Selection dialog appears.

   Note: If you select **All Levels**, you will import hundreds of tables that are related to each other through foreign key constraints.

9. Click **OK.**

   The selected objects appear in the right pane of the Object Selection page.

10. Click **Next.**

    The wizard imports definitions for the selected tables from the SAP Application Server and stores them in the *SAP* source module, then displays the Summary and Import page.

11. Review the information on the Summary and Import page.

    If you are re-importing metadata, the tables to be re-imported are displayed in bold. Click **Advanced Reconcile Options** to set your re-import options.

12. Click **Finish.**

    The SAP Integrator reads the table definitions from the SAP Application Server and creates the metadata objects in the Warehouse Builder repository.

6

Defining Source to Target Mappings

The following three chapters describe how to map data sources to targets. This chapter describes how to create and update mapping definitions. It also describes the Warehouse Builder Mapping Editor. The next chapter describes how to add operators and transformations to a mapping. The following chapter describes how to configure the logical and physical properties of a mapping, how to reconcile mapping operators with repository objects, and how to validate and generate the PL/SQL code used for deployment.

This chapter includes the following topics:

- Understanding Warehouse Builder Mappings
- Defining Mappings
- Editing Mapping Operator Attributes

Understanding Warehouse Builder Mappings

A mapping describes a series of operations that pulls data from sources, transforms it, and loads it into targets. When you create a mapping, you use operators to define the Extraction, Transformation, and Loading (ETL) operations that move data from a source object to a data warehouse target object. Mappings provide a visual representation of the flow of the data from sources to targets and the operations performed on the data. Define mappings using the Mapping Editor, Property Inspectors, Expression Builder, and Code Editor.

About Mapping Operators

All mappings are based on mapping operators. A mapping operator is a logical representation of a physical repository object. A mapping operator contains row sets. A row set is any set of zero or more rows of structured data brought into or emerging from a mapping operator. A row is the basic unit of data in a mapping. The number of rows in a row set is the cardinality of that row set.

A mapping operator defines how input row sets are manipulated to produce output row sets. A mapping operator can alter the cardinality of row sets. Warehouse Builder contains several mapping operators, each with its own purpose for processing row sets.

You define and edit mapping operators using the Mapping Editor and Property Sheets. The Mapping Editor contains a toolbox that visually represents the operators.

The operator types available within the Mapping Editor include:

- **Extract Operators (source):** Mapping Table, Mapping View, Mapping Materialized View, Mapping Sequence, Mapping Fact, Mapping Dimension, and Mapping Flat File.
- **Load Operators (target):** Mapping Table, Mapping Materialized View, Mapping Dimension, and Mapping Fact.
- **Standard Operators:** Aggregator, Pre- and Post-Mapping Processes, Filter, Joiner, Splitter, Sorter, Deduplicator, Set Operation.
- **Transformations**: Mapping Transformation, Expressions, Constants.
- **External Process:** Pure*Integrate, Pure*Extract, custom processes.

About Operator Properties

To specify the purpose of an operator, edit the property inspector for the operator. You can set properties at the following levels:

- **Operator:** Properties that affect the operator and what it does.
- **Group:** Properties that affect attributes within the group.
- **Attribute:** Properties that affect only operator or group attributes or their value.

About Attributes and Attribute Groups

Attributes are inputs and outputs for operators. Attributes belong to attribute groups. The group type determines the attribute type. An attribute group type can be an:

- Input
- Output
- Input/Output

Attribute and attribute group names are logical. Although the attribute names of the target object are often the same as the logical attribute names of the operator, their properties remain independent of the attributes of the operator to which they are connected. This protects any expression or use of an attribute from corruption if it is manipulated within the operator. You can rename attribute groups and attributes independent of their sources. The cardinality of attribute groups must match in order to be used in the same input group.

About Display Sets

A display set is a graphical representation of a group of operator attributes. By default, the All display set displays for all attribute groups. If an attribute group contains more than one display set, then you can select a different display set from the list on the View menu. You can define display sets for an attribute group in the mapping editor, or they can be inherited from the attribute sets of a warehouse object that are defined during the source and target definition phase of data warehouse design. See "Creating Attribute Sets" on page 3-16 for information on attribute sets.

You can see only one display set at a time for an attribute group. By default, the Mapping canvas shows the display sets with all the attributes in the respective attribute groups. You can edit display sets and define new display sets for an

operator in the Mapping Editor. See "Editing Operator Attributes" on page 6-23 for more information.

Table 6–1 describes the default attribute sets for Display Sets.

Table 6–1 The Default Attribute Sets

Attribute Set	Description
All	This is the default attribute set and includes all attributes.
Hierarchies	For each hierarchy in a Dimension, a display set containing all the level attributes in that hierarchy.

About Binding Mapping Operators

Mapping operators are independent from the repository objects they represent. When you bind a mapping operator to a repository object, you create a link between them. This link causes the operator to inherit the attributes and attribute sets of the repository object.

The Mapping Editor generates a default logical name for operators, which you can change at any time. Editing of bound operators is limited to changing the logical name and adding new attributes to the operator. Editing has no effect on the attributes of the physical repository object.

The bound name displays in the Bound Name property and in the tool tip for the mapping operator. The logical names used in the mapping for mapping operators and attributes can be different from the bound names. If you set your project preferences to logical name mode without propagation to physical names or if you define your mapping operator and attribute names in the Mapping Editor, then the logical names will be different. For more information, see "Using Inbound Reconciliation" on page 8-2.

If you choose to leave an operator unbound, it generates code that does not affect any repository object. You must bind the mapping operator to a repository object in order for the code to affect the repository object. After you bind a mapping object to a repository object, it cannot be unbound.

Defining Mappings

The first part of this section describes the Warehouse Builder Mapping Editor. The remainder of this section describes how to define a mapping.

About the Mapping Editor

The Mapping Editor is the interface for defining what you want to transform and map. The mapping editor uses an operator for each operation you want to perform. A series of operations defines a mapping.

The Mapping Editor includes:

- **Menu Bar:** Provides access to the mapping editor commands.
- **Toolbar:** Provides access to commonly used commands.

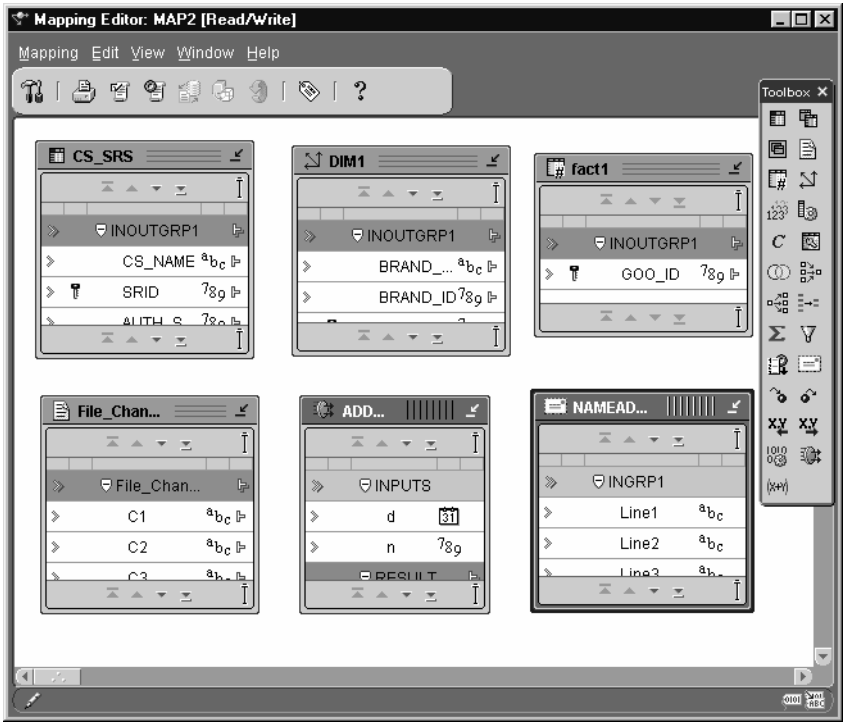
Figure 6–1 Mapping Editor Toolbar and Menu

- **Toolbox:** Contains mapping operator icons. To create a mapping, drag a mapping operator icon to the Mapping Editor canvas.

Figure 6–2 Toolbox

- **Mapping Editor Canvas:** Provides the white space where you can add and remove operators, link them, and edit their properties.

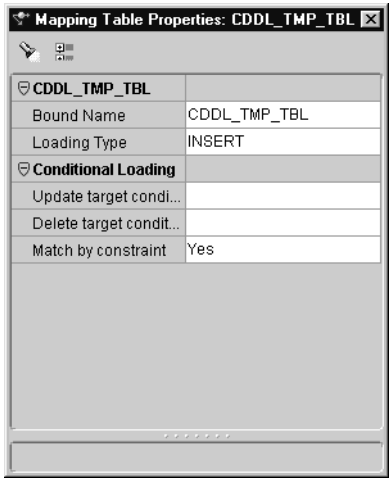
Figure 6–3 Mapping Editor Canvas Showing Mapping Operators



- **Operator Property Inspector:** Enables you to edit the properties of a mapping operator, attribute, or attribute group.

  When you open the property inspector, you can select another operator, attribute, or attribute group and the property inspector displays the property for the selected object. Only one property inspector can be open at a time.

Figure 6–4 Operator Property Inspector



If no operator, attribute group, or attribute is selected while a property inspector is open, the property inspector shows an empty list.

Creating a Mmapping

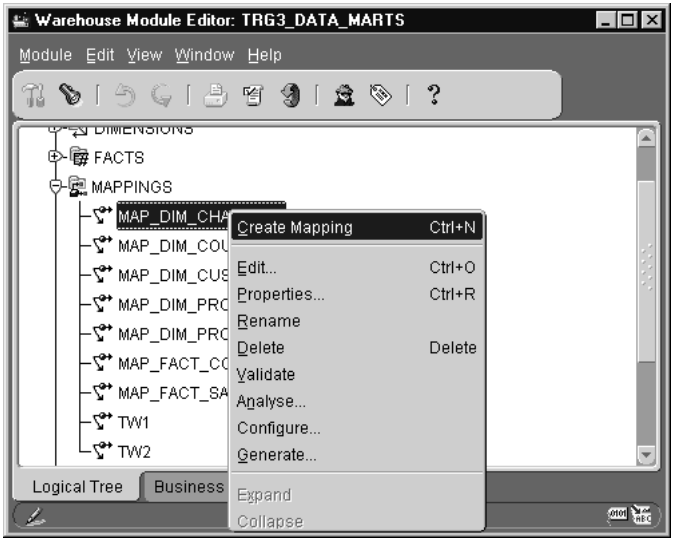
When you define a mapping object, you create a container that holds the mapping operators defining the ETL process. Use the Mapping Editor to define a mapping by:

1. Creating a mapping object in your warehouse module using the New Mapping Wizard.
2. Selecting data sources, data targets, operators, and transformations.
3. Linking the operator attributes.
4. Configuring operator attributes and properties. See "Configuring a Mapping" beginning on page 8-9 for more information.
5. Validating the generated code for the mapping. See "Validating and Generating a Mapping" on page 8-33 for more information.

To create a mapping object:

1. From the Warehouse Module Editor, right-click **Mappings** and then select **Create Mapping** from the pop-up menu.

Figure 6–5 Warehouse Module Editor

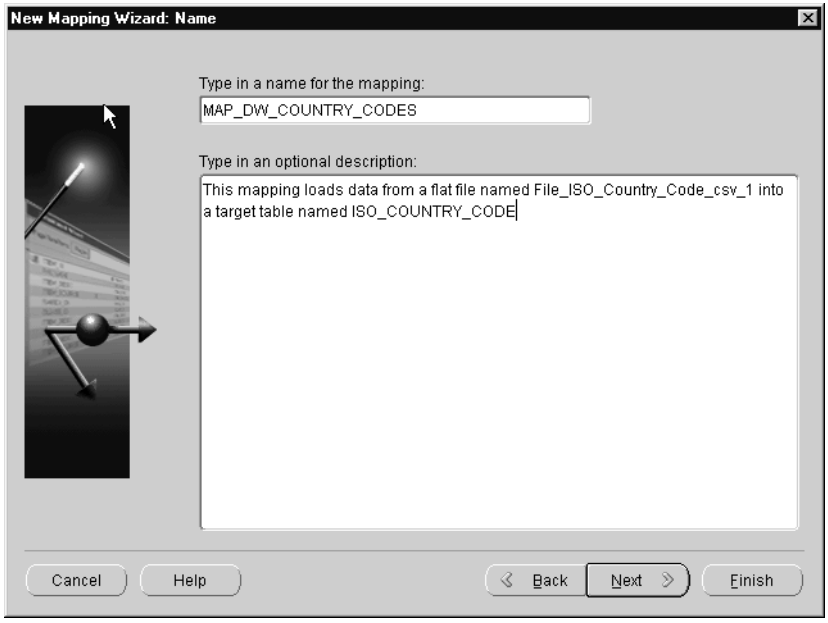


Warehouse Builder opens the New Mapping Wizard.

2. Click **Next.**

   The New Mapping Wizard displays the Name page.

Figure 6–6 New Mapping Wizard Name Page

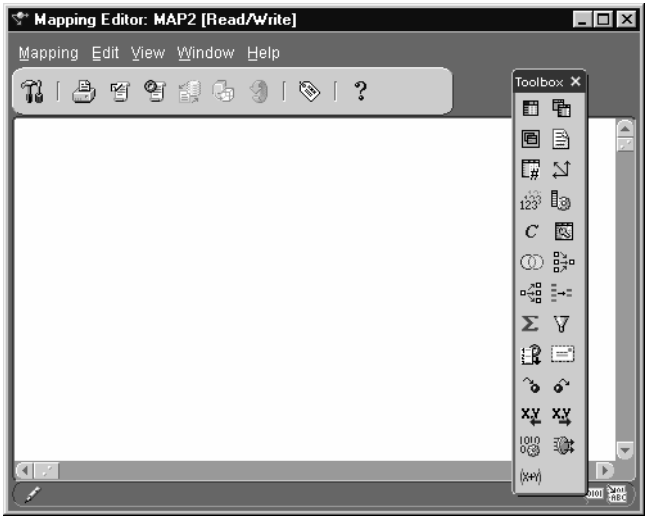


3. Enter a name and description for the new mapping. See "Object Names" on page 2-20 for information on naming.
4. Click **Next.**

   The New Mapping Wizard displays the Finish page. Verify the name of your new mapping.
5. Click **Finish.**

Warehouse Builder stores the definition for the mapping and inserts its name in the warehouse module navigation tree. The Mapping Editor opens.

Figure 6–7 Mapping Editor



Displaying the Mapping Editor

Use the Mapping Editor to define ETL operations. Open the Mapping Editor from the Warehouse Module Editor.

To display the Mapping Editor, do one of the following:

- Select a mapping and then from the **Edit** menu, select **Editor.**
- Double-click a mapping.
- Right-click a mapping, and select **Editor...** from the pop-up menu.

Selecting Data Operators

You can use a data operator as a data source or data target. Data operators include Mapping Tables, Mapping Dimensions, Mapping Facts, Mapping Views, and Mapping Materialized Views. Follow the same steps when you select any of these operators.

You can connect any data operator to any other mapping operator. A data operator generates a PL/SQL mapping.

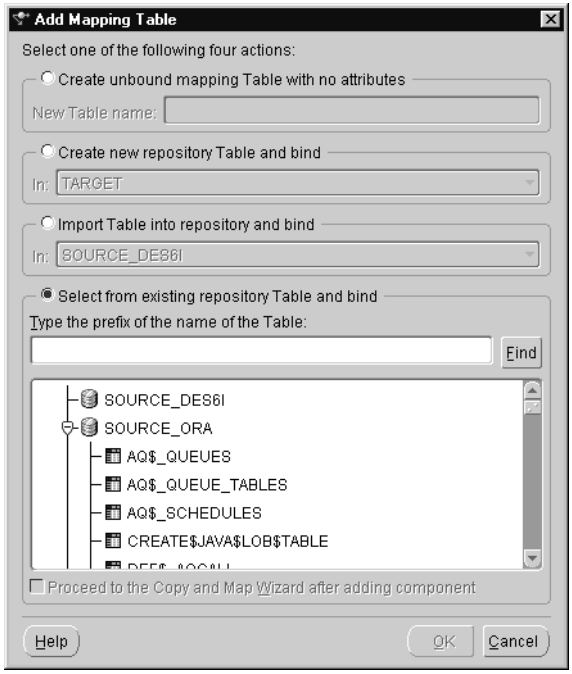
To select a table operator:

1. Open the Mapping Editor.

2. Drag a **Mapping Table** operator from the Toolbox and drop it onto the Mapping Editor canvas.

   The Add Mapping Table dialog displays.

Figure 6–8 Add Mapping Table Dialog



3. Select an option:

   - **Create unbound object with no attributes:** Enter a name for the new mapping object.
   - **Create new repository object and bind:** Select the warehouse module where you want the object to reside.
   - I**mport object into repository and bind:** Select the module from which you are importing the object.

     If you make this selection and click **OK** and have already defined a database link for the module from which you are importing the object, then Warehouse Builder returns the Database Link Information dialog. Enter the appropriate information in the dialog. If you have not defined a database

link, then Warehouse Builder displays the New Database Link Information dialog. See "Configuring Connection Information for Database Sources" on page 5-2 for information on how to define a new database link.

- **Select from existing repository object and bind:** Either type the prefix to search for the object or select from the displayed list of objects within the selected module.

Note: To select multiple items, hold down 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, hold down the Shift key, and then click the last object.

If you are selecting a Mapping Fact, a Mapping Dimension, or a Key Lookup operator, then you have two choices:

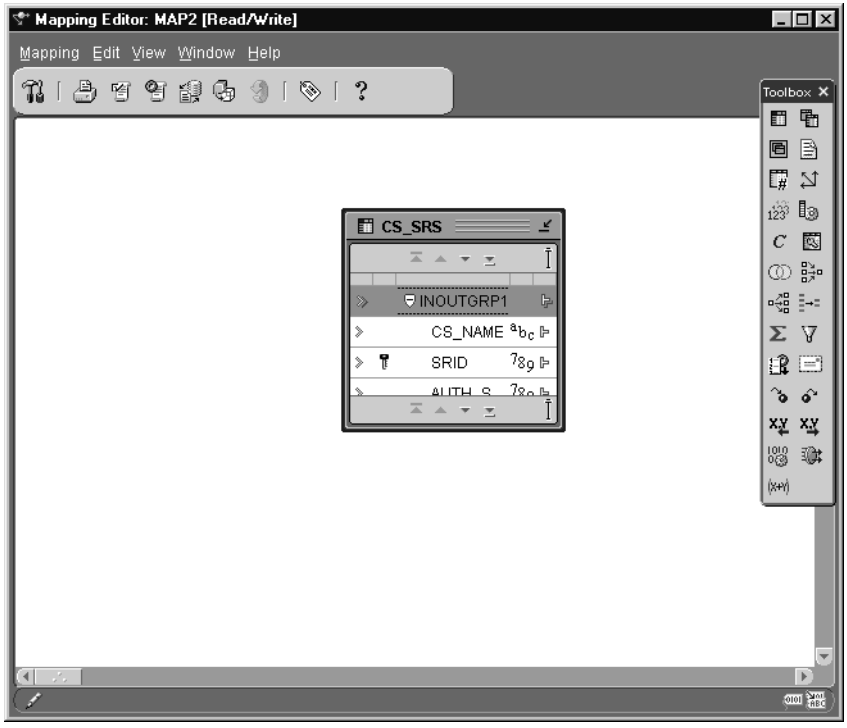
- Create a new repository object and bind.
- Select from existing repository object and bind.

You do not have the option to import an object if you are selecting a Mapping View or a Mapping Materialized View operator.

4. Click **OK.**

A Mapping Table operator displays on the canvas. This operator shows the names and attributes of the table to which it is bound.

Figure 6–9 Mapping Editor Showing a Mapping Table Operator Source



Selecting a Flat File Operator

You can use a Mapping Flat File operator as a data source or data target. As a data source, the Mapping Flat File operator acts as the row set generator for reading from a flat file.

To use a flat file as a target, a file source module must exist and it must contain valid flat files. A mapping to a flat file target generates a PL/SQL package that is spooled into a flat file instead of loading data into rows in a table. When you define a Mapping Flat File operator as a target, you are using an existing file as a template for the target flat file. Warehouse Builder comes with a file source module called TARGET_FILES for use as a template. This module contains a comma-delimited file (CSV_FILE) and a fixed-length file (FIXED_LENGTH_FILE), and each file contains ten fields of the CHAR data type.

A mapping can contain up to 50 flat files as targets, and it can also contain a mix of flat files, relational objects, and transformations. You can connect a Mapping Flat

File source operator to any other operator except another Mapping Flat File operator.

You have the following two options for a Mapping Flat File Operator:

- Import file into repository and bind.
- Select from existing repository file and bind.

Selecting an SAP Source

The following section describes how to select an SAP operator as a source for a mapping. An SAP operator used as a source can be connected to Mapping Table, Mapping Dimension, and Mapping Fact operators. You can generate ABAP or PL/SQL code for your mappings. If you generate ABAP code, only the Filter and Joiner operators are available.

To select an SAP source:

1. From the toolbox, drop the **Mapping Table** icon onto the Mapping Editor canvas.

   The Add Mapping Table dialog displays.

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 your source table name from this list and click **OK.**

   The editor places a mapping table on the mapping canvas to represent the SAP table.

When you select an SAP operator as a source, the option to **Proceed to the Copy and Map wizard after adding component** is enabled. If you check this option and click **OK,** Warehouse Builder displays the Copy and Map Wizard.

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, and map the source object to the new target object. The Copy and Map option is only available when you select a single table as a source.

To use the Copy and Map Wizard:

1. In the Add 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 Target Name page displays.

5. Enter 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.

     If the repository already contains an object with the same name, a name conflict error message displays. You must then rename the object you are creating.

   - **Object Type:** You can only create tables that map from SAP source tables. The choice in this field is preselected.

   - **Application:** The target application that contains the target object.

   - **Description:** Optional field used to describe the target object you are creating. Up to 2 MB is allowed.

6. Click **Next.**

   The Source Columns page displays.

7. Select the columns you want to copy from the source table to your target table.

   The fields on this page are read-only. By default, this page displays and selects all columns defined within the source table. To deselect a column, uncheck the selection check box next to that column. Click **Select All** to copy all the columns within the table. Click **Deselect All** to deselect all the columns within the table. You must select at least one column to copy.

8. Click **Next.**

   The Target Columns page displays. You 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 logical names of the columns. You cannot modify the logical names, you can only modify the physical names of a column. The default physical names are copied from the source physical names.

   Edit the following fields:

   - **Target Columns (Physical):** Change the physical name of a target column. 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.

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.

9. Click **Generate target physical name from source physical name (default)** to copy the physical target column names from the physical source column names. Click **Generate target physical name from source logical name** to copy the physical target column names from the logical source column names. The logical names can exceed the Oracle physical name limit of 30 characters.

   Note: If you rename a target column, then the new name is preserved regardless of how you choose to generate the target physical name. For example, the source physical name of a table is MANDT and its logical name is MANDT_Client. If you change the default target physical name from MANDT to MANDT_Client_Name, then this name is preserved.

   If the new name is not unique, a name conflict error message displays. Click **OK** to rename the column automatically or click **Cancel** to rename it yourself.

10. Click **Next.**

    The Summary and Copy page displays. Verify the information on this page.

11. Click **Finish.**

    The Mapping Editor displays the source table mapped to the newly created target table. You can locate this bound target table under the TABLES node in the Warehouse Module Editor.

Selecting a Data Flow Operator

You can add operators that alter the source data as it flows to the data targets. Warehouse Builder includes data flow operators in the Toolbox. You can create custom operators using the Oracle Transformation Library or Expression Builder.

See "Using Expression Builder" on page 7-2 for information on creating custom operators.

To select a data flow operator, drag a data flow operator from the Toolbox and drop it onto the Mapping Editor canvas. After you make your selections, a data flow operator displays on the canvas. Each data flow operator requires a different set of tasks when you add them to a mapping. Chapter 7, "Using Mapping Operators and Transformations" describes the flow operators and how to add them to a mapping.

Connecting Mapping Operators

After you have defined mapping source operators, data flow operators, and target operators, you are ready to connect them. You can connect individual operator attributes to each other, or you can connect attribute groups. When you connect groups, you can control which attributes are connected using the Auto-Mapping dialog.

To connect mapping operators, do one of the following:

- Drag a line from an output attribute group to an input attribute group of another operator.
- Drag a line from an output attribute to an input attribute of another operator.
- Drag a line from the source to the target to copy all selected attributes from the source node and add them to the selected target attribute.

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 connections you make in the mapping. If the mouse button is released over an invalid target, no data flow connection is established.

You cannot create the following links:

- Link an output attribute to an output attribute.
- Link an input attribute to an input attribute.
- Link attributes in the same operator.
- Link to the same input attribute twice.

Connecting Attributes

To connect mapping operator attributes, draw lines from output attributes or output attribute groups to input attributes or groups between the operators. These lines are

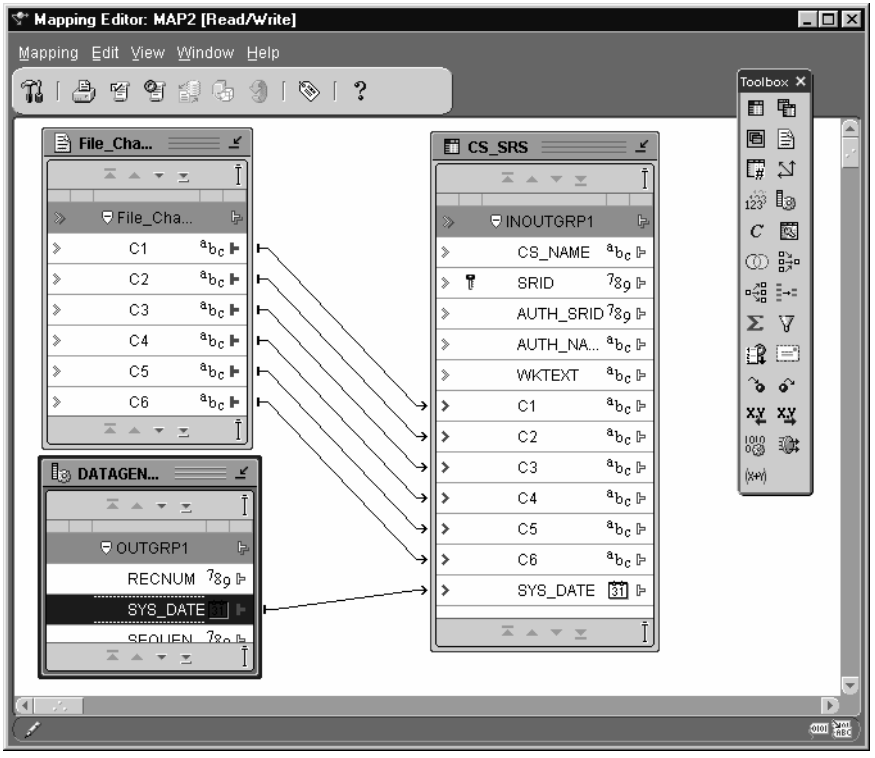
data flow connections. The data flow connections graphically represent how the data flows from a source to a target.

To connect operators:

1. Click and hold down the mouse button while the pointer is positioned over an output attribute.
2. Drag the mouse away from the output attribute and toward the input attribute to which you want data to flow.

   As you drag the mouse, a line appears on the Mapping Editor canvas to indicate a connection.
3. Release the mouse over the input attribute.

Figure 6–10 Connected Operators in a Mapping

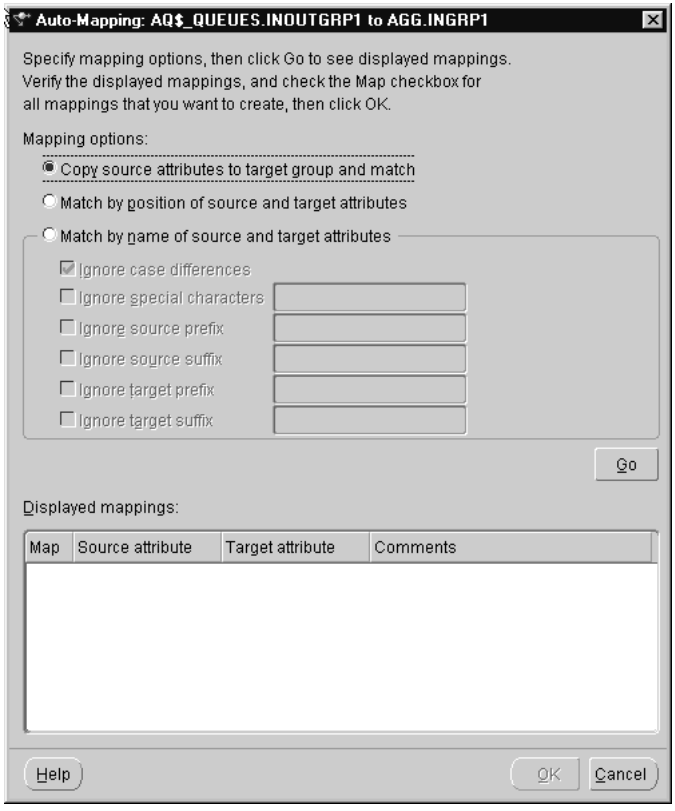


Repeat steps one through three until you have created all the data flow connections appropriate for your situation.

Connecting Attribute Groups

If you connect source attributes to target attribute groups, Warehouse Builder opens the Auto-Mapping dialog. Auto-mapping enables you to specify a rule that Warehouse Builder uses to automatically map source attributes to target attributes. After you specify Auto-Mapping options, Warehouse Builder creates the mapping.

Figure 6–11 Auto-Mapping Dialog



The Auto-Mapping dialog displays matching criteria and a panel that displays the mappings. Choose the match criteria by selecting:

Copy: Copies source attributes to the target group and creates mappings from the source attributes to the new target attributes.

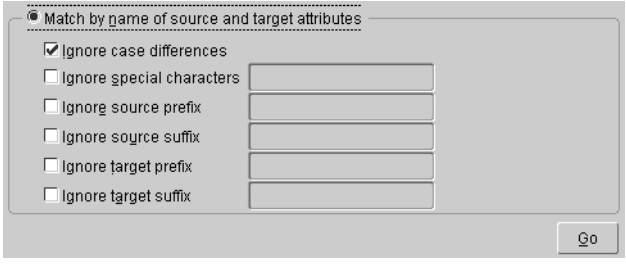
> **Note:** You cannot copy source attributes to dimension or fact targets.

Match by Position: Creates mappings 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: Creates mappings between existing attributes with matching names. 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 target prefix, Ignore target suffix:** These check boxes and text fields enable 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 6–12 Match by Name Matching Options

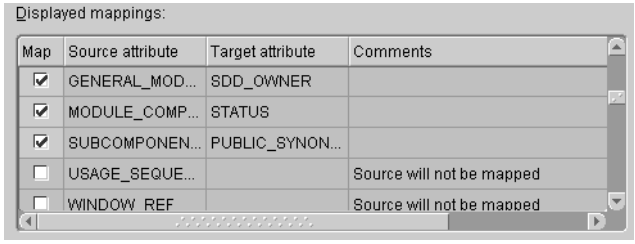


After you set the matching criteria, click **Go.**

If you attempt to map a source attribute 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 change the mappings before they are implemented.

Figure 6–13 Displayed Mappings Field



The check boxes for each attribute enable you to include or exclude attributes in a mapping. The source column lists the source attributes and the target column lists the matching target attributes. The comments column contains information about the results from your matching criteria. For example:

- When a matching results in more than one target attribute, the following messages appear in the comments column in the Displayed Mapping field:

  Target was already mapped: A target attribute can have only one mapping from one source attribute. The check box is unchecked and appears grayed out to indicate that the mapping will not be created.

  Target may not be double-mapped: If name matching finds multiple matches to a target attribute, only one mapping can be created. The check box for the first mapping is checked, and all other mappings to that target attribute are unchecked. You can select only one check box. 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 in the comments column in the Displayed Mappings field:

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.

Click **OK** to create the mapping.

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 "About Multi-User Access" on page 2-17 for more information on read-write permissions.

To complete the mapping, you may need to configure its operators. See "Configuring Mapping Table Operators" on page 8-9 for information.

Editing Mapping Operator Attributes

You can edit operator properties such as attributes, attribute groups, display sets, and names.

Adding or Removing Operator Attribute Groups

You can create a new attribute group in the mapping editor by right-clicking the name header and selecting **Add/Remove Groups.**

Figure 6–14 Add/Remove Groups Dialog

You cannot change the group direction. You add input groups to Joiners and output groups to Splitters. See "Adding Flow Operators to a Mapping" on page 7-4 for more information.

Note: To select multiple items from the selection box, hold down the Control key as you click each item. To select a group of items located in a series in the selection box, click the first object in your selection range, hold down the Shift key, and then click the last object in your selection range.

Editing Operator Attributes

For each mapping operator, you can add, remove, and rename attributes. After you have added or changed attributes or attribute groups, you must reconcile the mapping operators with their corresponding repository objects. See "Reconciling Mapping Operators with Repository Objects" on page 8-2 for more information.

Adding or Removing Attributes

You can add attributes to or remove attributes from an operator on the Mapping Editor canvas.

To add attributes to an Operator:

1. In the Mapping Editor, select a mapping operator.
2. Do one of the following:
   - From the **Edit** menu, select **Add/Remove Attribute.**
   - Right-click the operator display set and select **Add/Remove Attribute** from the pop-up menu.
   - Right-click an attribute group and select **Add/Remove Attribute** from the pop-up menu.

   The Add/Remove Attributes dialog displays.

Figure 6–15 Add/Remove Attributes Dialog

3. Type the new attribute name.
4. Click **Add.**
5. Click **OK.**

To remove attributes from an operator:

1. Select an attribute group in a mapping operator.
2. Do one of the following:
   - Select **Edit** and then **Add/Remove Attribute.**
   - Right-click the operator display set and then select **Add/Remove Attribute** from the pop-up menu.
   - Right-click the attribute group of the operator and select **Add/Remove Attribute** from the pop-up menu.

   The Add/Remove Attribute dialog displays.
3. Select the attribute you want to remove in the Add/Remove Attributes dialog.

   Note: To select multiple items from the selection box, hold down the Control key as you click each item. To select a group of items located in a series in the selection box, click the first object in your selection range, hold down the Shift key, and then click the last object in your selection range.

4. Click **Delete.**

Renaming Attributes

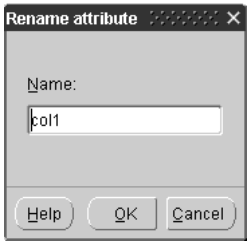
You can rename mapping operator attributes.

To rename an operator, attribute, or attribute group:

1. Select the appropriate item on the Mapping Editor canvas.
2. From the **Edit** menu, select **Rename** or right-click an attribute and select **Rename** from the pop-up menu.

   The Rename dialog displays.

Figure 6–16 Rename Attribute Dialog



3. Type the new name of the operator.
4. Click **OK.**

7

Using Mapping Operators and Transformations

This chapter describes how to use Expression Builder to create expressions that operate on the data as it flows from a source to a target. This chapter also describes data flow operators and how to add them to a mapping.

This chapter includes the following topics:

- Using Expression Builder
- Adding Flow Operators to a Mapping

Using Expression Builder

Most of the data flow operators described in this chapter require that you create expressions. An expression is a statement or clause that transforms data. These expressions can be portions of SQL that are used inline as part of a bigger SQL statement. They can also be in other languages such as PL/SQL, SAP, or SQL*Loader. 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 inspectors.

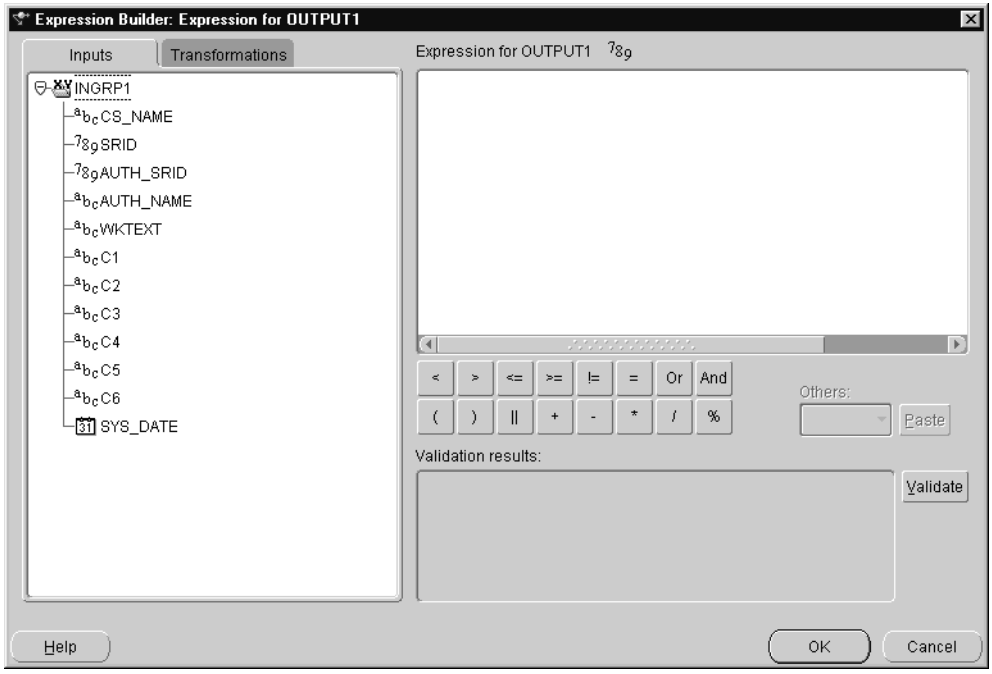
You can open Expression Builder from the operator properties inspectors in the following operators:

- Filter
- Joiner
- Splitter
- Aggregator

You can open Expression Builder from the attribute properties inspectors in the following operators:

- Expression
- Data Generator
- Constant

Figure 7–1 Expression Builder Interface



Expression Builder contains the following parts:

- **Navigation Tree:** A list that displays two tabs:
  - **Inputs:** A list of input parameters. Each operator determines the type of expressions it supports.
  - **Transformations:** A list of predefined functions and procedures located in the Oracle Transformation Library, the Global Shared Library, and a custom Transformation Library. See "About Oracle Transformation Libraries" on page 4-28 for more information.
- **Expression Field:** Use this field to type and edit expressions.
- **Operator Buttons:** Use these buttons to build an expression without typing. The available operators vary by the active data flow operator.
- **Others:** A drop-down list of available SQL clauses that are appropriate for the active expression type.

- **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 Expression Builder are limited to the operator inputs and to any transformations available in a project. This limitation protects the expression from becoming invalid because of changes external to the operator. If the deployment database is different from the design repository, it may not accept the expression. If this happens, the expression may be valid but incorrect against the database. In this case, expression errors can only be found at deployment time.
- **Validation Results Field:** This field displays the results of a validation.

To use Expression Builder:

1. Click the **...** button in the expression field located in the operator properties inspector or the attribute properties inspector.

   The Expression Builder displays.

2. Create an expression.

   You can create an expression by:

   - Typing it into the expression field.
   - Dragging predefined functions available from the navigation tree list and dropping them into the expression field.
   - Clicking operator buttons available below the expression field.

3. Click **Validate.**

   This verifies the accuracy of the expression syntax before you save the expression and close the editor.

4. Click **OK.**

See "About Mapping Operators" on page 6-2 for more information on operators.

Adding Flow Operators to a Mapping

This section describes how to add flow operators to a mapping. Flow operators control the data that moves from sources to targets. They also enable you to change your data as appropriate once it reaches the data target.

To add a flow operator to a mapping:

1. Drag and drop an icon from the Toolbox onto the Mapping Editor canvas.
2. Define attributes, procedures, or functions.
3. Connect the data flow operator to a data source or to a data target.

Before you add data flow operators, you add data source and target operators to your mapping. See "Defining Mappings" on page 6-4 for more information.

Adding Mapping Sequences

A Mapping Sequence operator generates sequential numbers that increment for each row. You can connect a Mapping Sequence operator 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 attribute set named OUTGRP containing two output attributes: CURRVAL and NEXTVAL. NEXTVAL generates a row set of consecutively incremented numbers beginning with the next value and CURRVAL generates from the current value.

You must bind a Mapping Sequence to a repository sequence in one of the modules. You must also reconcile it to that object. The repository sequence must be generated and deployed before the map is deployed to avoid errors in the generated code package for the mapping. See "About Binding Mapping Operators" on page 6-4 for more information.

You generate a mapping with a sequence in 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 "Setting Runtime Parameters" on page 8-19 for more information on configuring mode settings.

The Mapping Sequence operator contains the following property:

- **Bound Name (operator level):** 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 add a mapping sequence operator to a mapping:

1. Drop the **Mapping Sequence** operator onto the Mapping Editor canvas.

   The Add Mapping Sequence dialog displays.

2. Click one of the options.
   - Create a new repository sequence and bind.
   - Import sequence into repository and bind.
   - Select from existing repository sequence and bind.

     This selection contains a search text box and a tree for the sequences stored in the Warehouse Builder repository. Click the appropriate node to locate a sequence. Double-click a sequence to select it.

   For more information on these options, see "Selecting Data Operators" on page 6-10.
3. Connect the sequence to a target attribute.
4. From the **Mapping** menu, select **Generate** and then **Mapping.**

   The Code Viewer displays generated code for the mapping. The sequence appears in the SELECT list for the insert DML.

Adding Data Generators

Use a Data Generator operator to provide information such as record number, system date, and sequence values. It also provides a place to enter constant information. 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. Because RECNUM is incremented 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, it can be accessed only in that format. If the system date is loaded into a DATE column, then it can be accessed in a variety of formats that include 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 7–1 lists the options available for sequence values.

Table 7–1 Sequence Value Options

Value	Description
column_name	The name of the column in the database 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. Modification of these attributes is not recommended but you can create new attributes. The Data Generator operator is only valid for a SQL*Loader mapping. There can only be one Data Generator operator for a mapping.

The Data Generator contains the following property:

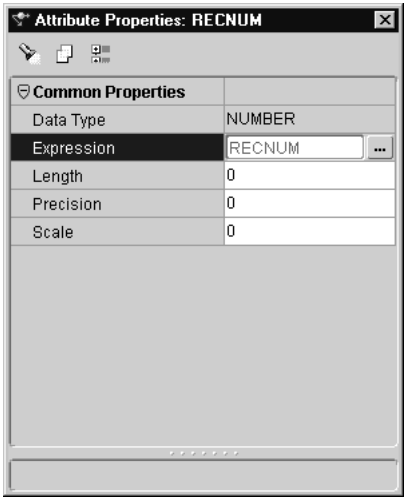
- **Expression (common attribute property):** Expression to use when this attribute is mapped. You must ensure the value entered for the expression is valid SQL*Loader syntax in the context used.

To add a data generator to a mapping:

1. Drop a **Data Generator** operator onto the Mapping Editor canvas.
2. Right-click the **RECNUM** attribute and select **Attribute Properties...** from the pop-up menu.

   The RECNUM Attribute Properties inspector displays.

Figure 7–2 RECNUM Attribute Properties Inspector



3. Enter an expression for RECNUM in the **Expression** field or click **...** to open Expression Builder and define an expression.
4. Repeat the previous steps for the SYSDATE and SEQUENCE attributes, if necessary.

Adding Constants

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 as inputs to a pre-mapping process, a post-mapping process, a mapping output parameter, and other operators in the data flow.

The Constant operator can have only one output attribute group. If the constant defined is of data type VARCHAR or VARCHAR2, the expression must be a valid SQL expression returning a value of data type VARCHAR or VARCHAR2. Constant string literals must be enclosed within single quotes, for example, `'my_string'`.

The Constant operator contains the following property:

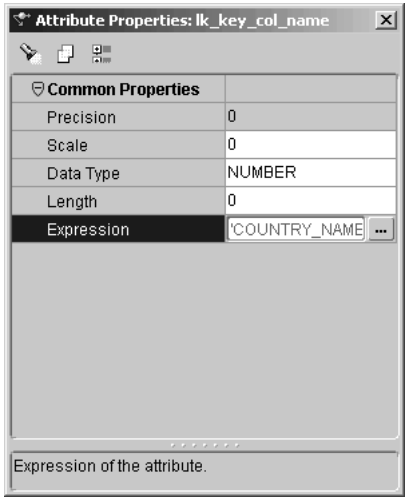
- **Expression (attribute level):** Defines the constant value represented by the attribute.

To add a constant operator:

1. Drop a **Constant** operator onto the Mapping Editor canvas.
2. Create an output attribute in the Constant operator output attribute group.
3. Right-click the attribute and select **Attribute Properties** from the pop-up menu.

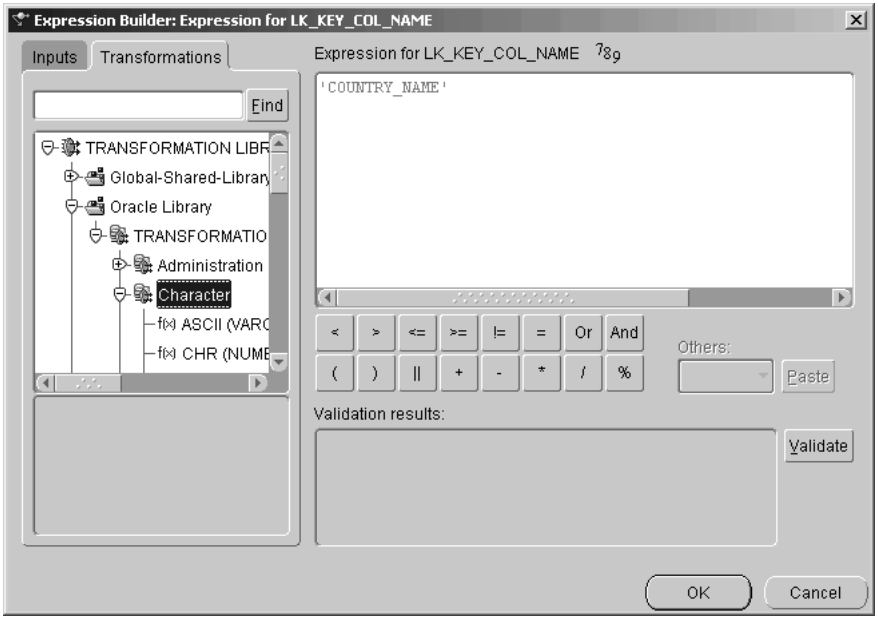
   The Attributes Properties inspector displays.

Figure 7–3 Attribute Properties Inspector



4. Enter an expression into the Expression field or click **...** to define an expression using Expression Builder.

Figure 7–4 Expression Builder Showing A Constant



5. Close the attribute property editor.
6. Connect the output attribute to the appropriate target attribute.
7. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code.

Using Key Lookups

The Key Lookup operator enables you to perform a lookup of data from a lookup object such as a table, view, fact, or dimension. The lookup object is bound to the Key Lookup operator. Use the Key Lookup operator if you have defined surrogate keys on warehouse objects such as dimensions. 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. 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 Mapping Operators with Repository Objects" beginning on page 8-2 for more information on reconciliation.

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.

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 Oracle Server performs implicit conversions, the results may not be what you want and may cause a runtime error.

To add a Key Lookup operator:

1. Drop a **Key Lookup** operator onto the Mapping Editor canvas.

   The Add Mapping Key Lookup dialog displays.

2. Click one of the following options:

   - Create a new repository Key Lookup and bind.
   - Select from existing repository Key Lookup and bind.

     This selection contains a search text box and a directory tree for the tables that can be Key Lookup entities. Click the appropriate node to locate a table. Double-click a table to select it.

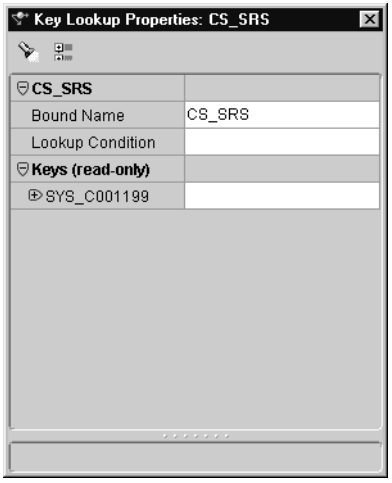
   A Key Lookup operator displays.

Note: To select multiple items, hold down the Control key as you click each item you want to select. To select a group of items located in a series, click the first object in your selection range, hold down the Shift key, and then click the last object.

3. Connect the source operator attributes to the input attribute group of the Key Lookup operator.
4. Right-click the Key Lookup operator header and select **Operator Properties** from the pop-up menu.

   The Key Lookup Properties inspector displays.

Figure 7–5 Key Lookup Properties Inspector



5. Click the field to the right of the Lookup Condition and click the **...** button.

   The Lookup Condition dialog displays.

Figure 7–6 Lookup Condition Dialog

6. Select the lookup entity attribute from the **Lookup Table Column or Key** drop-down list.

   Choose attributes to compare to the selected lookup table column.

   For a non-composite key:

   a. Select an Input Attribute from the drop-down list.

   b. Click **Add to List.**

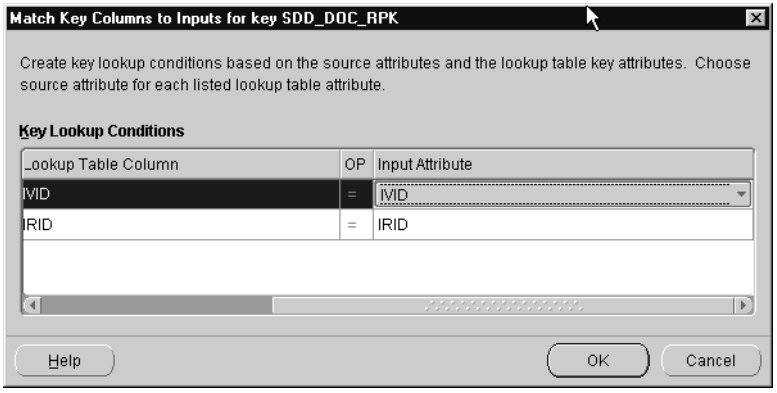
      The column or input pairs are added to the table at the bottom of the main dialog.

   For a composite key:

   a. Click **Add to List.**

      The Match Key Columns to Input for Key dialog displays.

Figure 7–7 Match Key Columns to Input for Key Dialog



b. Select the key input attributes from the **Input Attribute** drop-down list.

c. Click **OK.**

7. Click **OK** to close the Lookup Condition dialog.
8. Close the Key Lookup Properties dialog.

Using Set Operations

The Set Operation operator enables you to use set operations in a mapping. The operations are:

- Union (default)
- Union All
- Intersect
- Minus

The Set Operation operator contains two input groups and one output group. You can add input groups by connecting source operator attributes to the Set Operation operator. Mapping attributes to a Set Operation operator 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 attributes.

To execute 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.

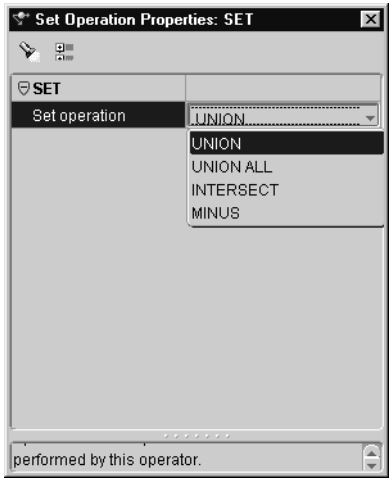
Apply the set operation in top-down order. The order of the input groups determines the execution order of the set operation. This order affects only the minus operation. For example, A minus B is not the same as B minus A. The order of the attributes within an input group determines the structure of a set. For example, {empno, ename} is not the same as {ename, empno}.

To create sets of data:

1. Drop a **Set Operation** operator onto the Mapping Editor canvas.
2. Connect source operator attributes to the Set Operation operator attribute groups.
3. Right-click the operator header and select **Operator Properties...** from the pop-up menu.

   The Set Operation Properties inspector displays.

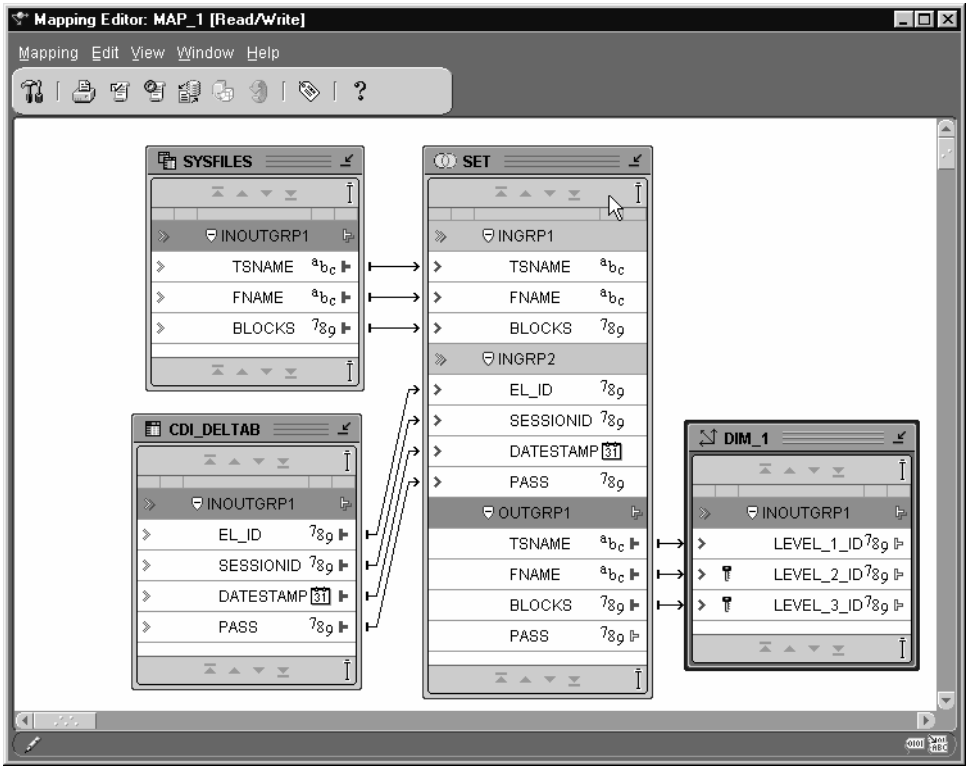
Figure 7–8 Set Operation Properties Inspector



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 inspector.
6. Connect the Set Operation operator OUTGRP1 attribute group to a target operator input attribute group.

Figure 7–9 Mapping Showing the Set Operation Operator



7. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code.

Joining Data Sources

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 expression 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. Connect data sources directly to a Joiner.

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.

The following can help you generate queries with optimal performance:

- Write a join condition statement. You can leave the join condition property for a Joiner operator blank. A blank condition does not generate a validation error in the Expression Builder. By leaving the join condition blank, the Joiner operator brings in all of the source columns despite the number of columns mapped.
- Define a join condition that relates columns that are not unique or foreign keys in the source table. In this case, create an index on the column in the source row set before attempting to run the generated map.
- Create a stage table with a column for the transformed column, populate the stage table with another map, and then join the stage table with the other sources. The join condition can contain transformation function calls but this method slows the performance of the generated query.
- Create a stage table with the transformed data and join the stage table with another source. The join condition can contain arithmetic operators, for example, `LINES.PO_OR_ORDER_ID + 10 = ORD.ORDER_ID`. All values from the source table must be read and transformed to find the joined rows for a given row.

The join condition expression cannot contain aggregation functions, such as SUM. Compile errors result when deploying the generated code for the map.

The Joiner can have an unlimited number of input groups and one output group.

The Joiner operator contains the following properties:

- **Join Condition (operator level):** 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.
- **Data Type (attributes):** The data type of the attribute.

- **Precision (attributes):** The precision of the attribute, used for numeric type attributes only.
- **Scale (attributes):** The scale of the attribute, used for numeric type attributes only.
- **Length (attributes):** The length of the attributes, used for string type attributes only.

To add a Joiner to a mapping:

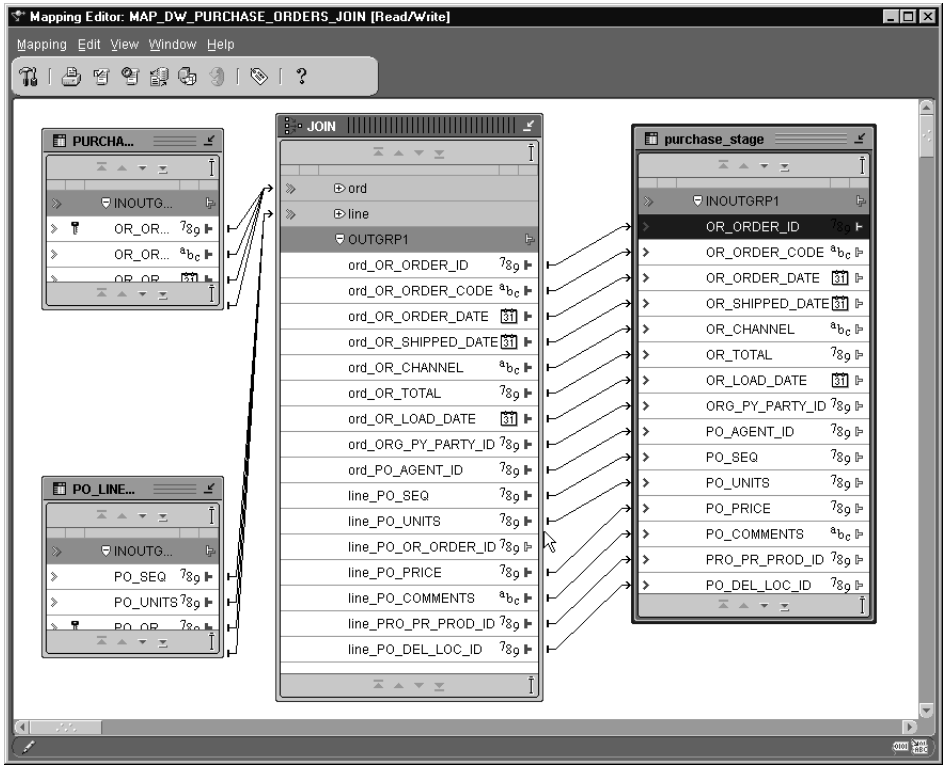
1. Drop the **Joiner** operator onto the Mapping Editor canvas.
2. Connect an attribute group from the first source to the INGRP1 group of the Joiner operator.

   The output attributes are created with data types matching the corresponding input data types.

   Tip: Rename the input groups on the Joiner operator to reflect their function.

3. Connect an attribute group from the second source operator to the INGRP2 group of the Joiner operator.

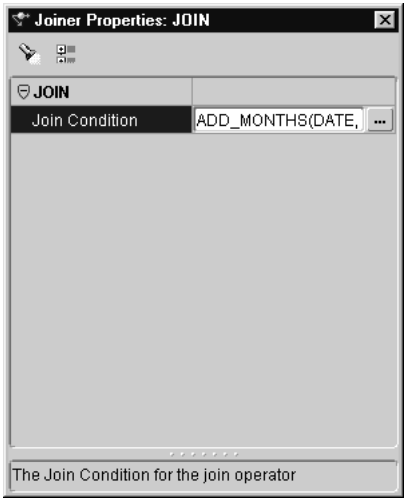
Figure 7–10 Mapping Editor Showing a Joiner Operator



4. Right-click the Joiner operator header and select **Operator Properties** from the pop-up menu.

The Joiner Properties inspector displays.

Figure 7–11 Joiner Properties Inspector



5. Enter a join condition in the Join Condition field or click **...** to define an expression using Expression Builder.
6. Close the Joiner Properties inspector.
7. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code. The join condition appears in the WHERE clause of the SELECT query for the insert DML.

Creating Full Outer Join Conditions

In an equijoin, key values from the two tables must match. In a full outer join, key values are matched and nulls are created in the resulting table for key values that cannot be matched. A left or a right outer join retains all rows in the specified table.

In Oracle8*i*, you create an outer join in SQL using the join condition variable (+):

```
SELECT ...
FROM A, B
WHERE A.key = B.key (+);
```

This example is a left outer join. Rows from table A are included in the joined result even though no rows from table B match them. To create a full outer join in Oracle8*i*, you must use multiple SQL statements.

The Expression Builder allows the following syntax for a full outer join:

```
TABLE1.COL1 (+) = TABLE2.COL2 (+)
```

This structure is not supported by Oracle8*i*. Oracle9*i* is ANSI SQL 1999 compliant. The ANSI SQL 1999 standard includes a solution syntax for performing full outer joins. The code generator translates the above expression into an ANSI SQL 1999 full outer join statement, similar to:

```
SELECT ...
FROM table1 FULL OUTER JOIN table2 ON (table1.col1 = table2.col2)
```

Because the full outer join statement complies to ANSI SQL 1999, it is only valid if the generated code is deployed to an Oracle9*i* database. Specifying a full outer join to an Oracle8*i* database results in a validation error.

A full outer join and a partial outer join cannot be used in one SQL statement. If a full outer join and partial outer join are used in the same SQL statement, a validation error occurs.

To use a full outer join in a mapping:

1. Follow steps one through four on page 7-18 for adding a join operator.
2. Enter a full outer join statement in the Join Condition field or click **...** to define an expression using Expression Builder.
3. Close the Joiner Operator Property inspector.

Splitting Data

You can use the Splitter operator to split 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.

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 (output attribute groups):** The text expression template for the Split Condition. For code generation, the actual source columns are substituted for the input attribute names in the expression template. Generally, the expression is a valid SQL expression that can be used in a WHERE clause.
- **Data Type (attributes):** The data type of the attribute.

- **Precision (attributes):** The precision of the attribute, used for numeric type attributes only.
- **Scale (attributes):** The scale of the attribute, used for numeric type attributes only.
- **Length (attributes):** The length of the attributes, used for string-type attributes only.

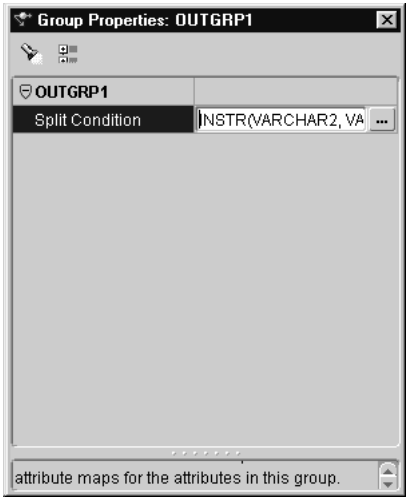
To split data from one source to several targets:

1. Drop the **Splitter** operator onto the Mapping Editor canvas.
2. Connect an attribute group from a source operator to the INGRP1 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** from the pop-up menu.

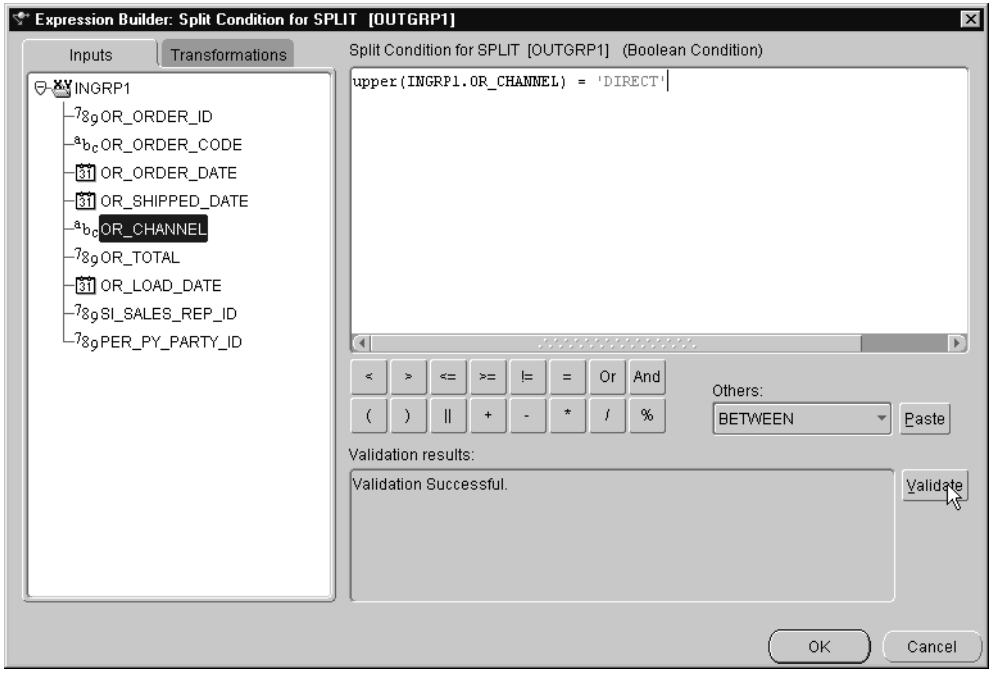
   The Splitter Properties inspector displays.

Figure 7–12 Group Properties Inspector for a Split Condition



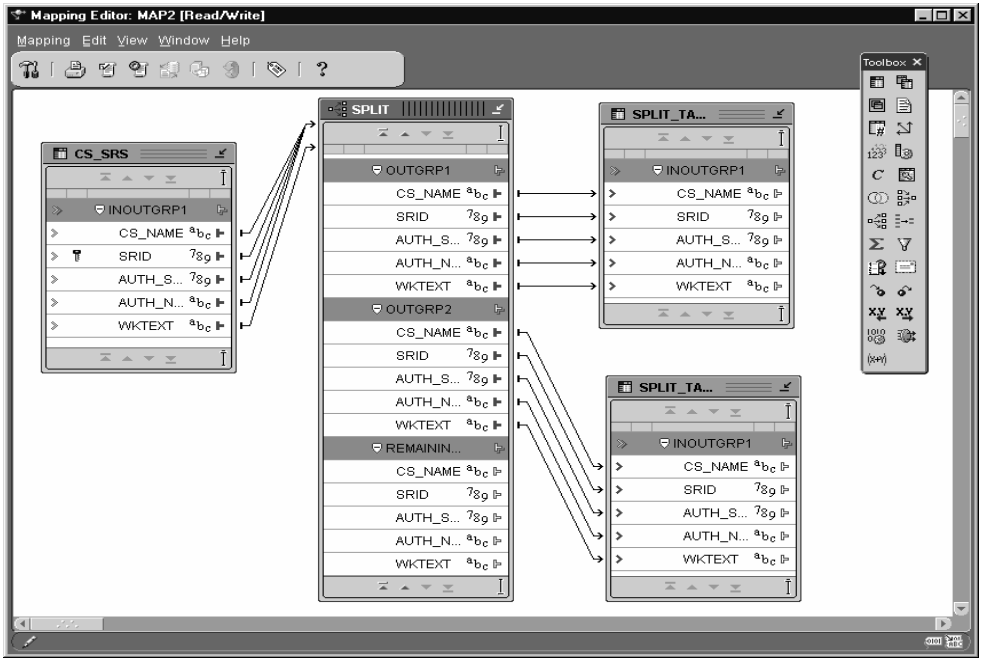
4. Enter an expression in the Split Condition field or click **...** to define an expression using Expression Builder.

Figure 7–13 Expression Builder Showing A Split Condition



5. Close the Splitter Properties inspector.
6. Define expressions for each of the output groups.
7. Connect the output groups to the targets.

Figure 7–14 Splitter Operator Mapping



8. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code. The generated code displays the split conditions in the WHERE clauses of the SELECT queries for the insert DML. There is a separate select query for each target.

Removing Redundant Data

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 redundant data:

1. Drop the **Deduplicator** operator onto the Mapping Editor canvas.
2. Connect the attributes from the source operator to the attribute group INOUTGRP1 of the Deduplicator operator.
3. Connect the attributes from INOUTGRP1 of the Deduplicator operator to the attributes of the target operator.

4. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code with a DISTINCT clause in the select statement.

Aggregating Data

The Aggregator operator performs data aggregations, such as SUM and AVG, and provides an output row set with aggregated data. Because each Aggregator operator shares a GROUP BY and HAVING clause, each attribute in the output attribute 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 attribute group and one output attribute group. Connecting the source to the input attribute group produces the corresponding aggregated row set in the output attribute group.

The Aggregator operator contains the following properties:

- **Group By Clause (operator level):** 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 attribute group specifies how this grouping is performed.
- **Having Clause (operator level):** A boolean condition restricting the groups of rows returned in the output attribute 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 attribute group. This clause can refer to any attributes or expression of the attributes in the input attribute group.
- **Expression (attribute level):** Defines the aggregation functions to be performed on the attribute. If no aggregation function is necessary, specify NONE for the function.

To add an aggregator to a mapping:

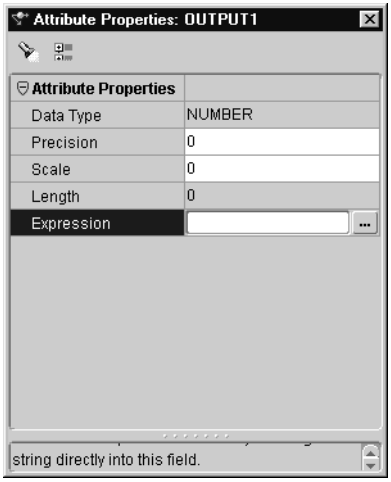
1. Drop an **Aggregator** operator onto the Mapping Editor canvas.
2. Connect source operator attributes to the INGRP1 of the Aggregator operator.
3. Add the appropriate attributes to the OUTGRP1 attribute group.

   See "Adding or Removing Operator Attribute Groups" on page 6-22 for information.

4. Right-click on an attribute in the Aggregator operator and select **Attribute Properties** from the pop-up menu.

   The Attribute Properties inspector displays.

Figure 7–15 Attribute Properties Inspector



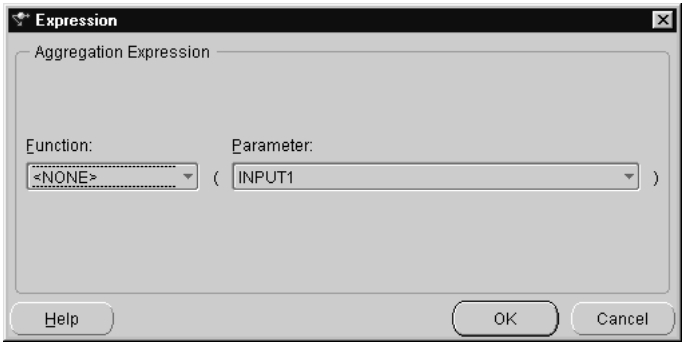
5. Define expressions for each output attribute.

   a. Click the **...** button to the right of the Expression property.

      The Expression dialog displays.

   b. Select a **Function** and a **Parameter** from the drop-down lists.

Figure 7–16 Expression Dialog

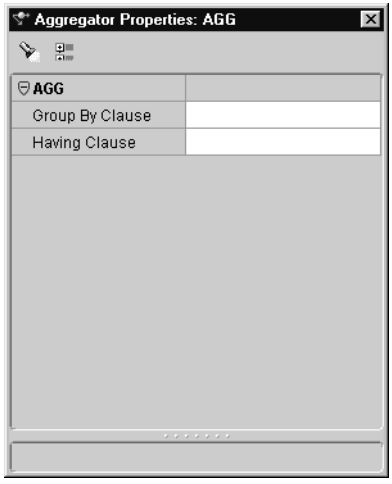


   c. Click **OK.**

6. Click the title bar of the Aggregator operator.

   The Aggregator Properties inspector displays.

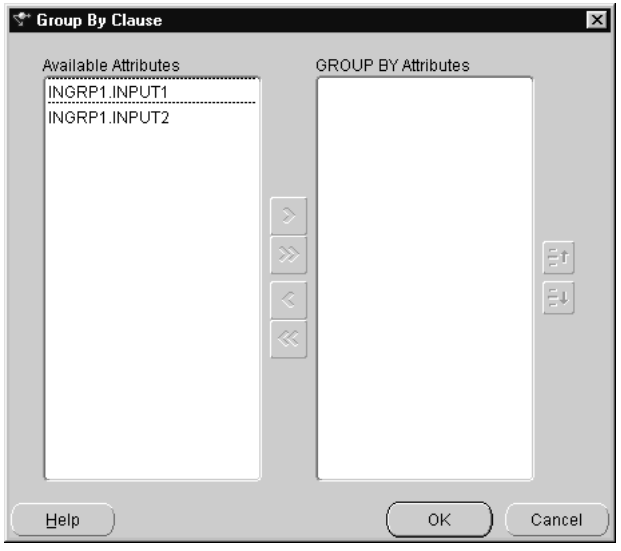
Figure 7–17 Aggregator Properties



7. Click the **...** button to the right of the Group By Clause property.

The Group By Clause dialog displays.

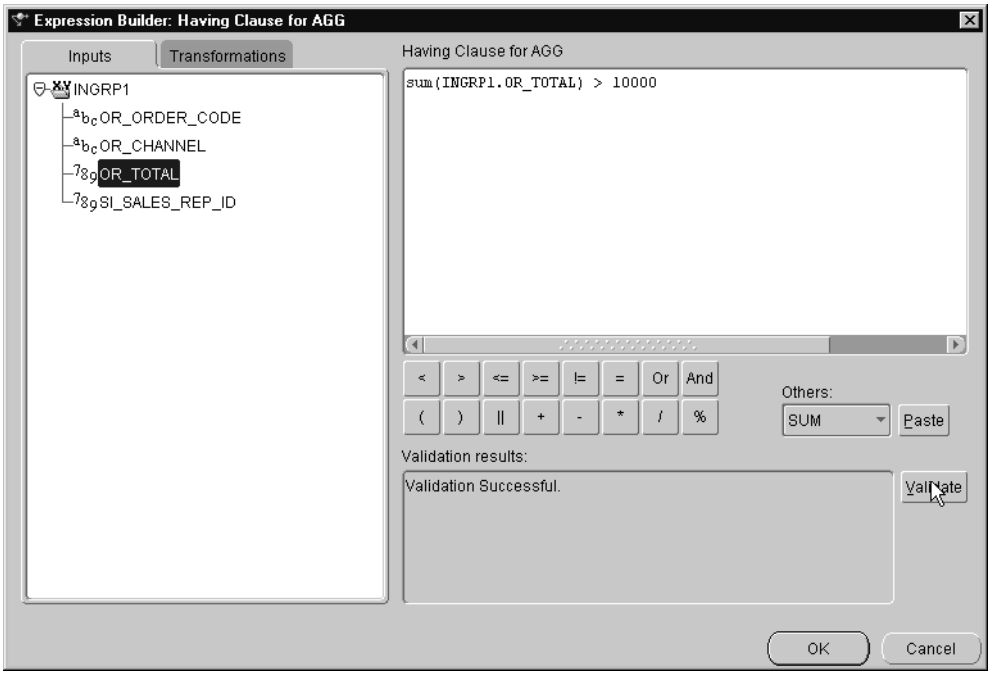
Figure 7–18 Group By Clause Dialog



8. Move the attributes to aggregate from the Available Attributes list to the GROUP BY Attributes list.
9. Click **OK.**
10. Click the **...** button to define a HAVING clause using Expression Builder.

    Create an expression, for example, `sum(INGRP1.OR_TOTAL) > 10000`.

Figure 7–19 Expression Builder Showing a Sum Statement



11. Map the attributes you edited from the attribute group OUTGRP1 of the aggregator operator to the attributes in the target.
12. From the **Mapping** menu, select **Generate,** and then **Mapping.**

    The Code Viewer displays generated code.

Filtering Data

You can conditionally filter out rows from a row set 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 mapping operator.

A Filter operator has only one input/output attribute 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:

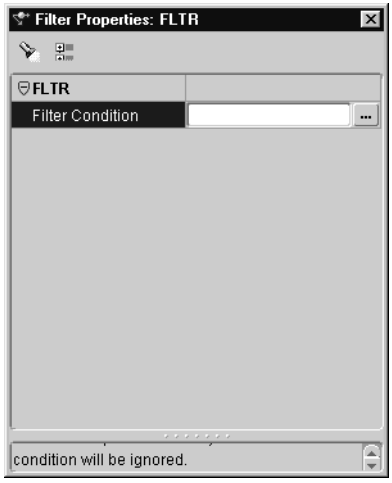
- **Filter Condition (operator level):** The boolean condition that determines which rows are sent to the output row set.

To filter data:

1. Drop the **Filter** operator onto the Mapping Editor canvas.
2. Connect source operator attributes to the Filter operator INOUTGRP1 attribute group.
3. Right-click the Filter operator header and select **Operator Properties** from the pop-up menu.

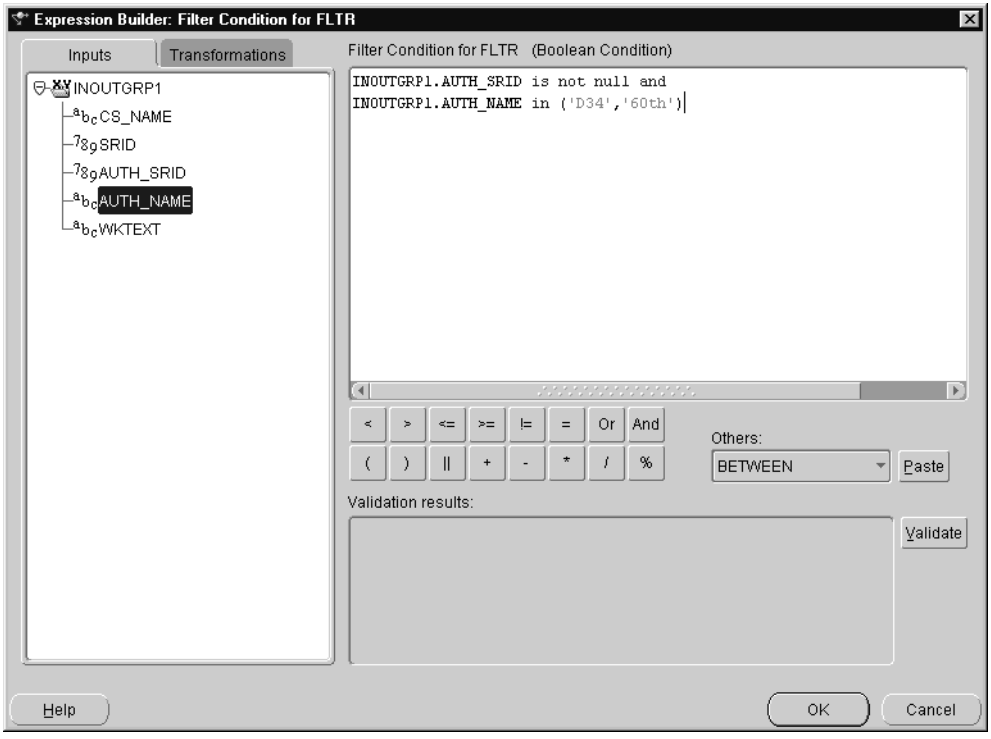
   The Filter Properties inspector displays.

Figure 7–20 Filter Properties Inspector



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 Expression Builder.

Figure 7–21 Expression Builder Showing a Filter Condition



5. Click **OK** in Expression Builder and close the Filter Properties inspector.
6. Connect the Filter operator outputs to the target INOUT attribute group.
7. Connect source operator attributes to the Filter operator and then to target operator attributes.
8. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code. The filter condition expressions generated 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.

Ordering Data

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.

The Sorter operator has one input/output attribute group. This group can be connected from any output or input/output attribute group and to any target input or input/output attribute group. Warehouse Builder sorts data by placing an ORDER BY clause in the code generated by the mapping.

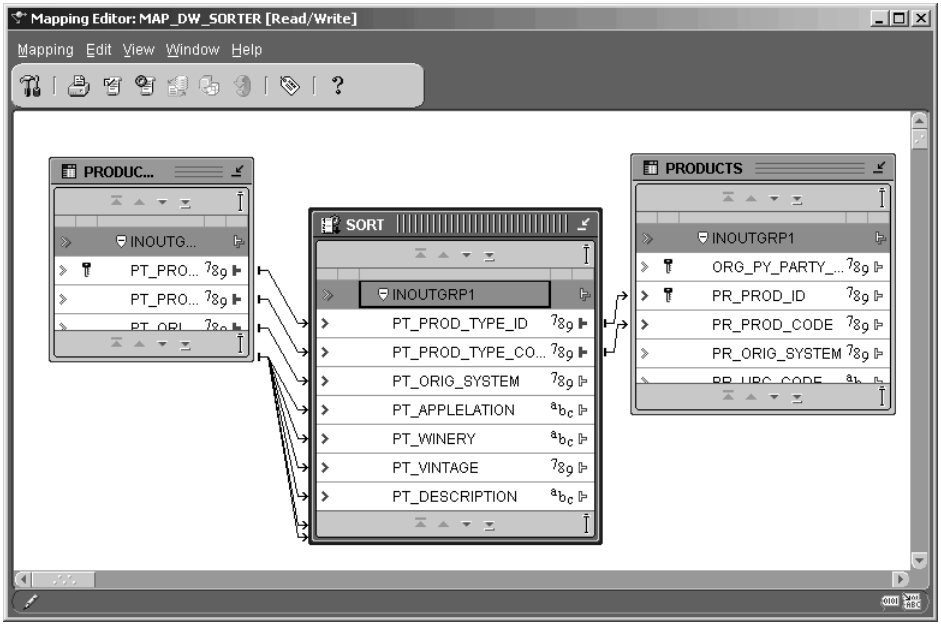
The Sorter operator contains the following property:

- **Order By Expression (operator level):** An ordered list of attributes in the INOUTGRP attribute group specifying that sorting is performed in the same order as the ordered attribute list. You can set ascending or descending sorting for each attribute.

To order data:

1. Drop the **Sorter** operator onto the Mapping Editor canvas.
2. Connect a source operator attribute group to the Sorter operator INOUTGRP1 attribute group.

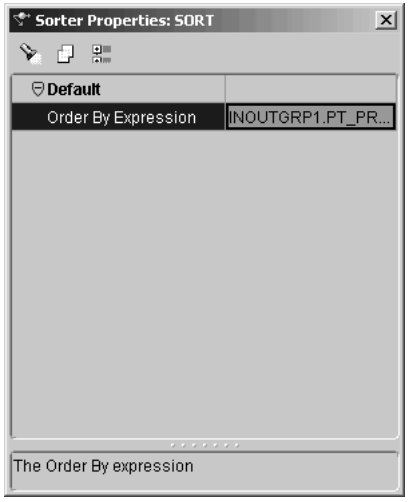
Figure 7–22 Mapping Editor with a Sorter Operator



3. Right-click the Sorter operator header and select **Operator Properties** from the pop-up menu.

   The Sorter Properties inspector displays.

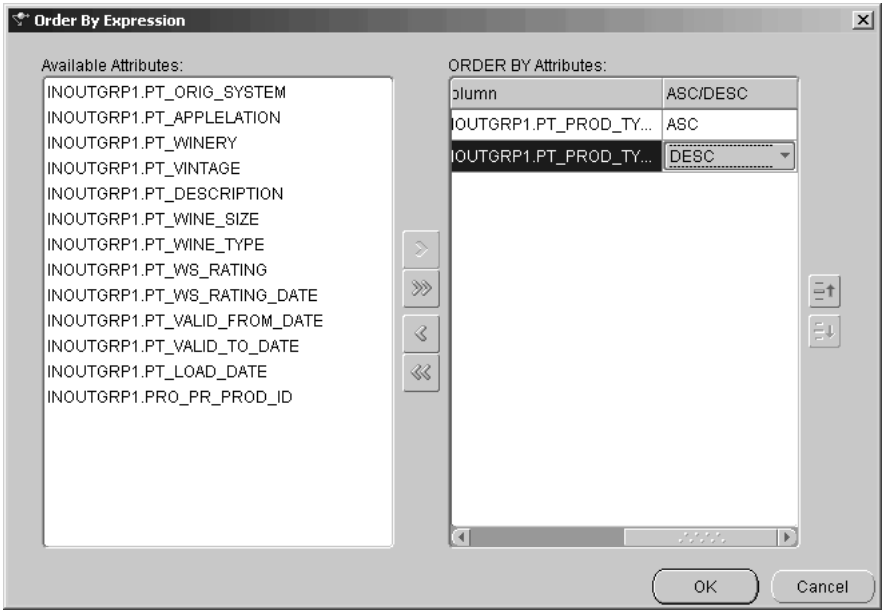
Figure 7–23 Sorter Properties Inspector



4. Click the **...** button in the Order By Expression field.

The Order By Expression dialog displays.

Figure 7–24 Order By Expression Dialog



5. Select the attributes you want to sort:
   - Select an attribute from the Available Attributes list and click the right arrow button.
   - Click the double right arrow button to select all of the Available Attributes.
6. Apply an ORDER BY clause to the attribute:
   a. Select the attribute in the ORDER BY Attributes list.
   b. Select **ASC** (ascending) or **DESC** (descending) from the drop-down list.
7. Click **OK.**
8. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code.

Cleansing Name and Address Data

The Name and Address operator enables you to cleanse data from multiple sources in differing formats. Name cleansing includes both personal name and business

name data. You can then map the clean data to a target or to another data flow operator.

By matching incoming data with data in the Warehouse Builder Name and Address data libraries, the Name and Address operator improves the accuracy and value of your name and address data. This operator enhances the quality of your data by:

- Separating name and address input data into individual elements.
- Correcting or validating address information such as street names, city names, and postal codes.
- Reformatting name and address data coming from differing formats.
- Augmenting names and addresses with additional data such as gender, ZIP+4, country code, apartment identification, and business and consumer identification.

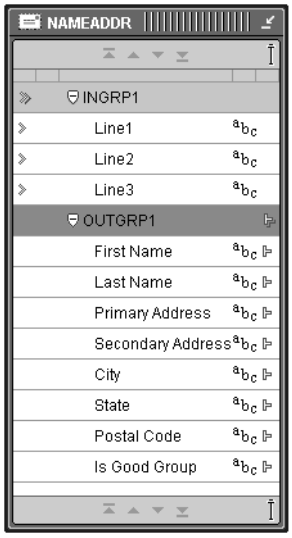
> **Note:** You must install the Warehouse Builder Name and Address data libraries to use the Name and Address operator. This option is available with the Oracle9*i* Database. Refer to the *Oracle9i Warehouse Builder Name and Address Installation Guide* for more information.

The Name and Address operator parses the data from a source operator into data that can then be reassembled appropriately for your business case. Name and address data is parsed according to these methods:

- **Name Only:** Sorts name-related data. Select this when the group contains only name data. A name can include both personal and business names.
- **Address Only:** Sorts address-related data. Select this when the group contains only address data and no name data.
- **Name/Address:** Sorts both name and address related data (default). Select this when the group contains both name and address data. This option is recommended when you do not know where the name ends and the address begins in your input data.

After you choose a parsing method, you define Input Roles and Output Components from lists of predefined role types that vary depending on the selected parsing option. Warehouse Builder adjusts the list of available Input Roles and Output Components based on the selected parsing method. Figure 7–25 shows how the Name and Address operator appears on the Mapping Editor canvas.

Figure 7–25 Name And Address Operator



The Name and Address operator contains one input group and one output group, both of which are created automatically when you drop the operator on the mapping editor canvas. You configure the attributes in the input group as Input Roles. An input role represents the input data mapped from a mapping source operator. You configure the attributes in the output group as Output Components.

Table 7–2 lists the input roles.

Table 7–2 Input Roles

Input Role	Description
None	This setting causes generation to fail.
First Name	First name. This can be a nickname or shortened version of the first name.
Middle Name	Middle name or initial.
Last Name	Last name (surname).
First Part Name	Person, first part, pre-name, first, middle.
Last Part Name	Person, last part, last name and title.
Person	Full person name, first part and last part.
Firm Name	Name of the company or organization, including divisions.

Table 7–2 Input Roles (Cont.)

Input Role	Description
Address	Full address line, primary address and secondary address.
Last Line	Last address line, city, state, and postal code. Use when city, state or province, and postal code are in one column.
City	Name of city.
State	Name of state or province.
Postal Code	Postal code, such as a ZIP code in the United States or a Postal Code in Canada.
Country Name	Full country name.
Country Code	The ISO 3166-1993 (E) two-character country code, for example, US for United States or CA for Canada.
Line1 through Line5	Intended for use as free-form name, business, personal, and address text. These selections can be used for any type of address system.

Table 7–3 lists the output components.

Table 7–3 Output Components

Output Component	Description
None	This setting causes generation to fail.
Person	First name, middle name, and last name.
Pre Name	Title or salutation appearing before a name.
First Name	The first name found in the input name.
First Name Standardized	Standard version of first name, for example, Theodore for Ted or James for Jim.
Middle Name	Middle name or initial.
Last Name	Last name (surname).
Other Post Name	Name suffix indicating certification, academic degree, or affiliation (Ph.D, M.D., R.N.).
Firm Name	Name of the company or organization, including divisions.
Gender	Probable gender (M=Male, F=Female, N=Neutral–gender is not defined).

Table 7–3 Output Components (Cont.)

Output Component	Description
Person Count	Number of personal names found in the Name Only or Name and Address groups.
Address	full address line, both primary and secondary.
Primary Address	Street name, house number, city map grid direction (SW, N), street type (avenue, street, road, etc.); does not include apartment or unit information.
Secondary Address	Unit designator and number, for example, Apt 3G.
Street Name	Name of street.
City	Name of city; US city names may be converted to United States Postal Service preferred names.
State	Name of state or province; this may be a county name for countries such as the United Kingdom.
Postal Code	Full postal code with spaces and other non-alphanumeric characters removed.
Postal Code Formatted	Formatted version of postal code that includes spaces and other non-alphanumeric characters, such as dashes.
Last Line	Neighborhood, city, state (or province), formatted postal code if address was fully assigned. If state contains a county instead of a state or province, it is not included.
Country Name	The full country name.
Latitude	Latitude in degrees north of the equator; positive for north of the equator, negative for south (always positive for North America).
Longitude	Longitude in degrees east of the Greenwich Meridian; positive for east of GM, negative for west (always negative for North America).
Address Type	Type of Address. T=Firm, G=General Delivery, H=High-rise apartment or office building, HD= High-rise default (the Name and Address operator can detect a finer level of postal code assignment if further input information were available), B=Box; R=Rural Code; S=Street.
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 may mean that the address is not a legal address, but may also indicate that postal matching is not available for the country.

Table 7–3 Output Components (Cont.)

Output Component	Description
Is Parsed	T=The name or address was parsed successfully. A name or address is considered to be successfully parsed even if some parsing errors exist. F=The name or address may not be parsed.
Is Good Group	Indicates whether the name group, address group, or name and address group was processed successfully. For name groups, this means the name has been successfully parsed; for address groups, the address has been found in a postal matching database if one is available, or has been successfully parsed if no postal database is installed. For name and address groups, both the name and the address have been successfully processed.
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.
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 errors exist. F=The name may not be parsed.
Is Address Verifiable	Indicates whether postal matching is available for the address. T= Postal matching is available for the address. This indicates that a postal matching database is installed for the country indicated by the address; matching is not available for the address. No postal matching database is installed for the country indicated by the address.

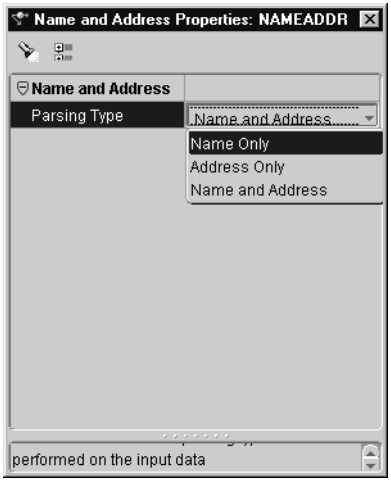
After parsing and matching the data, you are ready to map from the output parameter group to a target operator.

To cleanse name and address data:

1. Drop the **Name and Address** operator onto the Mapping Editor canvas.
2. Connect the source operator attributes to the Name and Address operator attribute group labeled INGRP1.
3. Right-click the Name and Address operator header and select **Operator Properties** from the pop-up menu.

The Name and Address Properties inspector displays.

Figure 7–26 Name and Address Operator Property Inspector

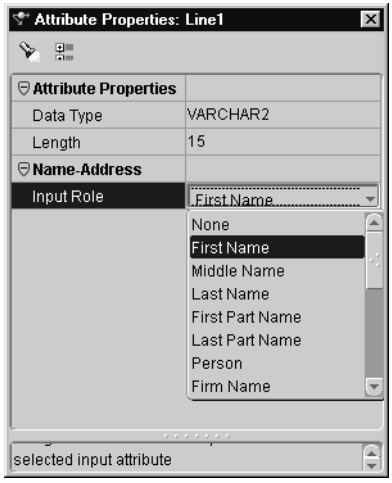


4. Click the field to the right of the **Parsing Type** property and select a parsing type from the drop-down list.
5. Close the Name and Address Properties inspector.
6. Connect the source operator attributes to the Name and Address attributes.
7. Define the Input Role for each input attribute.
   a. Right-click an input attribute and select **Attribute Properties** from the pop-up menu.

      The Attribute Properties inspector displays.

   b. Select an item from the drop-down list to the right of the Input Role field.

Figure 7–27 Attribute Properties Inspector Showing the Input Role



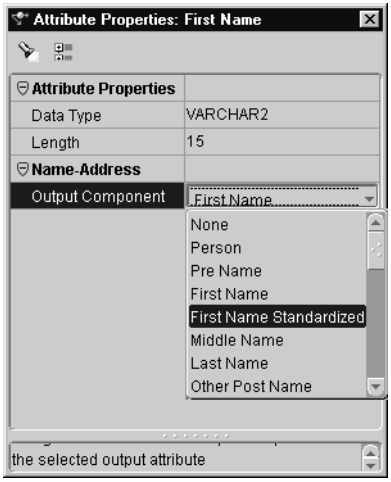
8. Define the Output Component for each output attribute.

a. Right-click an output attribute and select **Attribute Properties** from the pop-up menu.

The Attribute Properties inspector displays.

b. Select an item from the drop-down list located to the right of the **Output Component** field.

Figure 7–28 Attribute Properties Inspector Showing the Output Component



9. Connect the Name and Address operator outputs to a target attribute group.
10. From the **Mapping** menu, select **Generate,** and then **Mapping.**

    The Code Viewer displays generated code.

Adding a Pre-Mapping Process

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. You select a function or procedure from the tree, and the operator displays with predefined input and output parameters.

The Pre-Mapping Process operator has attribute groups corresponding to the number and direction of the parameters associated with the selected PL/SQL procedure or function. You can modify this list by using reconciliation.

There can be only one Pre-Mapping Process operator for a mapping. Only constants, mapping input parameters, and output from a Pre-Mapping Process can be mapped into a Post-Mapping Process operator.

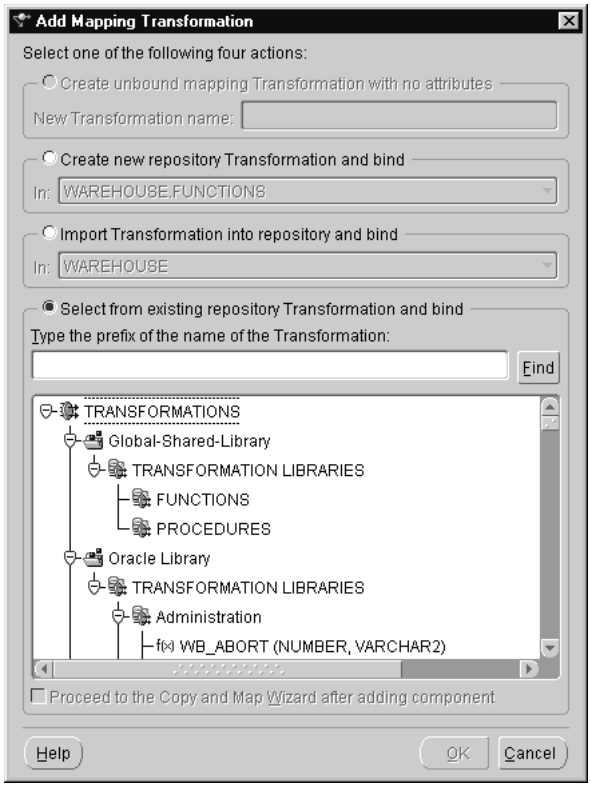
Note: Constants, Sequences, Data Generators, and Mapping Input Parameters can precede Pre-Mapping Processes.

To add a pre-mapping process operator to a mapping:

1. Drop a **Pre-Mapping Process** operator onto the Mapping Editor canvas.

   The Add Mapping Transformation dialog displays.

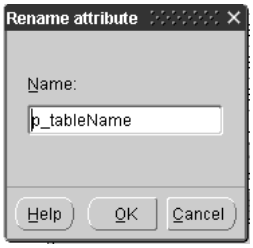
Figure 7–29 Add Mapping Transformation Dialog



2. Select a procedure from the selection list.
3. Connect the output attribute of the Pre-Mapping Process operator to the input attribute group of a target operator.

4. Rename the operator:
   a. Right-click the operator header.
   b. Select **Rename** from the pop-up menu.
   c. Enter a new name.
   d. Click **OK.**

Figure 7–30 Rename Attribute Dialog



5. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code.

Adding a Post-Mapping Process

The Post-Mapping Process operator calls a function or procedure whose metadata is defined in Warehouse Builder after the map is executed. The output parameter group provides the connection point for the returned value (if implemented via 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 has attribute 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.

There can be only one Post-Mapping Process operator for a mapping. Only constants, mapping input parameters, and output from a Pre-Mapping Process can be mapped into a Post-Mapping Process operator. The Post-Mapping Process operator is not valid for an SQL*Loader mapping.

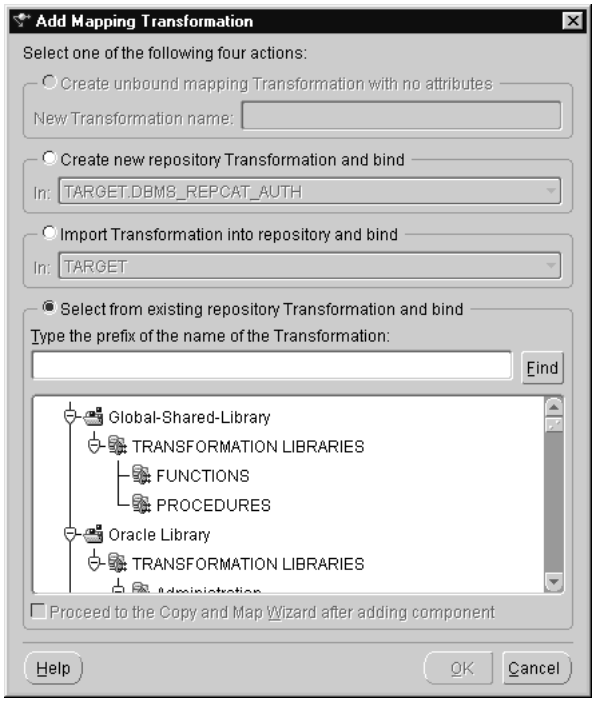
Note: Constants, Sequences, Data Generators, and Mapping Input Parameters can precede Post-Mapping Processes.

To add a post-mapping process operator to a mapping:

1. Drop a **Post-Mapping Process** operator onto the Mapping Editor canvas.

   The Add Mapping Transformation dialog displays.

Figure 7–31 Add Mapping Transformation Dialog



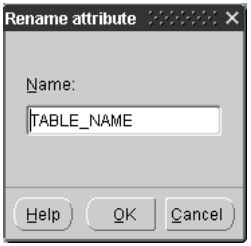
2. Select the appropriate procedure from the selection list.
3. Connect the output attribute of a source operator to the INOUTGRP1 of the Post-Mapping Process operator.
4. Rename the operator:
   a. Right-click the operator header.

b. Select **Rename** from the pop-up menu.

c. Enter a new name.

d. Click **OK.**

Figure 7–32 Rename Attribute Dialog



5. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code.

Adding Mapping Input Parameters

A Mapping Input Parameter operator enables you to pass parameter values into a mapping. The Mapping Input Parameter operator has a cardinality of one and 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 operator in a mapping.

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'...
```

If a mapping parameter output named param1 has the data type VARCHAR2, the generated main function in the PL/SQL package appears as:

```
... param1 IN VARCHAR2 ...
```

The Mapping Input Parameter operator contains the following properties:

- **Default Value (attribute level):** 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'.
- **Data Type (common attribute level):** Specifies the data type for this input parameter.

> **Note:** The Mapping Input Parameter operator can be mapped to the same target operators as a Constant operator.

To add a Mapping Input Parameter operator to a mapping:

1. Drop a **Mapping Input** operator onto the Mapping Editor canvas.
2. Add an attribute to the Mapping Input operator:
   a. Right-click the MAP_INPUT attribute group.
   b. Select **Add/Remove Attributes** from the pop-up menu.
   c. Add a name for the attribute in the Add/Remove Attributes dialog.
   d. Click **OK.**

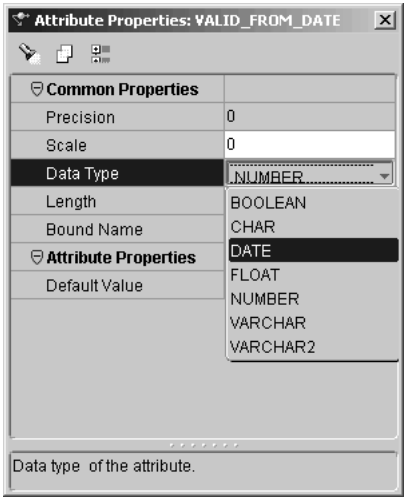
Figure 7–33 Add/Remove Attributes Dialog

3. Change the Data Type for the new attribute:
   a. Right-click the new property.

b. Select **Attribute Properties** from the pop-up menu.

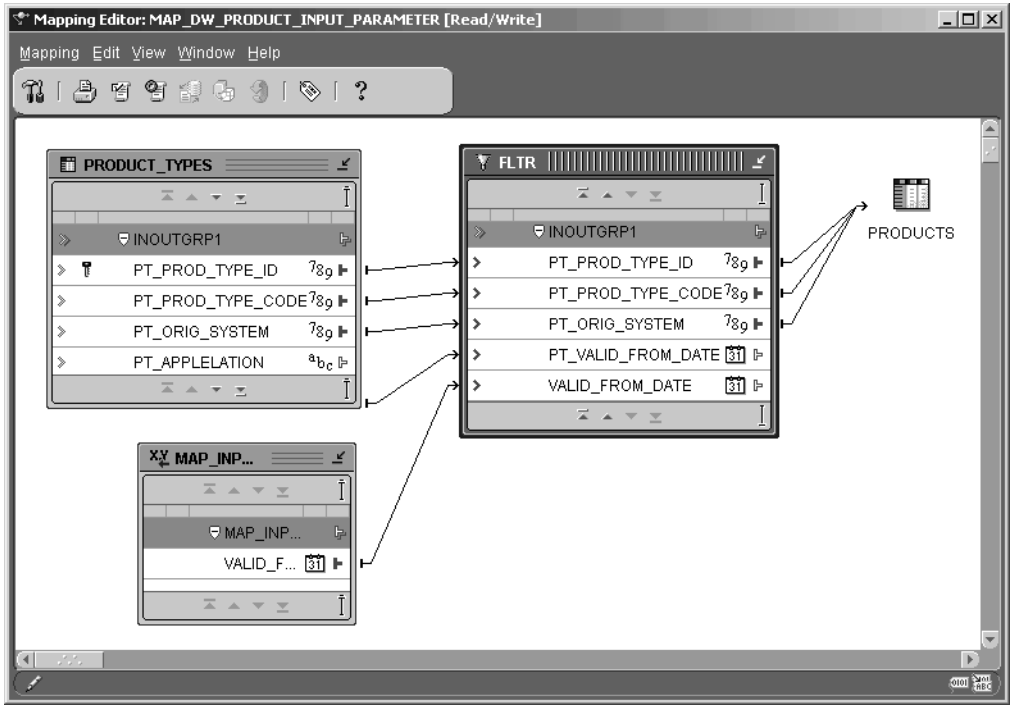
c. Select a new data type from the drop-down list.

Figure 7–34 Attribute Properties Inspector



4. Connect the Input Parameter operator MAP_INPUT attribute to an attribute group of the target operator.

Figure 7–35 Mapping Editor Showing A Mapping Input Parameter Operator



5. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code.

Adding Mapping Output Parameters

The Mapping Output Parameter operator enables you to send values out of a mapping. The Mapping Output Parameter operator must have only one input attribute group. You can have only one Mapping Output Parameter operator in a map. Attributes not associated with a row set can be mapped into a Mapping Output Parameter operator. For example, constant, input parameter, output from a pre-mapping process, or output from a post process can all contain attributes not associated with a row set. A Mapping Output Parameter operator 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 parameter output named `param1` has data type VARCHAR2, the generated main function in the PL/SQL package appears as:

```
... param1 OUT VARCHAR2 ...
```

The Mapping Output Parameter operator contains the following properties:

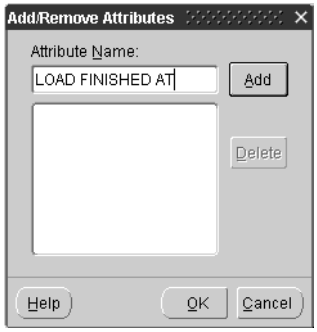
- **Data Type (common attribute level):** Specifies the data type for this output parameter.
- **Bound Name (common attribute level):** Specifies the actual physical name for this output parameter.

> **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 add a mapping output parameter to a mapping:

1. Drop a **Mapping Output Parameter** operator onto the Mapping Editor canvas.
2. Add an output attribute to the Mapping Output Parameter operator.
   a. Right-click the MAP_OUTPUT attribute group.
   b. Select **Add/Remove Attributes** from the pop-up menu.
   c. Add a name for the attribute in the Add/Remove Attributes dialog.
   d. Click **OK.**

Figure 7–36 Add/Remove Attributes Dialog



3. Change the data type of the new attribute:

a. Right-click the new property.

b. Select **Attribute Properties** from the pop-up menu.

c. Select a new data type from the drop-down list.

d. Close the Attribute Properties inspector.

Figure 7–37 Attribute Properties Inspector

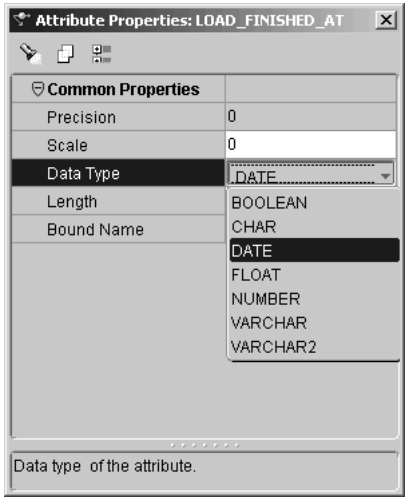
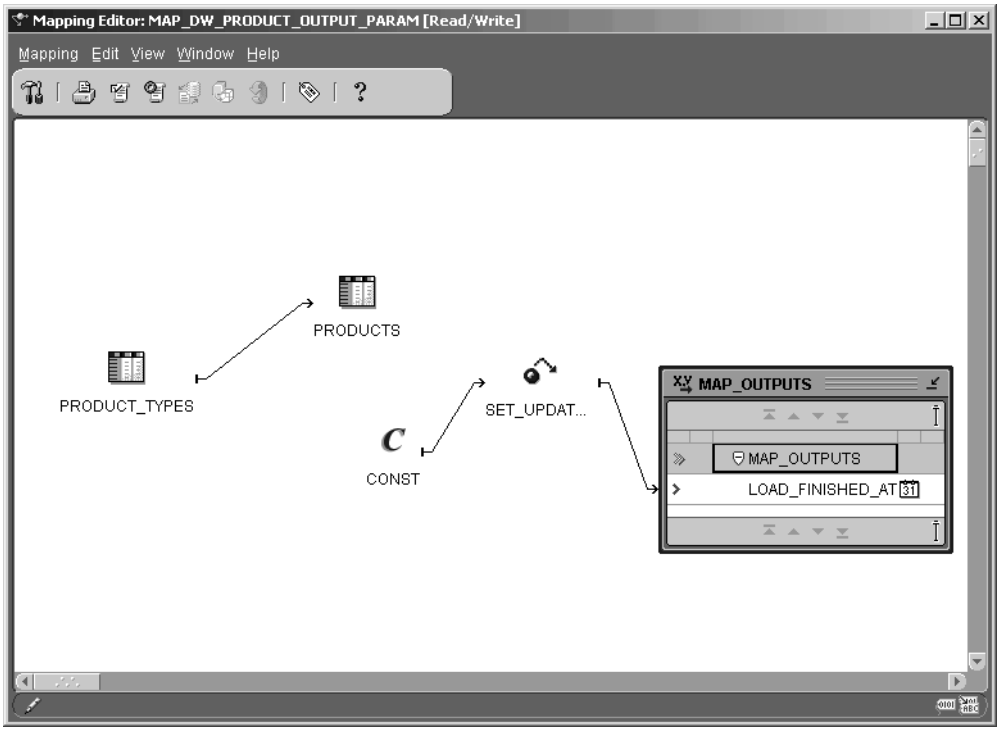


Figure 7–38 Mapping Editor Showing An Output Parameter Operator



4. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code.

Adding External Processes

The External Process operator enables you to represent a process not defined by Warehouse Builder to incorporate it into a mapping. The External Process operator is a self-contained operation and requires no inputs or outputs. During code generation, Warehouse Builder generates a job control code (TCL script) for external process operators that can be deployed as part of the Warehouse Builder workflow.

The following types of processes are available in Warehouse Builder:

- OS Executable
- Pure Integrate

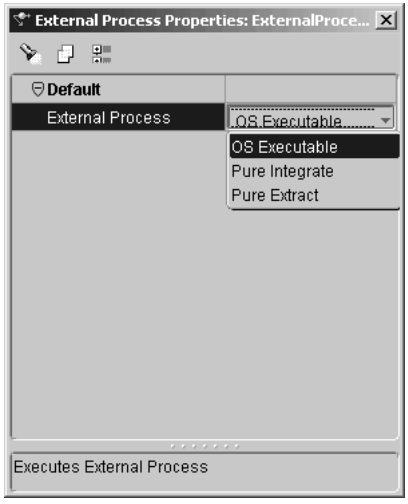
- Pure Extract

To add an External Process operator to a mapping:

1. Drop an **External Process** operator onto the Mapping Editor canvas.
2. Right-click the External Process operator header and select **Operator Properties** from the pop-up menu.

   The External Process Properties inspector displays.

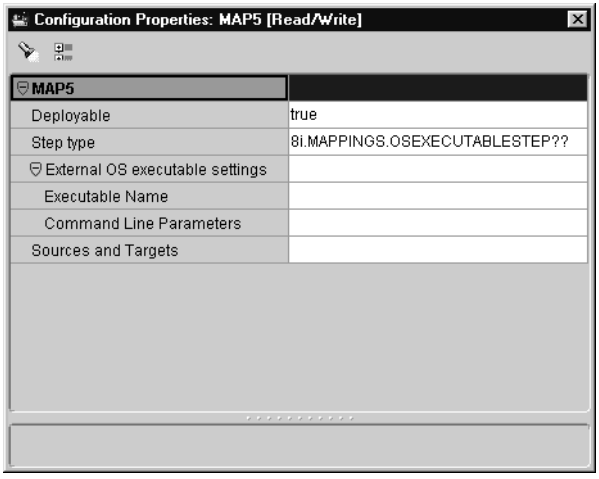
Figure 7–39 External Process Properties Inspector



3. Click the **External Process** field and select one from the drop-down list.
4. From the Warehouse Module Editor, right-click the map containing the External Process operator and select **Configure** from the pop-up menu.

   The Configuration Properties inspector displays.

Figure 7–40 External Process Configuration Properties Inspector



5. For OS Executable and Pure Integrate processes, enter:
   - **Executable Name:** The directory path and name of the executable.
   - **Command Line Parameters:** The string of parameters to be passed to the executable.
6. For Pure Extract processes, enter:
   - **FTP Directory:** Directory for performing FTP.
   - **Status File name:** The file containing the execution status of the Pure Extract process.
7. Close the Configuration Properties inspector.
8. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code.

Adding Transformations

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. Also, 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 repository.

Warehouse Builder provides pre-defined PL/SQL library functions in the runtime schema that can be selected as a bound function when dropping 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 (operator level, read-only):** The text template for the function call that is generated by the code generator for the, with the operator attribute names listed as the calling parameters. For the actual call, the operator attribute names are replaced with the actual source or target columns that are connected to the attributes.
- **Function Name (operator level, read-only):** The name of the function or procedure, to which this operator is bound.
- **Procedure (operator level, read-only):** A boolean value indicating, if true, that the bound transformation is a procedure rather than a function with no returned value.
- **Data Type (attributes, read-only):** Indicates the data type of the input, output, or result parameter of the bound function that corresponds to the given attribute.
- **Default Value (input attributes, read-only):** The default value (blank if none) for the given attribute.
- **Optional Input (input attributes, read-only):** 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 (output attributes, read-only):** 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 add a mapping transformation operator:

1. Drop a **Mapping Transformation** operator onto the Mapping Editor canvas.

The **Add Mapping Transformation** dialog displays.

2. Select one of the following options:

   - Create a new transformation repository and bind.
   - Import existing transformation into repository and bind.
   - Select from existing repository transformation and bind.

     This selection contains a search text box and a directory tree for all the transformations stored in the Warehouse Builder repository. Click the appropriate node to locate a transformation. Double-click a transformation to select it.

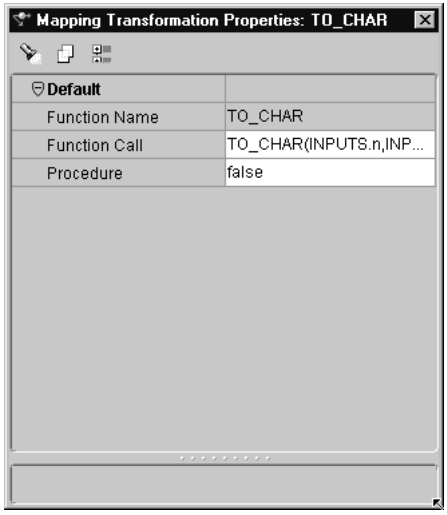
   For more information on these options, see "Selecting Data Operators" beginning on page 6-10.

   Note: To select multiple items, hold down the Control key as you click each item you want to select. To select a group of items located in a series, click the first object in your selection range, hold down the Shift key, and then click the last object.

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** from the pop-up menu.

   The Mapping Transformation Properties inspector displays.

Figure 7–41 Mapping Transformation Properties Inspector



a. Select an input attribute.

b. If the Procedure property is set to True**,** then do not connect the input parameter.

c. Close the attribute property inspector.

5. Connect the Transformation operator output attributes to the target attributes.

6. From the **Mapping** menu, select **Generate,** and then **Mapping.**

The Code Viewer displays generated code.

Adding Expressions

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 Property inspector for the output attribute and open Expression Builder.

You can have only one input attribute group and one output attribute group in the Expression operator, both of which 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.

The Expression operator contains the following properties:

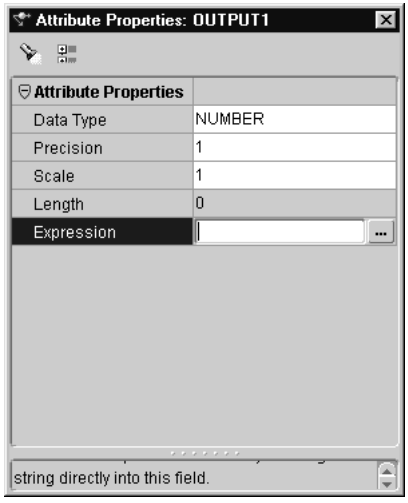
- **Data Type (attributes):** The data type of the attribute.
- **Precision (attributes):** The precision of the attribute, used for numeric type attributes only.
- **Scale (attributes):** The scale of the attribute, used for numeric type attributes only.
- **Length (attributes):** The length of the attributes, used for string type attributes only.
- **Expression (output attributes):** The text expression template for the output attribute. For code generation, the input attributes are replaced by the input attribute names in the expression template.

To add an expression operator:

1. Drop an **Expression** operator onto the Mapping Editor canvas.
2. Connect the appropriate source attributes to the INGRP of the Expression operator.

   This automatically creates the input attributes.
3. Create an output attribute in the Expression operator output attribute group:
   a. Right-click OUTGRP.
   b. Select **Add/Remove Attributes** from the pop-up menu.
   c. Add an attribute.
   d. Click **OK.**
4. Right-click the output attribute and select the Attribute Properties from the pop-up menu.

Figure 7–42 Attribute Properties Inspector



5. Click the field to the right of the **Expression** property and enter a filter condition expression or click **...** to open Expression Builder and define an expression.
6. Close the attribute properties inspector.
7. Connect the Expression output attribute to the appropriate target attribute.
8. From the **Mapping** menu, select **Generate,** and then **Mapping.**

   The Code Viewer displays generated code.

The next chapter, Chapter 8, "Configuring and Generating Mappings" shows you how to configure your mapping for generating deployment code.

8

Configuring and Generating Mappings

This chapter describes how to configure operator properties of a mapping, and how to manage code generation. It also describes how to reconcile mapping operators with repository objects, mapping validation, and techniques for improving deployment performance.

This chapter includes the following topics:

- Reconciling Mapping Operators with Repository Objects
- Configuring a Mapping
- Validating and Generating a Mapping

Reconciling Mapping Operators with Repository Objects

Once you have selected the source and target objects, configured their properties, and linked the operators, you may need to reconcile the mapping operators with their corresponding repository objects.

Coordinating mapping operators with the status of repository objects is called reconciliation. You can reconcile operators with repository objects (inbound) or you can reconcile repository objects with operators (outbound).

Do not confuse reconciliation with synchronization. While synchronization ensures that you are up-to-date with changes made by other users in a multi-user environment, reconciliation updates mapping operators with changes made to the physical repository objects.

Using Inbound Reconciliation

Inbound reconciliation updates the target mapping operator with its repository object. You can also use Inbound Reconcile to bind a mapping operator to a repository object. You can Inbound 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:

- To capture structural changes previously applied to the selected repository object and propagate these to the mapping operator.
- To capture attribute name changes previously applied to the selected repository object and propagate these to the mapping operator.
- To capture attribute data type changes previously applied to the selected repository object and propagate these to the mapping operator.
- To associate the mapping operator with a different repository object of the same type. 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
- To associate the mapping operator with a repository object of a different type. For example, if you want to provide access to the underlying source data through views rather than directly against the base tables.
- To prototype your mappings using tables. When you are satisfied with the transformation logic, you can switch to other object types for the production mappings, for example, views, materialized views, or facts.

Inbound reconciliation has no impact on the linking of other mapping operators in the mapping to repository objects; Warehouse Builder preserves these links. Table 8–1 describes the operator types supported by reconciliation.

Table 8–1 Operator Objects Reconciled with Repository Objects

Operator Object	Repository Object
Mapping Tables	Table, Views, Materialized Views, Sequences, Files, Dimensions and Facts.
Mapping Views	
Mapping Materialized Views	
Mapping Sequences	
Mapping Flat Files	
Mapping Dimensions	
Mapping Facts	
Mapping Transformations	Transformations only.

To reconcile inbound mapping operators:

1. Select a mapping operator on the Mapping Editor canvas.
2. From the **Edit** menu, select **Reconcile Inbound** or right-click the header of the mapping operator and select **Reconcile Inbound** from the pop-up menu.

   The Inbound Reconcile Operator dialog displays.

Figure 8–1 Inbound Reconcile Operator Dialog

3. Set the Reconciliation Strategies.

   - **Matching by Object Identifier:** This strategy uses the unique object identifiers to determine the correlation between the mapping operator attributes and those of the selected repository object. Attributes of the mapping operator that cannot be matched with those of the repository object are removed. This can happen if an attribute was added to the mapping 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 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 mapping operator are added as new attributes at the end of the mapping operator. Mapping lines for matched attributes are preserved. Use this strategy if you want to keep your mapping operators in step with changes to the bound repository object and if you want to maintain separate logical names for your mapping operator attributes despite changes to physical names in the repository object. Match by object identifier is not available if you want to reconcile to a different repository object.

- **Matching by Physical (Bound) Name:** This strategy uses matching between the bound names of the mapping operator attributes and the physical names of the repository object attributes. Matching is case-sensitive. On inbound reconciliation, attributes of the mapping 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 mapping operator are added as new attributes to the mapping operator. Mapping lines for matched attributes are preserved. Because bound names are read-only after you have bound a mapping operator to a repository object, it is not possible to manipulate them to achieve a different match result in inbound reconciliation. You use this strategy if you want to maintain equivalence of physical names and logical names in your mapping operators. 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 mapping operator.
- **Matching by Position**: This strategy matches mapping operator attributes with columns, fields, or parameters of the selected repository object by position. For example, the first attribute of the mapping operator is reconciled with the first attribute of the repository object, the second with the second, and so on. If the mapping operator has more attributes than the repository object, then the excess attributes are removed from the mapping 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 mapping operator, then they are added as new attributes to the end of the mapping operator. Mapping lines for existing attributes in the mapping operator are preserved. Use this strategy for reconciliation with a different repository object if you want to preserve the logical names of your mapping operator attributes. This strategy is most effective when the only changes to the repository object are the addition of extra columns, fields, or parameters at the end of the object.

If you do not select a strategy, then the reconcile process replaces the attributes of the operator with an exact copy of the columns or fields of the selected repository object.

If you select more than one strategy, then matching occurs in the following order:

- Match by object identifier
- Match by position
- Match by name

4. Click **OK.**

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.

Using Outbound Reconciliation

Outbound reconciliation updates a selected repository object to reflect changes in the mapping operator. You can use outbound reconciliation for any of the following reasons:

- To capture structural changes applied to the mapping operator and propagate these to the currently linked repository object.
- To capture attribute logical name changes applied to the mapping operator and propagate these to the currently linked repository object.
- To capture attribute data type changes applied to the mapping operator and propagate these to the currently linked repository object.
- To create a new repository object in the same warehouse module or to create or replace a different repository object in a different module. You can do this if you are defining staging tables with your mappings.
- To copy and map the attributes of one operator to a second operator and propagate these changes to the repository object associated with the second operator.

You cannot create a repository object of a different type using outbound reconciliation. Outbound reconciliation has no impact on the linking of other mapping operators in the mapping to repository objects.

Table 8–2 lists the eligible Mapping Operators for Outbound Reconcile:

Table 8–2 Outbound Reconcile Operators

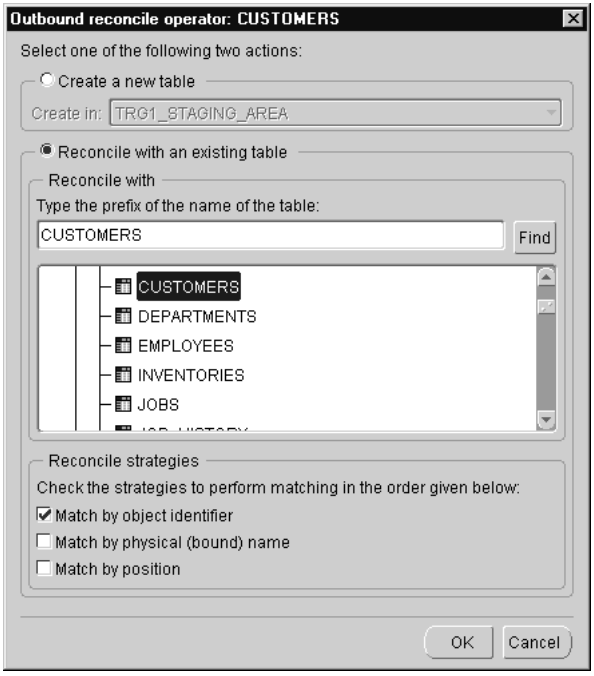
Operator Type	Notes
Table	Creates or updates a table object in the repository. Attributes and data type information are copied as columns of the table. Constraint properties are not copied.
View	Creates or updates a view object in the repository. Attributes and data type information are copied as columns of the view.
Materialized View	Creates or updates a materialized view object in the repository. Attributes and data type information are copied as columns of the materialized view. Constraint properties are not copied.
Transformation	Creates or updates a function object in the repository. Input attributes and data type information are copied as input parameters of the function. Output attribute and data type information is copied as return specification for the function.

To reconcile outbound mapping operators:

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** from the pop-up menu.

   The Outbound Reconcile dialog displays.

Figure 8–2 Outbound Reconcile Dialog



3. Set the Reconcile Strategies.
4. Click **OK.**

You can use the outbound reconcile feature to create new objects or update existing objects in the repository that are derived from a mapping operator. After you have created a mapping operator, you can copy and map its attributes into a new operator. You can then create a new corresponding repository object that inherits the same properties.

To create a new table in the repository associated with a Mapping Table:

1. Select a Mapping Table.
2. From the **Edit** menu, select **Reconcile Outbound.**

   The Reconcile Outbound dialog displays.
3. Select **Create a new table** and its location in the repository.
4. Click **OK.**

A table with identical properties and attributes as the Mapping Table is created in the repository.

Configuring a Mapping

You use the properties inspectors to configure how the operators behave in an ETL process. The properties that you configure vary according to the processes you defined when you created the mapping. You can configure the following sets of properties using properties inspectors:

- Mapping operator properties
- Mapping operator attribute properties
- Mapping physical properties

The following sections describe the settings on the Operator Properties inspectors for mapping relational table, flat file, and materialized view operators. See Chapter 7, "Using Mapping Operators and Transformations" for information on settings unique to data flow operators.

Configuring Mapping Table Operators

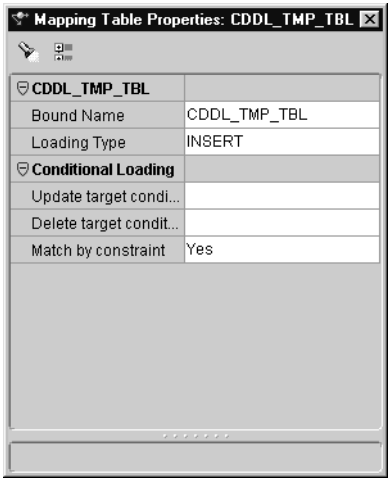
This section describes how to configure a Mapping Table operator using the Operator Properties inspector. The Operator Properties inspector described in this section contains the same settings for Mapping Facts, Mapping Dimensions, Mapping Views, and Materialized Views.

You configure settings in the following categories:

- Mapping Operator Name
- Conditional Loading

Mapping Dimension and Mapping Fact operators contain a read-only category for keys.

Figure 8–3 Mapping Table Properties Inspector



To configure a Mapping Table operator:

1. Select the operator you want to configure by clicking its header.
2. From the **Edit** menu, select **Properties** or right-click the header of the operator and select **Operator Properties.**

   The Mapping Table Properties inspector displays.
3. Click the operator name node to expand the properties list.
   a. Check the Bound Name.
   b. Select a **Loading Type** from the drop-down list.
4. Click the **Conditional Loading** node.
   a. Enter an **Update target condition** or click **...** to open Expression Builder and define a condition.

      If evaluated to true, the row is included in the update loading operation.
   b. Enter a **Delete target condition** or click **...** to open Expression Builder and define a condition.
   c. Click **Match by constraint** and select **Yes** or **No** from the drop-down list.
5. Check the Keys to ensure accuracy.
6. Close the property inspector.

Mapping Operator Name

These settings define your operator name if it is unbound, and the loading type.

- **Bound Name:** The name used by the code generator. If a mapping operator or attribute is currently bound and reconciled, then this property is read-only. If an object or attribute is not yet bound, you can edit the bound name within the mapping editor before you reconcile it to a repository object.
- **Loading Type:** Select a loading type for each target operator from the drop-down list:

  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. Update fails if no row exists for the specified match conditions.

  INSERT / UPDATE–For each incoming row, an insert operation is performed first. If the insert fails, an update operation occurs.

  UPDATE / INSERT–For each incoming row, an update operation is performed first.

  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.

  > **Note:** 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.

Note: If your warehouse module is set to Oracle9*i*, a MERGE statement is generated in the PL/SQL package for the INSERT/UPDATE and UPDATE/INSERT load types. MERGE enhances load performance by:

- Performing both inserts and updates in single DML statements
- Switching to a different mode in the kernel instead of PL/SQL exceptions
- Using row-based code

If a target attribute that is used for matching is also used for loading, a MERGE statement is not generated. If a sequence is mapped to a target, a MERGE statement does not generate and Validation does not issue a warning.

See the *Oracle9i SQL Reference* for more information on merge.

Conditional Loading

These settings define:

- **Update Target Condition:** If evaluated 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:** Indicates whether unique or primary key information on a target overrides the matching criteria obtained from its attributes.

Keys (read-only)

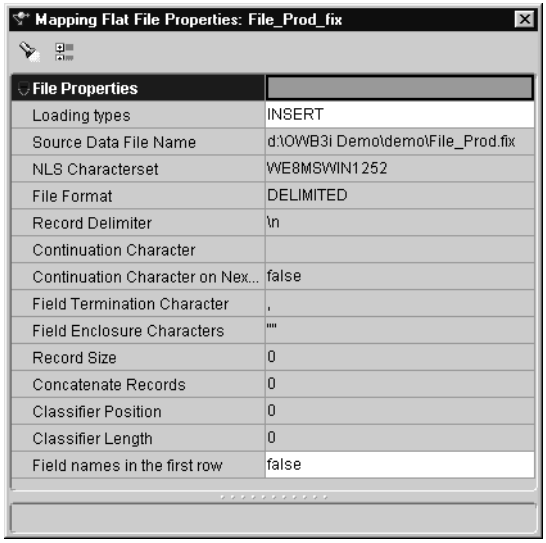
The following read-only settings are available in the mapping fact and mapping dimension operator properties.

- **Name:** Name of the primary, foreign, or unique key.
- **Key Columns:** Local columns that define this key. Each key is comma-separated if the operator contains more than one key.
- **Key Type:** Type of key, either primary, foreign, or unique.
- **Referenced Keys:** If the operator contains a foreign key, this is the key or keys used by the referenced object.

Configuring Flat File Operators

You can configure a Flat File operator as either a source or target. You can configure the Loading type and the first row naming for this operator. All other settings are read-only.

Figure 8–4 Operator Properties Inspector (Mapping Flat File)



- **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.

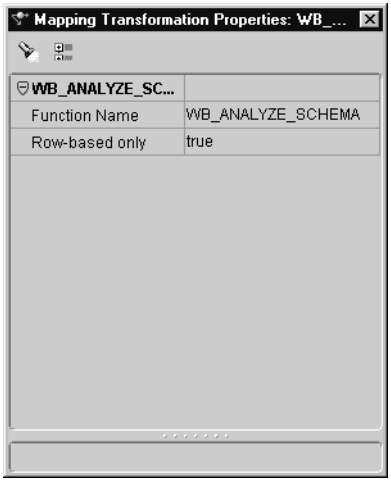
  NONE–No operation is performed on the data target. This setting is useful for test runs, where all transformations and extractions are run but have no effect on the target.

- **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.

Configuring Transformation Operators

The following settings are read-only. See "Adding Transformations" on page 4-27 for more information on transformations.

Figure 8–5 Operator Properties Inspector (Transformation)



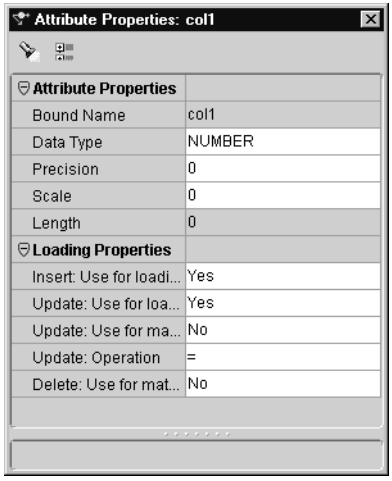
Function/Procedure Name: Name of the transformation.

Row-based Only: Indicates if the transformation is loaded using row-based mode. Some transformations can use set-based mode or row-based mode.

Configuring Mapping Operator Attributes

You set most mapping operator attributes when you define them during the module definition phase. However, if you want to alter operator attributes to execute your mapping, you can do this directly from the Attribute Properties inspectors.

Figure 8–6 Mapping Operator Attribute Property Inspector



Setting Attribute Properties

This category contains the data object attribute settings.

- **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.
- **Data Type:** Type of data that this attribute will hold.
- **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.

Setting Loading Properties

The mapping table, mapping dimension, mapping fact, mapping view, and mapping materialized view operators have a Loading Properties category. This category contains the following settings:

- **Insert: Use For Loading:** 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.

- **Update: Use For Loading:** 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 will reach 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 returns an error. If a column (not a key column) is not mapped, then it is not used in loading.
- **Update: Use For Matching:** 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:** When a matching row is located and an update operation is performed on the target, different computations can be done between the data of the source attribute and the target attribute on the matched row before the resulting value is stored onto the target. You can specify one of the following target conditions:

  = target:= source

  += target:= source + target

  -= target:= target - source

  =- target:= source - target

  ||= target:= target||source

  =|| target:= source||target

- **Delete: Use For Matching:** 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. The constraints can override this setting.

Configuring Physical Properties for a Mapping

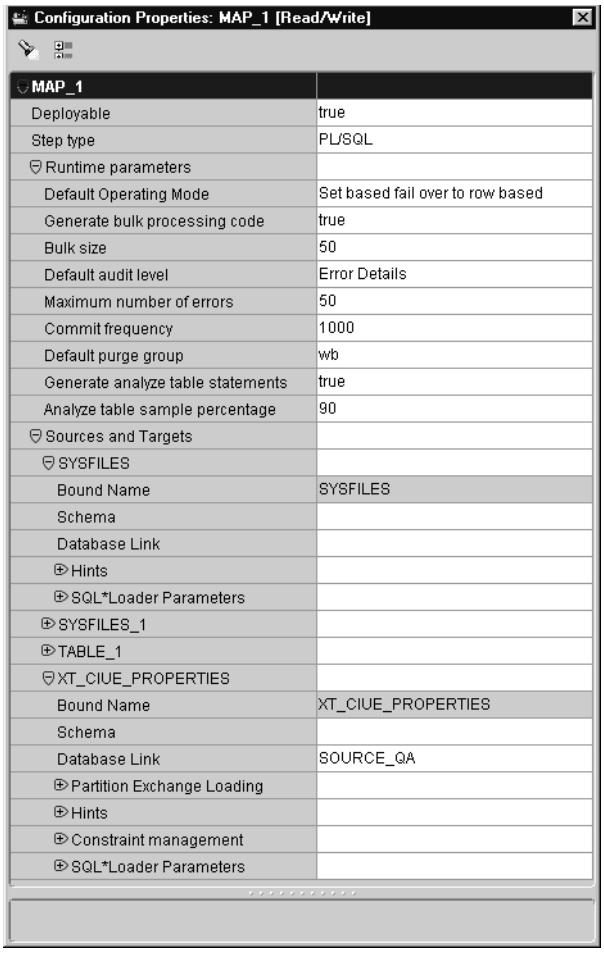
After you set the mapping operator properties, you can configure the physical properties of the mapping using the Configuration Properties dialog. The following sections describe the basic steps for configuring the physical properties of a mapping containing tables, flat files, and SAP files.

The Configuration Properties inspector displays the physical properties of the mapping grouped into different categories:

- Deployable
- Step Type
- Runtime Parameters
- Source and Target

The properties that are available for configuration vary according to the type of object. Use the properties inspectors to review and configure each object you are planning to deploy.

Figure 8–7 Configuration Properties Inspector For Mapping Tables



To configure mapping physical properties:

1. From the **Edit** menu, select **Properties...** or right-click the mapping you want to configure and select **Configure...** from the pop-up menu.

   The Configuration Properties inspector displays.

2. Expand the **Runtime Parameters** node to configure your mapping for deployment.

   See "Setting Runtime Parameters" below for information on these settings.

3. Expand the **Sources and Targets** node to set the physical properties of the operators in the mapping.

   See "Setting Sources and Targets" on page 8-21 for information on these settings.

4. Close the window.

Setting Deployable

This setting enables Warehouse Builder to generate a set of scripts to create a data warehouse object for those mapping entities marked as deployable.

- **True:** Marks the mapping entity as deployable (default).
- **False:** Marks the mapping entity as not deployable.

Setting the Step Type

A mapping step is the result of the implicit mapping graph analysis performed on a mapping. Mapping steps are automatically created and maintained; you cannot add, delete, or rename them. A step can be one of the following types:

- PL/SQL
- SQL*Loader

Each type has related physical configuration parameters that you can configure using the Configuration Property inspector. Code can be generated for mapping steps in its associated language.

Setting Runtime Parameters

This section describes the runtime parameters.

Default Operating Mode: You can set your mapping to generate code using the following settings:

SET BASED–Enables Warehouse Builder to insert all of the data in a single SQL command. This option depends upon a correctly designed mapping. This setting assigns SQL as the implementation language to each operator unless an operator does not support SQL. If an operator does not support SQL, then all operators in the mapping use PL/SQL as the implementation language. You can use set based if you:

- Want to increase the speed of DML operations
- Are confident your data is clean

- Do not require extensive auditing

ROW BASED–Starts from the beginning of the mapping and assigns SQL as the implementation language to each operator. If the code generator finds an operator that supports PL/SQL, it assigns that language to all subsequent mapping operators. This setting offers the most auditing or debugging information since it is likely that the expressions or transformations in the mapping will be implemented in PL/SQL. You can use row based if:

- Some of your data may not load successfully but you still want to load other rows
- You want the maximum amount of runtime auditing

ROW BASED (TARGET ONLY)–The implementation language of the data target operators using this setting is always PL/SQL. When used with Set Based Fail Over, this setting allows the error rows to be stored into the audit table when data is being loaded into the target. Although this is mainly used when the source data is clean, there may be errors when loading it into the targets. For example, a foreign key constraint violation when loading a fact with a missing primary key at the dimension, or an incompatible implicit data type conversion due to the limited length, scale, or precision of the data targets.

Note: If you are using a flat file as a target, then you must use this setting. Your mapping will not generate code in any other default operating mode.

SET BASED FAIL OVER TO ROW BASED–The mapping is first executed in set based mode. If any error occurrs, the execution fails over to row based. If this operating mode cannot be implemented by the code generator, a runtime exception occurs.

SET BASED FAIL OVER TO ROW BASED (TARGET ONLY)–The mapping is first executed in set based mode. If any error occurrs, the execution fails over to row based (Target Only) mode. If this operating mode cannot be implemented by the code generator, a runtime exception occurs.

- **Default Audit Level:** The audit level that is used when executing the package. Audit levels dictate the amount of audit information that is 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–Statistical plus error information is recorded in runtime.

COMPLETE–All auditing information is recorded in runtime.

- **Default Purge Group:** The purge group used when executing the package. Each audit record in the runtime schema is assigned to the purge group specified.
- **Maximum Number of Errors:** The number for maximum errors reached that are used 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:** The commit frequency used when executing the package. Data is committed to the database after processing the number of rows specified in this parameter.
- **Bulk Processing:** This setting enables you to load incremental amounts of rows from a source to a target. If you set it to false, PL/SQL processes the data one row at a time. If you set it to true, PL/SQL processes the data grouped by the amount of rows specified by the Bulk Size property.
- **Analyze Statistics Percentage:** The percentage of rows to estimate 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 that is used for this analysis.

Setting Sources and Targets

The Configuration Properties inspector displays each operator in a mapping by name under the Sources and Target node. Each operator contains a set of properties you can edit.

- **Bound Name:** The name used by the code generator to identify this item. By default, it is the same name as the item.
- **Schema:** You can configure your mapping for Schema and Remote Schema Access. You can also link the mapping to a particular schema by clicking on the Schema field and entering a name. When you specify a schema by name, the PL/SQL and SQL generation handlers for the extract operators consider the property value when referring to the object name in the generated FROM clause.
- **Database Link:** Database links define the physical connection information for remote access that can be referenced by the Remote Access property of an Extract Operator. The mapping configuration enables you to link the mapping

to a database link in the warehouse module. Select a link by name from the drop-down list. See "Configuring Connection Information for Database Sources" on page 5-2 for information on setting database links for modules, and see "Creating Database Links" on page 9-12 for more information on database links.

- **Partition Exchange Loading:** Use this parameter to match the partitioning of your targets and enable PEL. By default, PEL is disabled for all mappings. Set the following:
  - **Enabled:** set to true to use PEL. the target table must be partitioned to use PEL.
  - **Granularity:** Select the level of granularity for your partitions.

  See "Configuring Partition Exchange Loading (PEL)" on page 9-22 for more information on PEL. See "Creating Partitions" on page 9-17 for information on partitioning.

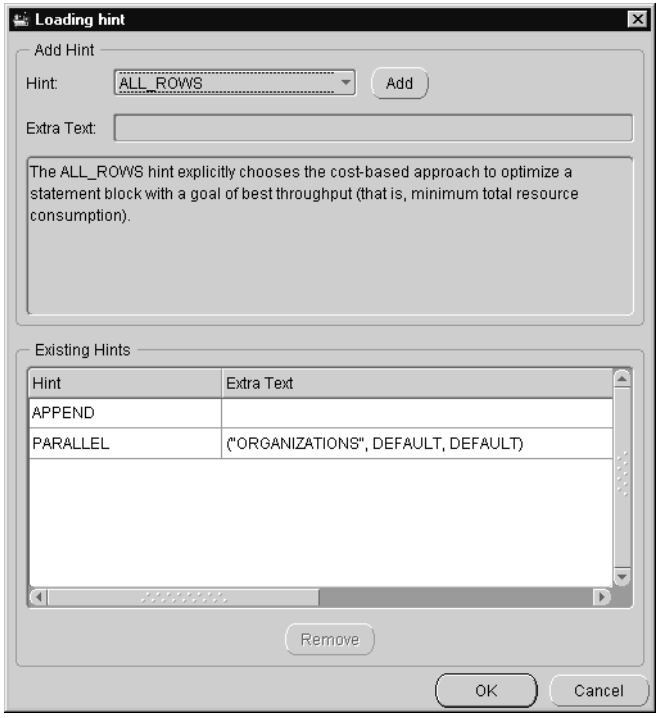
- **Hints:** This setting enables you to define extraction or loading hints. Application developers often develop insights into their data. For example, they know that a query runs much faster if a set of tables is joined in one order rather than another. Warehouse Builder can incorporate these insights into the generated SQL code packages as SQL Optimizer Hints.

  To define hints for a mapping:

  1. Expand the name of the operator for which you want to define a hint.
  2. Expand **Hints.**
  3. Click either an Extraction Hint or a Loading Hint field.
  4. Click the **...** button.

     The Extraction or the Loading Hint dialog displays.

Figure 8–8 Loading Hint Dialog



5. Select a **Hint** from the drop-down list.
6. Click **Add.**

   The hint appears in the **Existing Hints** field. Type additional text as appropriate in the **Extra Text** column.
7. Click **OK.**

   The editor includes the hint in the mapping definition.

For information on optimizer hints and how to use them, see *Oracle8i/9i Designing and Tuning for Performance.*

- **Constraint Management:** This groups the following settings:

  Enable Constraints: Setting this parameter to false disables all referential constraints to and from the target table. This speeds loading by omitting constraint checking by the database. After all the data is loaded, the constraints are automatically enabled by Warehouse Builder. This property compromises

referential integrity. If the data does not comply with the constraints, the constraint becomes invalid. In such cases, you have to manually resolve the referential integrity of the data. If set to the value true, then the REENABLE DISABLED_CONSTRAINTS clause is generated. This option automatically re-enables integrity constraints at the end of a direct-path load.

Exceptions Table Name: If set to a non-blank value, then the EXCEPTIONS table clause is generated. This option specifies a table that must exist when SQL*Loader is run. It is used to insert the records of all rows that have violated one of the integrity constraints when constraint re-enabling is processed during a direct path load.

- **SQL*Loader Parameters:** The SQL*Loader Parameters properties enable you to define the SQL*Loader options appropriate 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
  - Direct Path Load

  A conventional path load executes a SQL INSERT statement to populate tables in an Oracle database. A 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 *Oracle9i Database Utilities* for more information.

  When designing and implementing a mapping to extract data from a flat file using SQL*Loader, there are several places where you can choose values for properties that affect the generated SQL*Loader script.

  Each load operator in a map 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. Table 8–3 lists the INTO TABLE clauses associated with each load type.

Table 8–3 Loading Types and INTO TABLE Relationship

Loading Types	INTO TABLE
INSERT / UPDATE	APPEND

Table 8–3 Loading Types and INTO TABLE Relationship (Cont.)

Loading Types	INTO TABLE
DELETE/INSERT	REPLACE
TRUNCATE/INSERT	TRUNCATE
CHECK/INSERT	INSERT
NONE	INSERT

You can supply physical configuration information to affect the generated SQL*Loader output from either the configuration properties inspector or by setting the values in the physical step configuration. The following parameters are available in the Configuration Properties inspector:

- **Partition Name:** Indicates that the load is a partition-level load. Partition-level loading lets you load one or more specified partitions or subpartitions within 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 loaded selectively.
- **Sorted Indexes:** Identifies the indexes on which the data is presorted. This clause is allowed only for direct path loads. Because data that is 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. Instead, index entries are put into a separate, temporary storage 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 suggested for use when either of the following situations exists:

  - Available storage is limited
  - The number of records to be loaded is small compared to the size of the table (a ratio of 1:20 or less, is recommended)

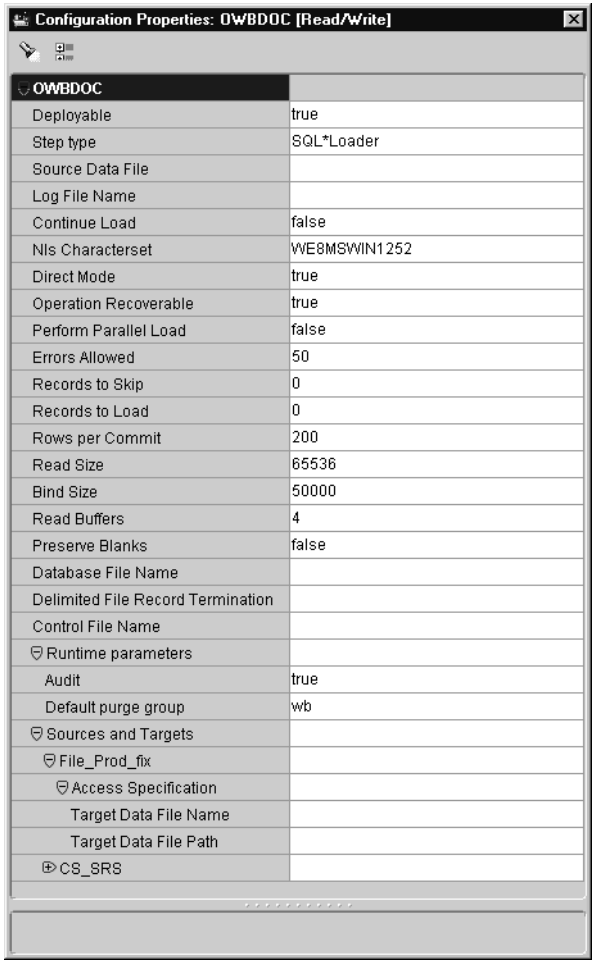
- **Trailing Nullcols:** Sets SQL*Loader to treat any relatively positioned columns that are not present in the record as null columns.
- **Records To Skip:** Invokes the SKIP command in SQL*Loader. SKIP specifies the number of logical records from the beginning of the file that should not be loaded. 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 Export supports 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 Physical Properties

When a mapping contains a flat file operator, the Configuration Properties inspector contains additional settings. When a mapping includes a Mapping Flat File operator as target, it generates a PL/SQL deployment code package.

Mappings containing Mapping Flat File operators as sources generate SQL*Loader scripts. When a mapping uses SQL*Loader, the Configuration Properties inspector also contains settings for defining source file data and a category called Default that contains script parameters.

Figure 8–9 Configuration Properties Inspector (Flat File Source)



To configure a mapping with a flat file:

1. Select a mapping object from the navigation tree.
2. From the **Edit** menu, select **Configure...** or right-click the mapping you want to configure and select **Configure...** from the pop-up menu.

   The Configuration Properties inspector displays.
3. Expand the **Runtime Parameters** node to set the deployment code generation.

See "Setting Runtime Parameters" on page 8-19 for information on these settings.

4. Expand the **Sources and Targets** node to set the file access information.

   See "Configuring Flat File Physical Properties" on page 8-26 for information on these settings.

5. If your flat file is being used as a source, then expand the **Source Data File** node to set data file information.

   See "Configuring Flat File Physical Properties" on page 8-26 for information on these settings.

6. If the flat file is being used as a source, then expand the **Default** node to set the control file information.

7. Close the window.

This section describes the configuration parameters for a SQL*Loader mapping.

Parameters affecting script type:

- **Continue Load:** If this parameter is set to true, then the generated SQL*Loader script is CONTINUE_LOAD. Otherwise, the SQL*Loader script 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.
- **Nls Characterset:** Specifies the character set to place in the CHARACTERSET clause.
- **Operation Recoverable:** Controls the output of the RECOVERABLE clause. True indicates that the load is recoverable. False indicates that the load is not recoverable and records are not recorded in the redo log.

Parameters affecting the OPTIONS clause:

- **Direct Mode:** Specifies 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.
- **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 this error limit is reached.

- **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 trailing white space intact when fields are specified with a predetermined size.
- **Database File Name:** If this parameter is set to a non-blank value, then the FILE= option is generated.

> **Note:** The value specified is enclosed in single quotes in the generated code. FILE specifies the database file from which the temporary segments are allocated for the entire session.

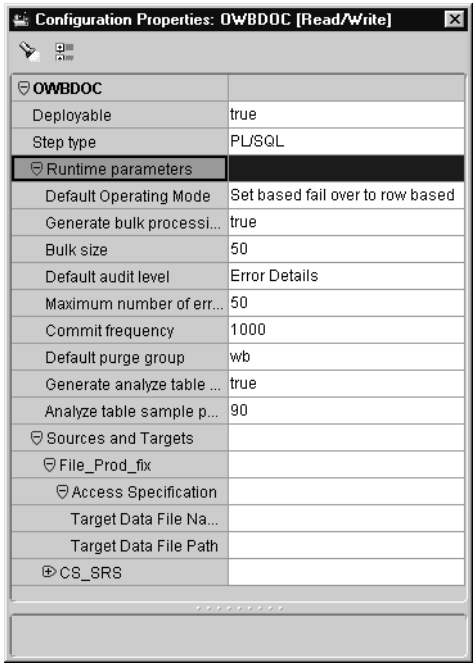
Parameters affecting the INFILE clause:

The Data Files configuration 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.

- **Data File Name:** The full physical filename of a data file to be loaded. The value entered is placed in the INFILE clause. You must enter the complete path name for the file in the syntax of the operating system on which the SQL*Loader script is deployed. The value specified is enclosed in single quotes in the generated code.
- **Bad File Name:** The full physical filename of a bad file that SQL*Loader creates to store records that cause errors during insert or are improperly formatted. The value entered is placed in the BADFILE clause. You must enter the complete path name for the file in the syntax of the operating system on which the SQL*Loader script is deployed. The value specified is enclosed in single quotes in the generated code.
- **Discard File Name:** The full physical filename of a discard that SQL*Loader creates to store records that are neither inserted into a table nor rejected. The value entered is placed in the DISCARDFILE clause. You must enter the complete path name for the file in the syntax of the operating system on which the SQL*Loader script is deployed. The value specified is enclosed in single quotes in the generated code.
- **Discard Max:** The value entered, if greater than 0, is used to generate the DISCARD n clause. It specifies the number of discards that will terminate the load.

For more information on each SQL*Loader option and clause, see *Oracle8i Utilities* or *Oracle9i Database Utilities*.

Figure 8–10 Configuration Properties Inspector (Flat File Target)



To configure a mapping with a flat file as a target:

1. Click **Default Operating Mode** and select **Row based (target only)** from the drop-down list.

   Note: Your mapping will not generate code in any other default operating mode.

2. Expand the **Sources and Targets** node, then the **Flat File** node, and then the **Access Specification** node.
3. Enter the name of the Target Data File.
4. Enter the path of the target file into the **Target Data File Path** field.

You must first set this path in the Init.ora file for your warehouse instance. The UTL_FILE_DIR parameter contains the path setting. For example, to load files in D:\Data\FlatFiles\File1.dat, set the parameter to:

```
UTL_FILE_DIR=D:\Data\FlatFiles\
```

You can create more than one valid path setting by copying and editing this line, ensuring the copied lines are consecutive:

```
UTL_FILE_DIR=D:\Data\FlatFiles\
UTL_FILE_DIR=E:\OtherData\
```

You can bypass this by setting the parameter to:

```
UTL_FILE_DIR=*
```

Do not bypass this setting for production databases.

Configuring SAP File Physical Properties

This section describes how to configure the Step Type properties and Runtime Parameters specific to ABAP mappings. The configuration of a PL/SQL mapping from a transparent table in an SAP application is the same as the configuration of any other PL/SQL mapping.

Setting the Step Type

This parameter enables you to choose the type of code you want to generate for your SAP mappings.

To choose the step type:

1. Click the **Step Type** field and click the **...** button.

   The Step Type dialog displays.

2. From the drop-down list, select the type of code you want to generate: ABAP or PL/SQL scripts (for transparent tables only).

3. Click **OK.**

Setting the Runtime Parameters

The following is the list of runtime parameters for SAP mappings and their recommended corresponding values. The default values are recommended for most parameters.

- **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.

Validating and Generating a Mapping

After you have set the Configuration Properties you are ready to validate and generate the scripts that deploy the mapping to the repository. Warehouse Builder generates code that executes the mapping. It can generate the following types of code:

- PL/SQL for executing within the database
- SQL Loader for flat files
- TCL for use with Oracle Enterprise Manager

Warehouse Builder selects the language based on the operators and objects you use in your mapping.

Validating a Mapping

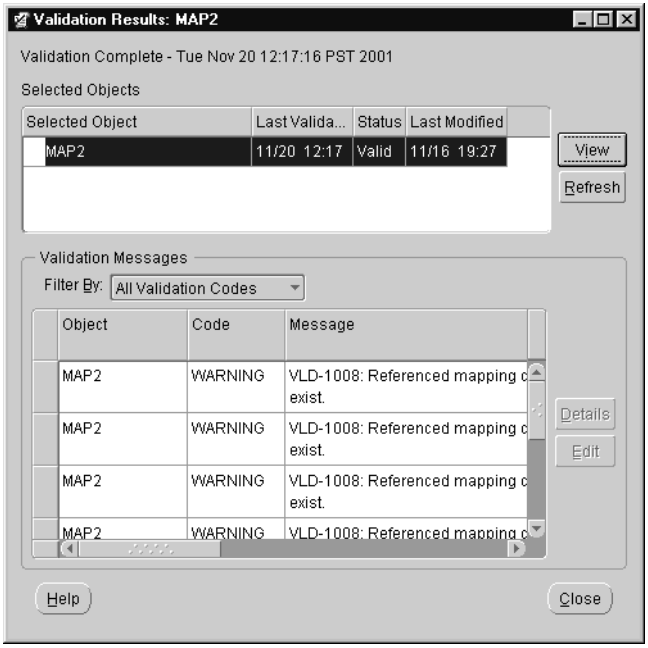
You can validate your mapping prior to generating it to check for errors. The validation procedures verify foreign key references, object references, data type matches, and other properties in your mapping. See "Viewing the Generated Code for a Mapping" on page 8-37 for information on viewing generated code in the Mapping Editor, and "Validating Definitions and Generating Scripts" on page 9-23 for more information on validation and generation from the Module Editor.

To validate a mapping:

1. Select a mapping in the Warehouse Module Editor.
2. From the **Mapping** menu, select **Validate...**.

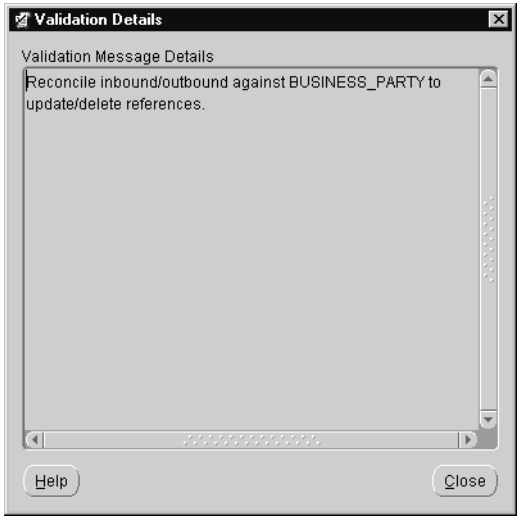
   The Validation Results dialog displays.

Figure 8–11 Validation Results Dialog



3. Click **Details** to see the full message that appears in the Validation Message field.

Figure 8–12 Details Window



If you have errors, the Validation Details window title bar lists their numbers. The Validation Details window contains suggestions for correcting them. You can open an editor to make these corrections by clicking **Edit** in the Validation Results Window. See "Validating Definitions" on page 9-23 for more information on the Validation Results window.

Generating a Mapping

You can generate the code for a mapping from the:

- Module Editor, which creates scripts with the generated code that you can deploy and run.
- Mapping Editor, which generates code that you can inspect from a code viewer. Code generated from within the Mapping Editor does not remain in the repository as an object. Also, Mapping Editor code excludes auditing information and deployment code.

When you generate a mapping, you are creating the SQL code that performs the DML and DDL commands necessary to move the data from the sources to the targets defined in your mapping. The code generator also validates the mapping before creating the code that implements your ETL design. See "Generating Scripts" on page 9-25 for more information on generating scripts.

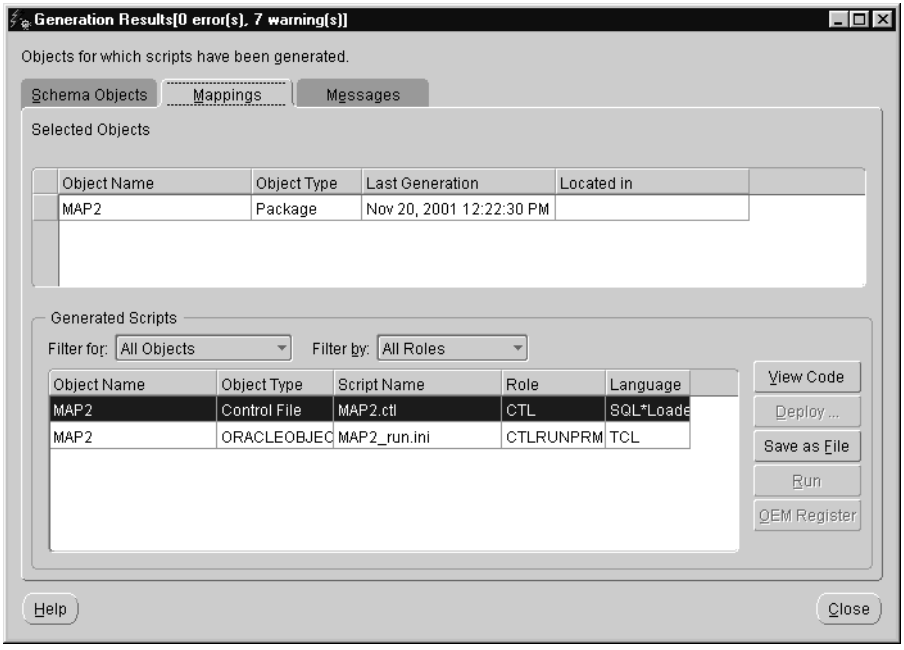
Note: If you generate code at any time during your Warehouse Builder session, multi-user locking remains in effect until after you commit your changes.

To generate code from the mapping editor:

1. Select a mapping in the Warehouse Module Editor.
2. From the **Module** menu, select **Generate** or right-click a mapping and select **Generate** from the pop-up menu.

   Warehouse Builder generates the code for the mapping and displays the Generation Results dialog.

Figure 8–13 Generation Results Dialog



For a Mapping that extracts records from a flat file and loads them as rows into a target table, the generated code results in a SQL*Loader control file. The control file reflects a mapping definition from a flat file to a table source and includes a date stamp.

Viewing the Generated Code for a Mapping

You can view generated code from the Mapping Editor at any stage of your mapping process. Warehouse Builder validates this code automatically before generating it. See "Validating Definitions and Generating Scripts" on page 9-23 for more information on validation and generation from the Module Editor.

Note: If you generate code at any time during your Warehouse Builder session, multi-user locking remains in effect until after you commit your changes.

Viewing Intermediate Results

You can view the generated code for a mapping from the Mapping Editor as an intermediate result. This type of code generation produces mapping code up to the selected attribute group in the mapping. The Code Viewer displays the code that goes into an input attribute group, the code that goes out of an output attribute group, and the load code is generated for a terminating input group.

These incoming, outgoing, and load types of code are called aspects. You select aspects from the Code Viewer View menu. You can view one aspect at a time. For example, an output attribute group does not contain the input intermediate code result.

You can leave the Code Viewer open and select any attribute group on the mapping canvas. The Code Viewer displays the code generated for the selected attribute group.

To view intermediate results:

1. Select an attribute group in an operator.
2. From the **Mapping** menu, select **Generate**, and then **Intermediate Result** or right-click the mapping on the Module Editor navigation tree, select **Generate** and then **Intermediate Result** from the pop-up menu

Viewing Code from the Module Editor

You can generate simplified code from the mapping editor showing how data extraction, transformation, and load are performed. The generated code does not include support for auditing or bulk-processing.

The code generator only generates code that handles the loading type specified in the data target operator. For example, if you choose INSERT/UPDATE as the loading type, only INSERT and UPDATE code appears in the generated code.

> **Note:** The system can generate code for multiple strategies. For example, if a step can be implemented in set-based and row-based operating mode, the Code Viewer will include both modes.

To generate mapping code:

1. From the **Module** menu, select **Generate...** or right-click the mapping and select **Generate...** from the pop-up menu.
2. Select **Generate create scripts** in the Generation Mode dialog.

The generated code contains all possible strategies that can be generated for that mapping. For PL/SQL mappings, a runtime parameter determines which operating mode to use when running the package.

> **Note:** If a mapping contains a Mapping View operator, Warehouse Builder does not generate a DDL statement for that view. Warehouse Builder generates a regular PL/SQL package with the assumption that the view can be updated.
>
> No error checking is done to verify if the view can be updated.
>
> Check to ensure that the view can be updated before including this view in the map. Warehouse Builder validation or generation treats views or materialized views the same as regular tables.

9

Configuring, Generating, and Deploying

This chapter describes how to generate a physical warehouse instance from a logical warehouse design. You begin this process by configuring the properties of the target warehouse module. You can then validate the definitions and generate the scripts used to create the warehouse objects and the mapping scripts used to load the warehouse. When you deploy the scripts, the warehouse objects are created and the mapping scripts are stored in the warehouse instance. This chapter also describes how to upgrade the warehouse.

This chapter includes the following topics:

- Configuring a Physical Instance
- Creating Database Links, Indexes, and Partitions
- Validating Definitions and Generating Scripts
- Deploying Scripts
- Upgrading the Warehouse

Configuring a Physical Instance

When you configure a physical instance, you specify which objects in the logical warehouse design to physically deploy. You also define the physical characteristics of the warehouse objects. Physical characteristics are determined by a set of parameters stored in the configuration properties inspector for the object. Most parameters have default values.

To configure a data warehouse, you must define the physical properties listed in Table 9–1, which summarizes the properties you can configure for each object deployed to the physical warehouse.

Table 9–1 Physical Configuration Properties

Warehouse Object	Properties
Warehouse Module	Database link definitions, generation preferences, the job name to register with Oracle Enterprise Manager, runtime and target directory names, and warehouse module identification properties.
Dimension	Indexes, partitions, partition keys, constraints, columns, performance, parallel, partition parameters, table space used to store, generation options for time dimensions, generation options, and identification.
Fact	Indexes, partitions, partition keys, constraints, columns, performance, parallel, partition parameters, table space used to store, and identification.
Mapping	These parameters are described in Chapter 8.
Materialized View	Indexes, partitions, partition keys, performance, parallel, partition parameters, materialized view parameters, table space used to store, and identification.
Sequence	Sequence Parameters and Identification.
Table	Indexes, partitions, partition keys, constraints, columns, performance, parallel, partition parameters, table space used to store, and identification.
Transformation Libraries	Identification.
View	Identification.

Note: Database links, indexes, and partitions are physical objects. They are not created during the logical design. They must be created in the Configuration Properties inspector before you can configure their parameters. See "Creating Database Links, Indexes, and Partitions" on page 9-11 for more information.

Setting up Configuration Properties

To set up configuration properties:

1. Select the module you want to configure from the navigation tree in the Warehouse Module Editor.
2. From the **Edit** menu, select **Configure.**

   The Configuration Properties inspector displays.
3. Choose the parameter you want to configure and click the space to the right of the parameter name to edit the value.

   Depending on the parameter, you can either select an option from a drop-down list or type a value.

Guidelines for Configuring Warehouse Modules

Table 9–2 describes the categories of parameters in the Configuration Properties inspector for Warehouse Modules.

Figure 9–1 Configuration Properties Inspector for Modules

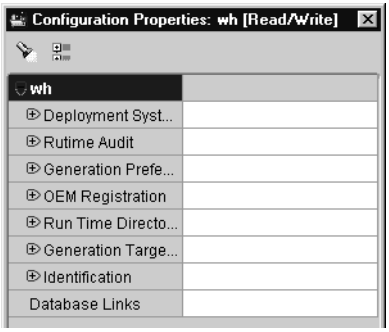


Table 9–2 Warehouse Module Configuration Parameters

Category	Parameter	Description
Deployment System Type	Target Database Type	Defines the target database type. Select either Oracle8*i* or Oracle9*i* from the drop-down list
Runtime Audit	Audit Level	Defines how the audit data is collected and stored in the Runtime Audit Tables. Select one of the following options from the drop-down list: ■ **None:** No audit data is collected. ■ **Statistics:** Records basic runtime statistics such as the number of rows processed. ■ **Error Details:** Records basic runtime statistics and detailed information about rows containing errors. ■ **Complete:** Records basic runtime statistics, detailed information about rows containing errors, and the rowid of every row.
Generation Preferences	End of Line	Defines the end of line markers. This is dependent on the platform to which you are deploying your warehouse. For UNIX, use \n, and for NT, use \r\n.
OEM Registration	OEM Job Name	Type in the suffix that is appended to the name of a mapping script when it is registered with Oracle Enterprise Manager. The default is _job.
Runtime Directories	Log Directory	Log directory for the SQL*Loader. The default is log\.
Runtime Directories	Archive Directory	Not currently used. The default is archive\.
Runtime Directories	Receive Directory	Not currently used. The default is receive\.
Runtime Directories	Input Directory	Not currently used. The default is input\.
Runtime Directories	Invalid Directory	Directory for Loader error and rejected records. The default is invalid\.
Runtime Directories	Work Directory	Not currently used. The default is work\.
Runtime Directories	Sort Directory	Not currently used. The default is sort\.
Runtime Directories	Queue Listener Directory	Location of Warehouse Builder Work Flow Queue Listener. The default is workflow\.

Table 9–2 Warehouse Module Configuration Parameters (Cont.)

Category	Parameter	Description
Runtime Directories	Queue Listener Host	Computer name for the Work Flow Queue Listener. The default is the local host.
Generation Target Directories	TCL Directory	Location of the Tcl scripts that can be registered with Oracle Enterprise Manager. These are used to schedule and run initial and incremental loader jobs. The default is tcl\.
Generation Target Directories	DDL Directory	Location of the scripts that create database objects for the target schema. The default is ddl\.
Generation Target Directories	DDL Extension	File name extension for DDL scripts. The default is .ddl.
Generation Target Directories	DDL Spool Directory	Buffer location for DDL scripts during the script generation processing. The default is ddl\log.
Generation Target Directories	LIB Directory	Location of scripts that generate Oracle functions and procedures. The default is lib\.
Generation Target Directories	LIB Extension	Suffix appended to a mapping name. The default is .lib.
Generation Target Directories	LIB Spool Directory	Location of scripts that generate user-defined functions and procedures. The default is lib\log\.
Generation Target Directories	PL/SQL Directory	Location of the PL/SQL scripts. The default is pls\.
Generation Target Directories	PL/SQL Extension	File name extension for PL/SQL scripts. The default is .pls.
Generation Target Directories	PL/SQL Run Parameter File	Suffix for the parameter script in a PL/SQL job. The default is _run.ini.
Generation Target Directories	PL/SQL Spool Directory	Buffer location for PL/SQL scripts during the script generation processing. The default is pls\log\.

Table 9–2 Warehouse Module Configuration Parameters (Cont.)

Category	Parameter	Description
Generation Target Directories	Loader Directory	Location of control files. The default is ctl\.
Generation Target Directories	Loader Extension	Suffix for the loader scripts. The default is .ctl.
Generation Target Directories	Loader Run Parameter File	Suffix for the parameter initialization file. The default is _run.ini.
Generation Target Directories	Pure Extract Run Parameter File	Suffix for the parameter script in a Pure Extract job. The default is _run.ini.
Generation Target Directories	Pure Integrate Run Parameter File	Suffix for the parameter script in a Pure Integrate job. The default is _run.ini.
Identification	Name, Previous Deployment Name and Deployable	Contains general information about the dimension, fact, or table including the name, the name that it was last deployed with, and whether it is deployable.

Guidelines for Configuring Dimensions, Facts, and Tables

Warehouse Builder generates two scripts for a dimension: one to create the dimension object and the other to create the underlying table. Additional scripts are generated when indexes are defined on the dimension table, or when the definition is generated with the Time Dimension Wizard. The configuration parameters for a dimension specify how these scripts are generated and whether the dimension is deployed.

Figure 9–2 Configuration Properties Inspector for a Dimension

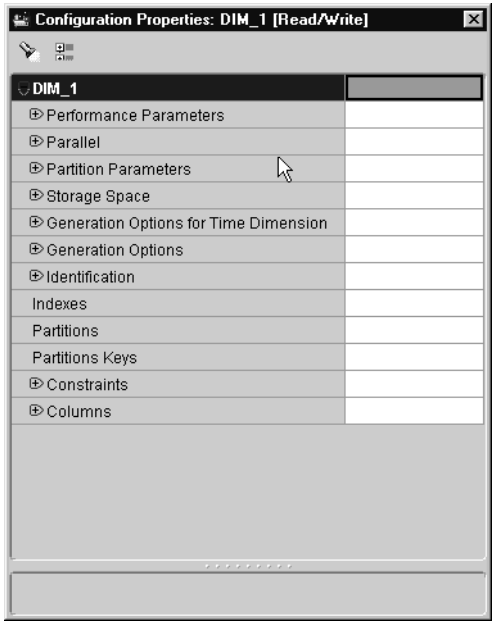


Table 9–3 defines the configurations parameters for dimensions, facts, and tables. For information on creating indexes, partitions, and partition keys see "Creating Database Links, Indexes, and Partitions" on page 9-11.

Table 9–3 Dimension/Fact/Table Configuration Parameters

Category	Parameter	Description
Performance Parameters	Log to Redo Log File	Recovery requirements for a data warehouse are usually less strenuous than for a transaction system. Set to nologging to improve performance. The default is logging.
Performance Parameters	Analyze Table: Estimate Percent	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. To improve performance, use a high number for a large dimension table and a low number for a small dimension table. The default is 99.

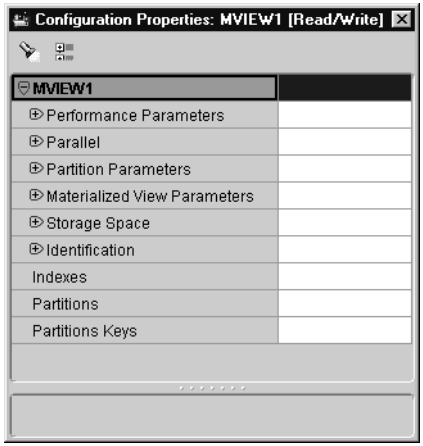
Table 9–3 Dimension/Fact/Table Configuration Parameters (Cont.)

Category	Parameter	Description
Parallel	Parallel	Enables parallel processing when the table is created. If you are using a single CPU or a non-Oracle database, set to nonparallel to improve performance. The default is parallel.
Partition Parameters	Hash SubPartition Number and Store in Tablespace	Defines the partition key and method. Select Hash or Range. These parameters only apply if you create hash partitions. For more information, see "Creating Partitions" on page 9-17.
Storage Space	Tablespace	Defines the name of each tablespace. A dimension table can reside in one or more tablespaces if partitions are defined. If you do not use partitions, then the storage space resides on a single tablespace.
Generation Options for Time Dimension	Start and End Date	When you create a definition for a dimension using the Time Dimension Wizard, Warehouse Builder generates an insert statement to load the underlying table. The Start and End Date parameters determine the range of dates.
Generation Options	Generate Table	Specifies whether or not to create the dimension table. The default is true.
Generation Options	Generate Dimension	Specifies whether or not to generate a script to create the object. Set this parameter to true only when the warehouse requires the hierarchy and level information. Create this information for an instance that relies on materialized views that reference this dimension.
Identification	Name, Previous Deployment Name and Deployable	Contains general information about the dimension, fact, or table including the name, the name that it was last deployed with, and whether it is deployable.

Guidelines for Configuring Materialized Views

Warehouse Builder generates one script to create a materialized view and another for its foreign key reference constraints. This is the same generation process used for views and tables. Additional scripts are generated when indexes are defined on the view.

Figure 9–3 Configuration Properties Inspector for Materialized Views



The Configuration Properties inspector for a materialized view contains most of the same categories as dimensions, facts, and tables. See Table 9–3 for the parameter descriptions. Table 9–4 defines parameters specific to materialized views.

Note:

- Warehouse Builder does not generate code for a view if its query text is not included in its configuration properties and if its columns are not defined.
- Warehouse Builder generates a create materialized view statement to deploy the view even if its syntax is invalid. Warehouse Builder does not check the syntax of the select statement used to define a materialized view.

Table 9–4 Materialized View Parameters

Parameter	Description
Build	**Immediate:** Populates the view when it is created. **Deferred:** Delays population until the next refresh operation. The default is Immediate.

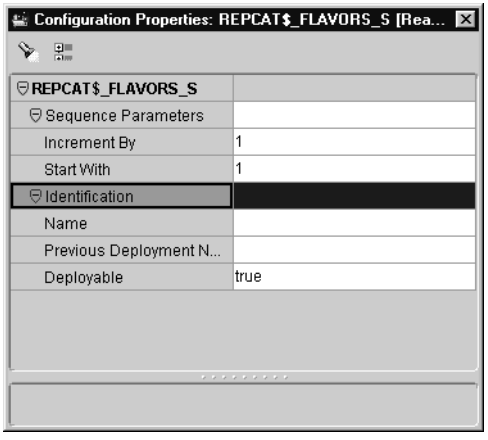
Table 9–4 Materialized View Parameters (Cont.)

Parameter	Description
Refresh	**Complete:** Specifies the complete refresh method implemented by executing the view's query. **Fast:** Specifies the incremental refresh method which refreshes the view according to changes that have occurred to the master tables. **Force:** Specifies that when a refresh occurs, Oracle8*i*/9*i* performs a fast refresh if possible or a complete refresh otherwise. The default is Complete. For more information on restrictions to the refresh parameter, see the Oracle8*i*/9*i* documentation.
Query Rewrite	**Enable:** Marks the view eligible for query rewrite. **Disable:** Marks the view ineligible for query rewrite. The default is Enable.
Base Tables	Type the exact name of the base tables separated by commas. If the tables were created in Preserve Case, enclose table names with quotes. The default is blank.

Guideline for 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.

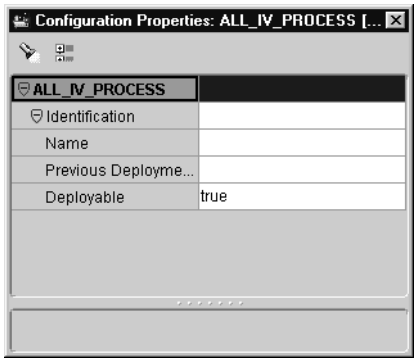
Figure 9–4 Sequences Configuration Properties Inspector



Guidelines for Configuring Views

Warehouse Builder generates a script for each view defined in a warehouse module. You can configure whether to deploy specific views or not.

Figure 9–5 View Configuration Properties



Note:

- Warehouse Builder does not generate code for a view if its query text is not included in its configuration properties and if its columns are not defined.
- Warehouse Builder generates a create view statement to deploy the view even if its syntax is invalid. Warehouse Builder does not check the syntax of the select statement used to define a view.

Creating Database Links, Indexes, and Partitions

Physical objects such as database links, indexes, partitions, and partition keys cannot be created when you are designing the warehouse. If you want to create these objects in your database when you deploy, you must define the configuration properties before deploying the warehouse. These objects can only be created in relation to certain objects. Database links can only be created in the warehouse module Configuration Properties inspector. Indexes, partitions, and partition keys can only be created in the dimension, fact, table, and materialized view Configuration Properties inspectors.

Creating Database Links

Target schemas pull data from database sources using PL/SQL packages. These packages rely on database links to connect with the sources. A definition for a database link must be created, configured, and deployed for each database source. The database link is deployed in the target schema.

Each time you create a mapping that pulls data from a database source, Warehouse Builder verifies that a definition for the required database link exists. If the definition does not exist, Warehouse Builder creates one from the connection information stored in a source module for a mapping. This means, the default parameter values for a database link correspond with the connection information in a source module.

When you deploy a physical instance, the database links generated by Warehouse Builder must point to the correct sources. For example, if a test application system was used to develop the logical warehouse but the physical instances use real data stored in production systems, then you must reconfigure the definitions to reflect this change.

Definitions for database links are visible in the Configuration Properties inspector for the warehouse module. You can specify the host machine using a SQL*Net connect string or by providing the host name, port, and ID of the database instance.

Note: If your source and target tables are in the same instance, you can link directly to the schema. You set this up in the mapping configuration. When the source module references an Oracle Designer repository, you must configure the database link to point to the actual database and not the Designer repository.

- A large number of dimensions
- A sparsely populated fact table
- Queries where not all dimension tables have constraining predicates
- Queries where multiple dimension tables are joined to a single column in the fact

You can create bitmap indexes on facts, dimensions, tables, and materialized views.

To create bitmap indexes:

1. Select a table, fact, dimension, or materialized view from the Warehouse Module Editor.

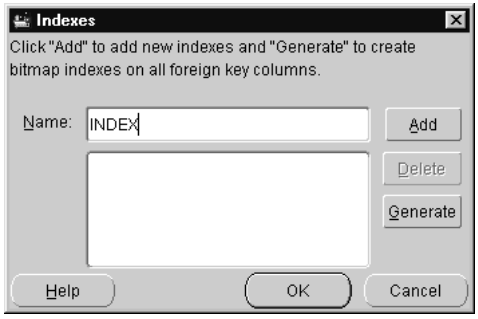
2. From the **Edit** menu, select **Configure** or right-click the table and select **Configure** from the pop-up menu.

   The Configuration Properties inspector for the entity displays.

3. Select **Indexes** and then the **...** button located in the right column.

   The Indexes dialog displays.

Figure 9–6 Indexes Dialog



4. Type a name in the **Name** field and click **Add.**

   Repeat this step for each index you want to create.

5. Click **Generate.**

   The Impact report for Bitmap Index creation dialog displays.

   Before indexes are generated, Warehouse Builder checks the table for foreign keys. Warehouse Builder does not generate indexes if:

   - There are no foreign keys on the table.

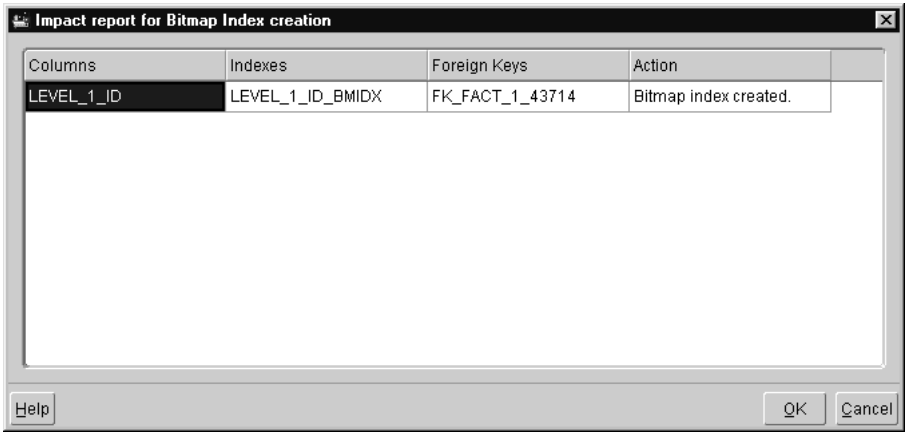
     In this case, Warehouse Builder informs you that no index can be created because there are no foreign keys on the table.

   - There are foreign keys on the table but there are no columns defined for them.

   - 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.

Figure 9–7 Impact Report For Bitmap Index Creation Dialog



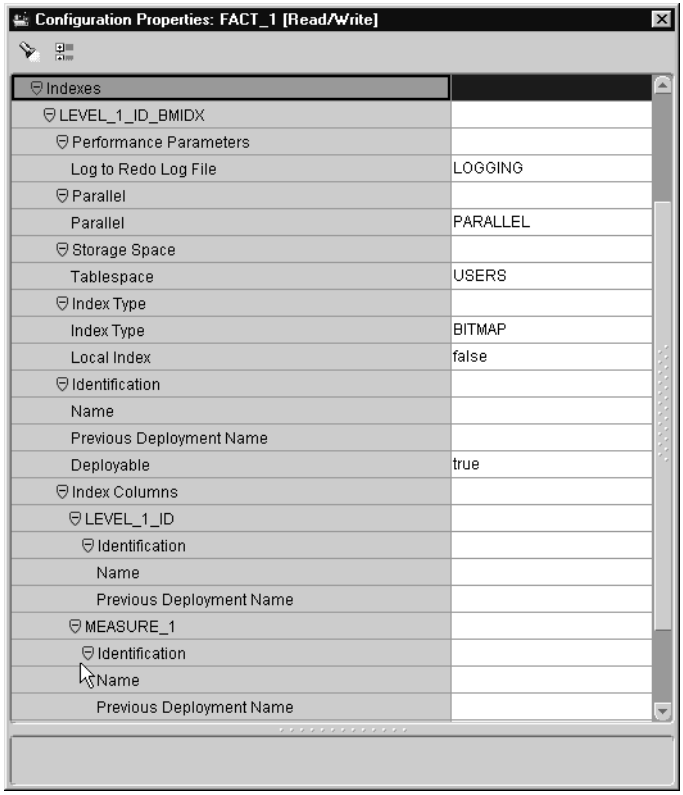
6. Click **OK** if the indexes are correct, otherwise click **Cancel.**

After you create indexes, you can edit their parameters using the Configuration Properties inspector.

To configure bitmap indexes:

1. Click the **Index** node in the Configuration Properties inspector.

Figure 9–8 Configuration Properties Inspector Showing Indexes



2. Set the **Index Type** to **Bitmap**.

Table 9–5 describes the remaining parameters for indexes.

Table 9–5 Index Parameters

Parameter	Default	Description
Index Columns	None	The columns in the index.
Log to Redo	Logging	Indicates whether the creation of the index logs into the redo log file. Set to nologging to improve performance.
Parallel	Parallel	Enables parallel processing when the table is created. If you are using a single CPU or a non-Oracle database, set to nonparallel to improve performance.

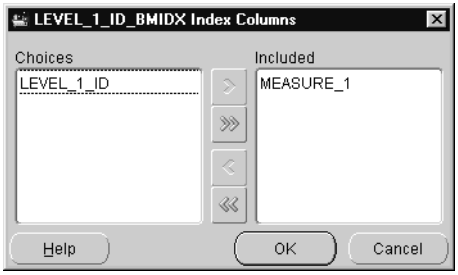
Table 9–5 Index Parameters (Cont.)

Parameter	Default	Description
Tablespace	Users	Specifies the tablespace to hold the index in.
Index Type	Bitmap	The type of index: bitmap, unique, or none.
Local Index	False	Specifies that the index is partitioned on the same columns, with the same number of partitions and the same partition bounds as the table.
Name	blank	The name of the index.
Previous Deployment Name	blank	Previous name of the index when it was last deployed. This is used during a warehouse upgrade.
Deployable	true	Specifies if this index is deployable.

3. Select the Index Columns:
   a. Click the **...** button in the **Index Columns** field.

      The Index Columns dialog displays.
   b. Select the columns you want to include from the **Choices** list.
   c. Click the right arrow to move them to the **Included** list.
   d. Click **OK.**

Figure 9–9 Index Columns Dialog



4. Close the Configuration Properties inspector.

Note: If you alter or delete foreign key columns, Warehouse Builder does not maintain the accuracy of indexes. You must regenerate indexes to reflect foreign key column changes.

For information about indexing strategies, see:

- *Oracle8i/9i Data Warehousing Guide*
- *Oracle8i/9i Database Performance Guide and Reference*
- *Oracle Enterprise Manager Database Tuning with the Oracle Tuning Pack*

Creating Partitions

Partitions are created on tables to improve query and load performance and to simplify the management of physical storage. Transportable tablespaces are introduced for performance reasons and to transport data across Oracle8*i/9i* databases. A transportable tablespace is the fastest means to move a large volume of data between two Oracle8*i/9i* databases. Dimension tables and materialized views occupy storage space and are candidates for intelligent partitioning.

You can create and configure definitions for partitions from the Configuration Properties inspector. A table can be partitioned using hash, range, or a combination of hash and range partitions.

Figure 9–10 Configuration Properties Inspector Showing Partitions

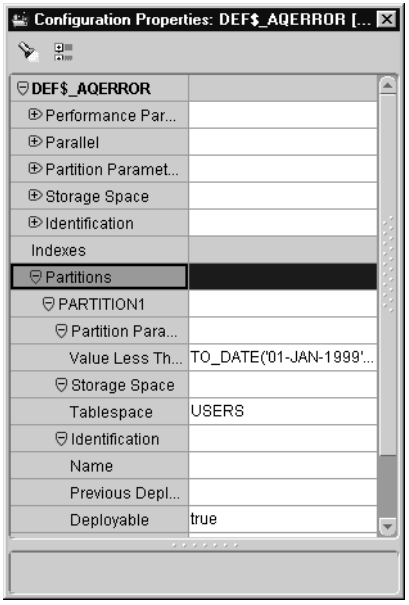


Table 9–6 describes the parameter settings for partitions.

Table 9–6 Partition Parameters

Parameter	Default	Description
Value Less Than	TO_ DATE('01-JAN-1999', 'DD-MON-YYYY')	Specify the non-inclusive upper bound for the current partition. The 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. MAVALUE specifies a maximum value that sorts higher than any other value, including NULL.
Tablespace	USERS	The value specified is the actual physical attribute of the segment associated with the table. For partitioned tables, the value specified for TABLESPACE is the default physical attribute of the segments associated with all partitions specified for the table.
Name	Blank	Name of the partition.

Table 9–6 Partition Parameters (Cont.)

Parameter	Default	Description
Previous Deployment Name	Blank	Previous name of the partition when it was last deployed. This is used during a warehouse upgrade.
Deployable	True	Specifies if this partition is deployed.

In order to use Partition Exchange Loading, you need to set key constraints on DATE columns. This column must be part of a primary or unique key on the table, an it must be the first column if it is segmented. This key must be index enabled and that index must be set on the same columns as the primary or unique key. See "Configuring Partition Exchange Loading (PEL)" on page 9-22 for information on PEL.

Table 9–7 describes the partition key parameters.

Table 9–7 Partition Key Parameters

Parameter	Default	Description
Type	Range	Can be Range or Hash.
Name	Blank	Name of the partition key.
Previous Deployment Name	Blank	Previous name of the partition key when it was last deployed. This is used during a warehouse upgrade.
Deployable	True	Specifies if this partition key is deployable.

To set partitions on a table:

1. Open the Configuration Properties inspector for the target table.
2. Click the **Partitions** field and then click the **...** button.

   The Partitions dialog appears.

Figure 9–11 Partitions Dialog

3. Click the **Name** field and enter a partition name.
4. Click **Add.**

   You can delete a partition name from the list by highlighting it and clicking **Delete.**

   A partition naming convention enables the PEL to automatically construct the correct partition name based on freshly loaded data. The partition name must contain information on all the time levels leading to the intended granularity. This convention also enables Warehouse Builder to calculate the Value Less Than property automatically. Table 9–8 describes the PEL naming convention.

Table 9–8 The Warehouse Builder Partition Naming Convention

Name Pattern	Example
Ydddd	for example Y1999
Ydddd_Qd	Y1999_Q1)
Ydddd_Qd_Mdd	Y1999_Q1_M02)
Ydddd_Qd_Mdd_Ddd	Y1999_Q1_M02_D03)
Ydddd_Qd_Mdd_Ddd_Hdd	Y1999_Q1_M02_D03_H16)
Ydddd_Qd_Mdd_Ddd_Hdd_Mdd	Y1999_Q1_M02_D03_H16_M20)

   If the partition name does not match one of the above patterns, then the default value is TO_DATE ('01-jan-1999','dd-mon-yyyy').

5. Click **OK.**

Using the partition name and the Value Less Than property, Warehouse Builder generates a DDL script for setting the partitioned table. The Value Less Than property defines the contents of the partition.

To edit the partition values:

1. Open the Configuration Properties inspector for the target table.
2. Click the partition you want to edit.
3. If you want to edit the partition boundary value:
   a. Click the **Value Less Than** field.
   b. Enter a value.
4. If you want to rename a partition:
   a. Click the field of the partition you want to rename and click the **...** button.

      The Partitions dialog appears.

Figure 9–12 Partitions Dialog

   b. Highlight the value in the rename field and enter a value.
   c. Click **OK.**
5. Close the Configuration Properties inspector.

Defining Hash Partitions

To create a hash partition, specify the hash key, number of partitions, and tablespace names. Oracle8*i*/*9i* partitions the storage space and stores rows according to a hash algorithm.

Defining Range Partitions

To create a set of range partitions, you specify a range key, names for the partitions, and a set of tablespaces. Oracle8*i*/*9i* then stores rows in a partition according to the specified ranges.

Note: If you are using Partition Exchange Loading, refer to the Oracle8*i*/*9i* documentation to set up range partitions.

Defining a Range with Hash Subpartitions

You can create a set of range partitions that have hashed subpartitions. Create a range partition key, a set of partitions, and then create a hash partition key to specify the number of logical partitions.

Configuring Partition Exchange Loading (PEL)

You can increase the speed at which data loads by using the Oracle8*i*/*9i* server partitioning swapping capability. The PEL technique performs data loading by exchanging the partition identities of a target table with a temporary table. Data loads into the temporary table in parallel or using direct path units. If this load completes, then the temporary table holding the new data takes over the identity of one empty partition from the user-defined target table. At the same time, the empty partition assumes the identity of the source table. The exchange process is a DDL operation and involves no data movement. PEL is useful for:

- Loading relatively small amounts of data into a target containing a much larger amount of historical data.
- Incremental inserts spanning only one empty partition.
- When a target is partitioned by DATE.
- Repetitive data loading into the same partition in a target table.

If you have an Oracle8*i* warehouse module, then you must use local indexes in order to take advantage of PEL. Local indexes require that all indexes be partitioned like the table. When the temporary table is swapped into the target table using the PEL, so are the identities of index segments. There is no need to manipulate actual index blocks.

To set partition exchange loading:

1. Open the Configuration Properties inspector for the mapping.
2. Expand the **Sources and Targets** node.
3. Expand node for the name of the mapping target operator.
4. Expand the **Partition Exchange Loading** node.
5. Click the **Enabled** field and select **True** from the drop-down list.

6. Click the **Partition Granularity** field and select the granularity level from the drop-down list.

   This property must match the partition granularity of your target table.

Validating Definitions and Generating Scripts

After you configure a set of definitions for a physical instance, you need to validate them before you generate and deploy objects and scripts to a physical instance. This section describes how to validate and generate a selected set of definitions.

Validating Definitions

Validation works as an error reporting tool for the data warehouse loading scripts. When you generate deployment code, Warehouse Builder validates it before deploying it to the repository. A set of configured definitions must be validated before scripts are generated for these reasons:

- Incorrect data warehouse definitions

  You can create incorrect definitions. For example, if you add a foreign key constraint to a fact table that does not reference a column in a dimension table, Warehouse Builder stores the definition in the warehouse module. The definition fails the validation process until the reference is satisfied.

- Data type mismatches

  You can create definitions with data type mismatches. When the validation process runs, a warning message displays.

- Metadata Import Utility

  The Import Utility can include definitions that are malformed for a number of reasons. For more information, see "Metadata Import Utility" on page 11-7.

Validation can apply to objects that are not generated. For example, if you may need to validate a source module to understand a foreign key constraint violation. Only target modules can generate code.

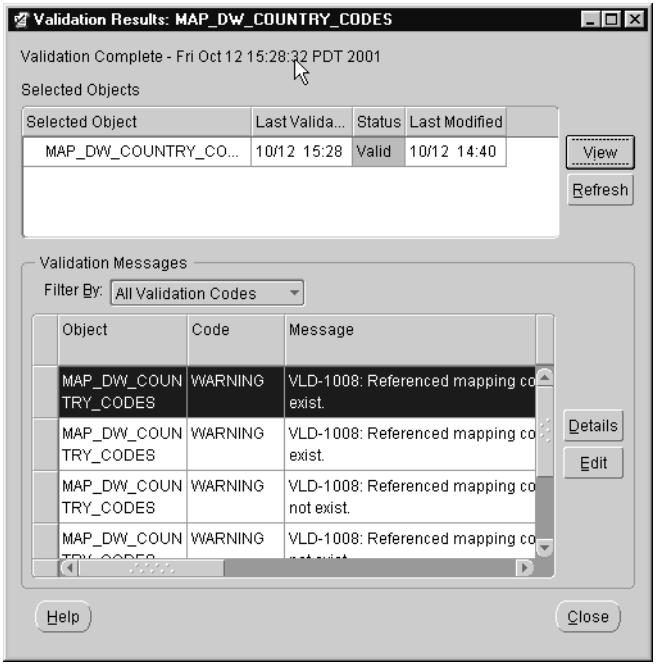
If you validate and generate in read-only mode, then the generated code does not remain in the repository. An alert dialog informs you if you are generating in read-only mode. You can validate a single definition, a set of definitions, or all the definitions stored in a warehouse module.

To validate a set of definitions:

1. Select an object or a group of objects from the Warehouse Module Editor.
2. From the **Module** 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.

Figure 9–13 Validation Results Dialog



The dialog lists summary information for each object selected for validation. The summary information includes:

- Name of the object selected for the validation
- Date and time of the last validation
- Valid or Invalid status
- Date and time of the last update

Click **View** to display the validation results in a text editor. This can be useful if you want to save these results to a file.

Click **Refresh** to refresh your validation results. This can be useful if you are running validations in more that one window.

The validation messages lists individual messages for the highlighted object:

- Name of the object
- Validation Code:

  Error: Displays if the script causes code generation or deployment failure.

  Warning: Displays if the script may cause failure during generation, deployment, or runtime.

  Success: Displays if the script does not cause any known problems.
- Validation error message

Click **Details** to view additional details on the highlighted object.

Click **Edit** to open an Editor in Warehouse Builder to edit the highlighted object. For example, if you have a table highlighted and you click Edit, the Table Editor opens with the highlighted table displayed.

Note: A Validation Results dialog displays when you validate a definition. To view these results at a later time, select **Validation Messages** from the **View** menu.

Valid and Invalid Definitions

During the validation process, Warehouse Builder assigns a warning or an error code to each malformed expression within an individual definition.

- **Valid:** The validation process runs and no error codes are generated.
- **Invalid:** The validation process runs and at least one error code is generated. The definition can also include one or more warning codes.

Generating Scripts

During this phase, Warehouse Builder validates and generates the scripts required to create and populate the instance.

Warehouse Builder generates the following types of scripts for a target warehouse:

- DDL scripts to create or drop database objects.
- SQL*Loader control files to extract and transport data from file sources.
- Tcl scripts to schedule and manage jobs registered with Enterprise Manager.

When you generate objects, 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 deploying the entire warehouse again.

To generate scripts for a set of objects in a warehouse module:

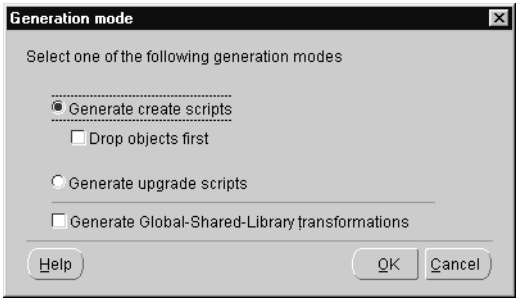
1. Open the Warehouse Module Editor and select a set of warehouse objects.

   You can select an individual object, a set of objects, or all the objects in the module.

2. From the **Module** menu, select **Generate**.

   The Generation Mode dialog displays.

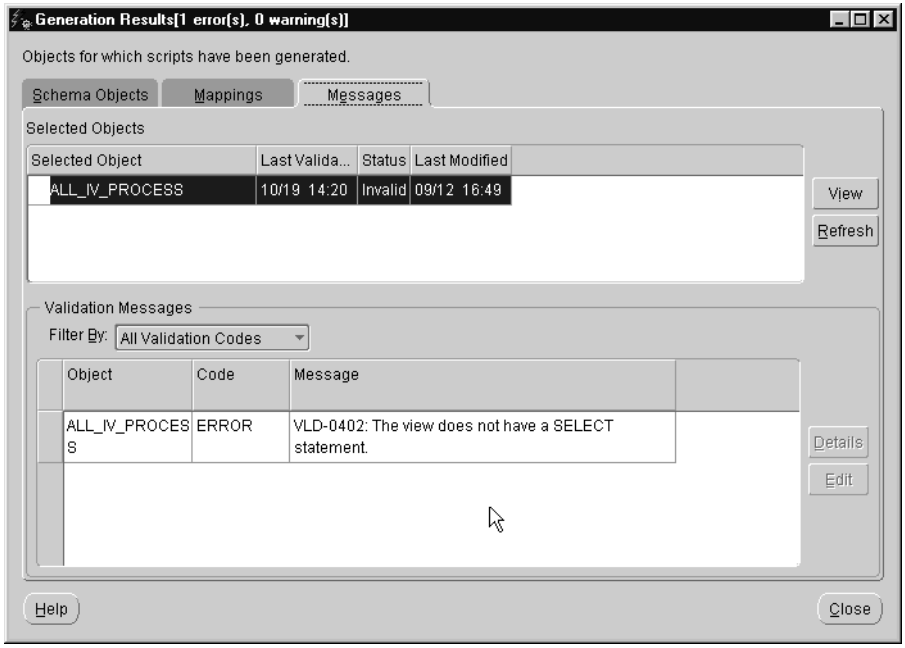
Figure 9–14 Generation Mode Dialog



3. Select the type of script to generate and click **OK.**

   The scripts are generated and the Generation Results dialog displays.

Figure 9–15 Generation Results Dialog



Note: A Generation Results dialog 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.

Verifying the Generation Results

The following sections summarize the scripts generated for each object in the warehouse module.

Database Links

When a data warehouse pulls data from its database sources using database links, the following occurs:

1. Warehouse Builder generates a script for a database link to the source database.
2. The database link is deployed in the target schema.

3. The PL/SQL scripts that pull data from a source database reference the source tables using the deployed synonyms.

For example, Warehouse Builder generates the following script to create a database link to a source.

```
ALTER SESSION ENABLE PARALLEL DDL;
CREATE DATABASE LINK SOURCE
 CONNECT TO source IDENTIFIED BY manager
 USING 'source';
```

This database link is deployed to the physical instance by executing the script.

Dimensions

Warehouse Builder can generate multiple DDL scripts for each dimension depending on whether indexes are configured. The scripts can include a DDL script for:

- The underlying table and its configured partitions.
- The hierarchies and levels defined on the table.
- The indexes and index partitions configured for the table.

After generating the DDL, the Generation Results dialog displays.

The Generation Results dialog contains a list of scripts that create the following objects:

- Table
- Dimension
- Index

You can view any of the generated scripts by selecting the dimension name and clicking **View.** The example below shows the code generated to add hierarchies onto an existing table:

```
ALTER SESSION ENABLE PARALLEL DDL;
 CREATE  FORCE  DIMENSION customers_DIM
LEVEL customer IS  CUSTOMERS.cs_cust_WH
LEVEL account IS  CUSTOMERS.at_account_ID
LEVEL segment IS  CUSTOMERS.sg_segment_ID
HIERARCHY customer_sums (
customer CHILD OF account CHILD OF segment )
ATTRIBUTE customer DETERMINES (cs_loctype,cs_ship_to_num)
ATTRIBUTE account DETERMINES (at_account_desc)
```

```
ATTRIBUTE segment DETERMINES (sg_segment_desc);
```

Views and Materialized Views

Warehouse Builder can generate multiple DDL scripts for each materialized view depending on whether indexes are configured for the view. The scripts can include DDL scripts for:

- The materialized view and its configured partitions.
- The foreign key reference constraints.
- The indexes and index partitions configured for the table.
- The partitions configured for the table.

After generating the DDL, the Generation Results dialog displays. You can view the generated code and deploy the materialized view from this dialog.

If the view you are generating is part of a mapping, Warehouse Builder does not generate a DDL script for the view. Warehouse Builder generates a regular package assuming the view can be updated.

Check to ensure the view can be updated before including it in the map. Views and materialized views are generated in the same way as tables.

Note: When you generate scripts for a view:

- Warehouse Builder does not generate code for the view if a query script in its definition has an error. Warehouse Builder does not check the query script for errors when you define the view.
- If a view references dimensions, then you must deploy the dimensions before you deploy the view.

Mappings

Warehouse Builder generates multiple scripts to implement each PL/SQL mapping:

- PL/SQL scripts create the PL/SQL packages that pull data from database sources, perform transformation operations, and load data into the physical instance of the data warehouse.
- DDL scripts create database links.

- Tcl scripts that are registered with Oracle Enterprise Manager and Oracle Workflow to support job scheduling as well as to capture and log audit and error information.

The Generation Results dialog includes a Run button. Click **Run** to load data into the target warehouse. See Chapter 10, "Managing Loads and Updates" for information on how to schedule and monitor database loading using Oracle Enterprise Manager and Oracle Workflow. See Chapter 11, "Administration" for information on analyzing audit and error messages using the Runtime Audit Viewer.

Sequences

Warehouse Builder generates a single DDL script for each sequence defined by the warehouse module. Each script creates a sequence object in the physical instance.

Tables

Warehouse Builder generates multiple DDL scripts for each table, including DDL scripts for:

- The table and its configured partitions.
- Constraints on the table.
- The indexes and index partitions configured for the table.
- The partitions configured for the table.

After generating the DDL, the Generation Results dialog displays. You can view the generated code and deploy the dimensions from this dialog.

Note: Fact tables reference dimensions using foreign key reference columns. You must deploy the referenced dimensions to the physical instance before you deploy the fact table.

Post-Generation Tasks

After the scripts have been generated, the Generated Scripts dialog displays with two tabs:

- Schema Objects tab
- Transforms tab

Schema Objects Tab

The Schema Objects tab displays objects that you can deploy to create your warehouse.

Select from these actions:

- **View Code:** View and edit the code generated by Warehouse Builder. Select the object name and then click **View Code.** The generated code displays in a code editor. You can view or edit the code. If you edit the code, Warehouse Builder prompts you to save your changes when you close the dialog.
- **Deploy:** Deploy the selected objects to your target data warehouse. See the description under the Transforms tab for more information.
- **Save as File:** Save the code to a file in order to use another tool to deploy. See the description under the Transforms tab for more information.

Transforms Tab

The Transforms tab displays mappings that you can deploy to create your warehouse. Select an object and choose from these actions:

- **View Code:** View and edit the code generated by Warehouse Builder. Select the object name and then click **View Code.** The generated code displays in a code editor. You can view or edit the code. If you edit the code, Warehouse Builder prompts you to save your changes when you close the dialog.
- **Deploy:** Deploy the selected object to a physical instance of the warehouse. This is when your data warehouse is physically created. You must select all the necessary generated scripts and then click **Deploy.** You must select the runtime database you are deploying to each time you deploy. You can configure once and deploy to many instances.

  You can select which scripts to deploy to your warehouse. You deploy all scripts when you create the warehouse. If you decide to create additional objects, such as indexes, at a later time, you can generate and deploy those scripts without re-deploying the entire warehouse.
- **Save as File:** Save the code to a file in order to use another tool to deploy. Choose this option when you want to deploy a script that creates an object to a subdirectory of the Top Directory. The Top Directory is an operating system directory that is configured for the warehouse module.

Warehouse Builder writes the generated script to a subdirectory that corresponds with the script language as defined in the Generation Results or Generated Scripts dialog. The script is written to the file:

\Top Directory\ddl\database_link_name.ddl

For example, the generated scripts for the Customers dimension are written to the DDL subdirectory in the following form:

e:\target\ddl\Customers.ddl

e:\target\ddl\Customers_DIM.ddl

Where the Top Directory was configured as e:\target.

- **Run:** Run a deployed PL/SQL package that implements a mapping. Use this to test PL/SQL load packages with small data samples. Schedule initial loads and periodic refreshes of a physical instance as jobs in Oracle Enterprise Manager.

  Note: This function runs the last deployed package that is currently in the database. If you have made changes, you must first re-deploy before running the scripts.

- **OEM Register:** Register the selected Tcl scripts with Oracle Enterprise Manager. Warehouse Builder generates one Tcl scripts for each mapping. Oracle Enterprise Manager and Oracle Workflow use the Tcl script to initiate, execute, and terminate the capture of audit statistics and error messages. If you plan to schedule jobs using Oracle Enterprise Manager or Oracle Workflow, you must register the Tcl script with Oracle Enterprise Manager.

Notes:

- If you modify the configuration parameters, you must re-generate the scripts, re-deploy all of them, and then re-register them with Oracle Enterprise Manager.
- If you attempt to register a job with Enterprise Manager that already exists in its Job Library, an error displays. You must remove the job from the Job Library before you can register a new version.

Deploying Scripts

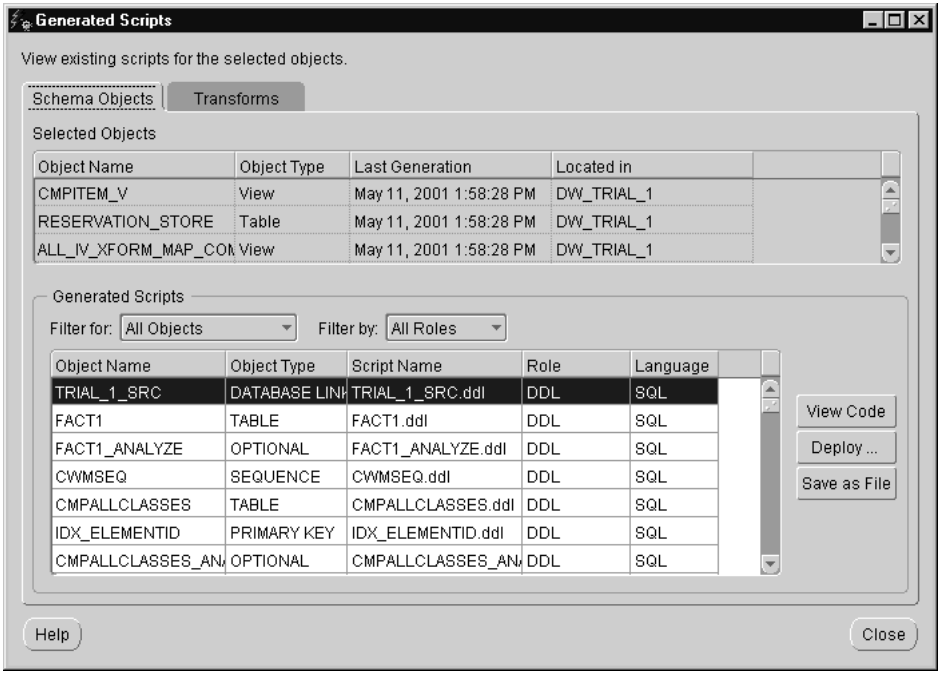
To create and populate the physical instance of the data warehouse, you run the generated scripts to deploy objects in the target schema, deploy the scripts to an operating system directory, register the scripts with Oracle Enterprise Manager and Oracle Workflow, and then schedule jobs to initially load the warehouse.

To deploy generated job scripts:

1. Select a module and open the Warehouse Module Editor.
2. From the **View** menu, select **Generated Scripts.**

   The Generated Scripts dialog displays with the Schema Objects tab on top.

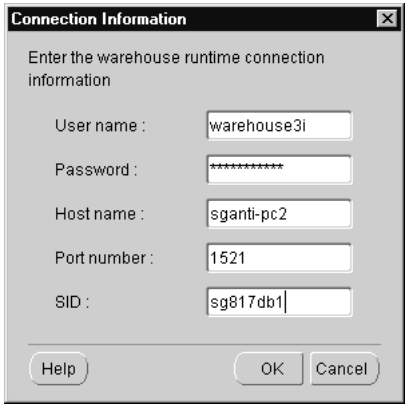
Figure 9–16 Generated Scripts Dialog



3. Select the tab that contains the object or mapping you want to deploy and click **Deploy.**

   The Connection Information dialog displays.

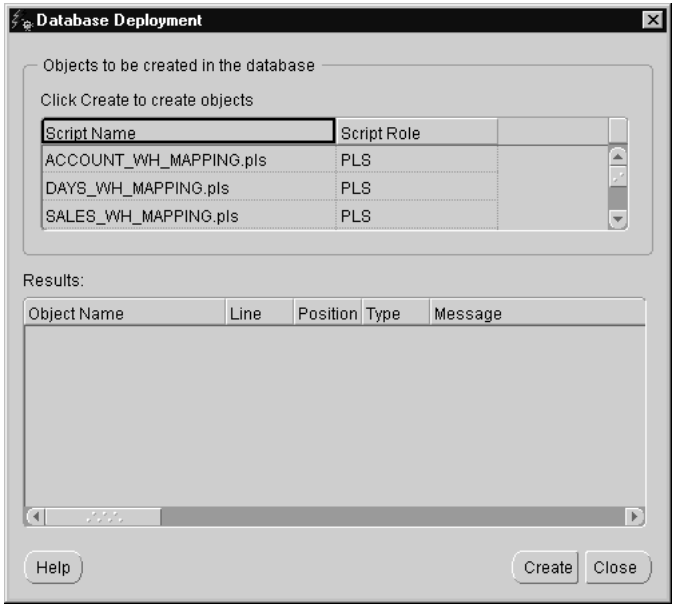
Figure 9–17 Connection Information Dialog



4. Specify the runtime connection information and click **OK.**

The Database Deployment dialog displays. Use this to verify the objects you are deploying.

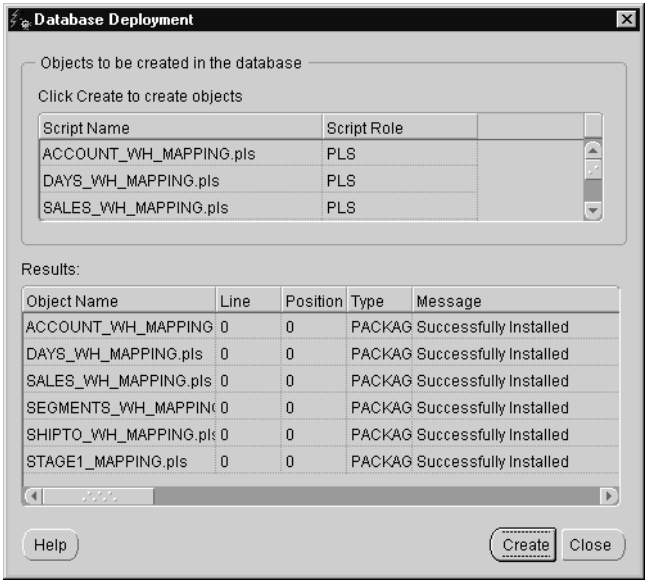
Figure 9–18 Database Deployment Dialog



5. Click **Create** to continue with the deployment.

   The deployment process writes the scripts to the file system configured as Top Directory in the warehouse module. Once the process is complete, the Database Deployment dialog displays again with the results of the deployment displayed in the lower-half of the dialog.

Figure 9–19 Database Deployment Dialog Showing Results



6. Click **Close.**

   The scripts have now been deployed to the database.

Note: When you schedule and execute jobs using Oracle Enterprise Manager or Oracle Workflow, you must deploy the following Tcl scripts generated by Warehouse Builder to the Top Directory:

- SQL*Loader mapping
- PL/SQL mapping
- External OS Command mapping
- Pure Extract mapping
- Pure Integrate mapping

Deploying SAP Definitions

After you generate the scripts for SAP mappings (choose **Module** then **Generate**), Warehouse Builder opens the Generation Results dialog. Use this dialog to deploy the ABAP code to the SAP target system.

Choose from these actions:

View Code: Views the generated code in a code editor dialog. You can edit, print, or save the file through this editor.

Save as File: Displays the File System Deployment dialog. Click **Save** to save the generated scripts to a file system.

Run: Displays the Connection Information dialog. Enter the following information to connect to the runtime library where you want to deploy the ABAP code.

- **User name:** User name to connect to your target schema.
- **Password:** Password for your target schema user.
- **Host name:** Name of the computer where your target schema resides.
- **Port Number:** Port number of the host computer.
- **SID:** Unique database identifier of the target schema.

The Connection Information (SAP Runtime) dialog for SAP targets displays the system settings for the target. Enter the following information to connect to the SAP system where you want to deploy the ABAP code.

- **Application Server:** Name of the SAP source application server.

- **System Number:** SAP system number for SAP GUI logon.
- **Client:** SAP client number.
- **SAP User Name:** SAP GUI user name.
- **SAP Password:** SAP GUI password.
- **FTP User Name:** SAP user (OS) account on the SAP application server that enables FTP access to the data file generated by ABAP code.
- **FTP Password:** SAP password for the user (OS) account on the SAP application server that enables FTP access to the data file generated by ABAP code.

Deploying ABAP Scripts for Non-Transparent Tables

Once you have entered the complete connection information to your SAP target system, the Run the Objects dialog displays. When you click **Run,** Warehouse Builder automatically performs the following procedure:

1. Uploads the generated ABAP code.
2. Executes the ABAP code on the SAP system.
3. Uses FTP to fetch data to the Warehouse Builder staging area.
4. Uses SQL*Loader to upload data into your warehouse tables.

Deploying ABAP Scripts Manually

You can execute this procedure manually. When you generate an ABAP script for an SAP mapping, Warehouse Builder creates an ABAP program that you can run from the SAP GUI. Running this program creates two files for the SAP mapping:

- A datafile of extracted data
- A SQL*Loader control file based on the target table

Copy the data files and control files for all of your SAP mappings to your Warehouse Builder staging area, and then run the SQL*Loader control files to process the data files into your data warehouse.

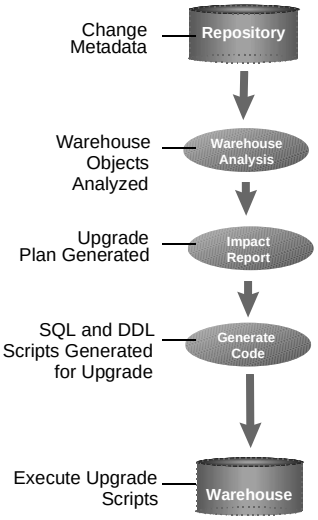
Deploying PL/SQL Scripts for Transparent Tables

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.

Upgrading the Warehouse

You can upgrade a warehouse physical instance after the initial load. You can drop, reconfigure, rename, and upgrade the objects in the instance to reflect changes in your environment. Warehouse Builder enables you to propagate incremental changes from your logical warehouse design to your physical instance, without having to drop objects or lose existing data. For example, you have tables containing data in your physical instance. If you modify the definition of these tables within Warehouse Builder by adding an index, changing a constraint, or renaming a column, you can directly reconcile these changes with the tables in your physical instance with a warehouse upgrade. Your changes are applied in a manner that preserves the existing rows of data within the tables in the physical instance.

Figure 9–20 Warehouse Physical Instance Upgrade



Warehouse Builder analyzes the existing contents of the warehouse and synchronizes them with changes in the Warehouse Builder repository. Next, Warehouse Builder provides an Impact Report detailing the plan for the upgrade and generates necessary SQL and DDL scripts to manage this transition. Before executing the scripts, review the Impact Report to decide whether to perform the upgrade.

You can create, add, update, and rename the following Warehouse objects:

Dimension Table	Materialized View	Constraint (UK, FK, and check constraint)
Dimension Object	Materialized View log	Indexes
Hierarchy	Table	Database link
Fact Table	View	

Note: An upgrade does not work if you deploy a table with Float data type. Use the Number data type to achieve the same results.

A SQL summary of the changes is provided by Warehouse Builder. Warehouse Builder then generates a Tcl script to deploy the changes.

Generating and Executing Upgrade Scripts

To generate and execute upgrade scripts:

1. Open the Warehouse Module Editor and select the set of definitions you want to upgrade.

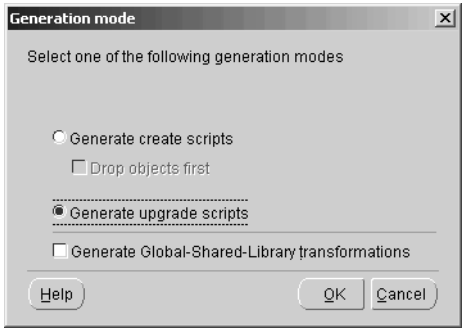
   You can select an individual object, a set of objects, or all the objects in the module.

2. From the **Module** menu, select **Generate**.

   The Generation Mode dialog displays.

3. Select **Generate upgrade scripts.**

Figure 9–21 Generation Mode

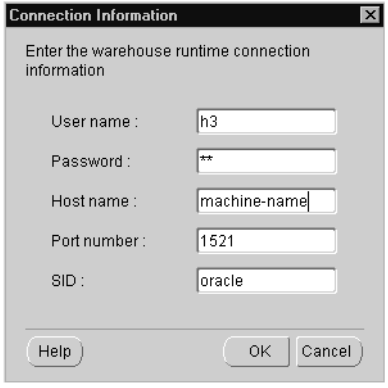


4. Click **OK.**

   The Connection Information dialog displays.

5. Enter the user name, password, machine name, port number, and SID for the target runtime schema you want to upgrade.

Figure 9–22 Runtime Database Connection Dialog



6. Click **OK.**

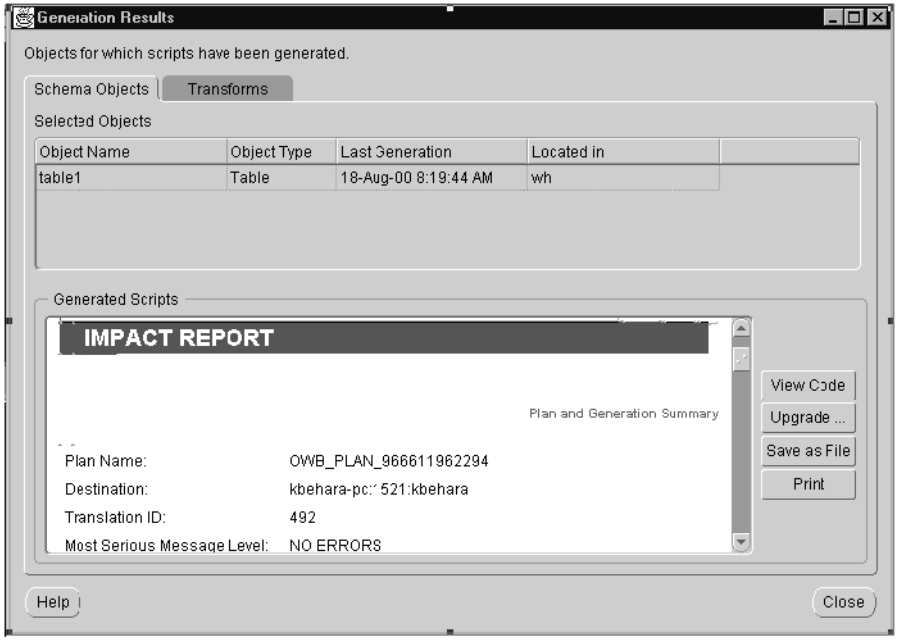
   The Generation Progress panel displays indicating the progress of the script generation for the selected objects.

   After the process is complete, the Generation Results dialog displays an Impact Report listing all objects to be upgraded using the newly generated scripts. For

more information on the options available on this dialog, "Post-Generation Tasks" on page 9-30.

Note: There is only one Impact Report generated for all the objects selected for upgrade.

Figure 9–23 Generation Results



If there are no errors and your schema objects can be readily upgraded from the information supplied, the Upgrade button is activated. If the button is grayed-out, look for error messages within the Impact Report.

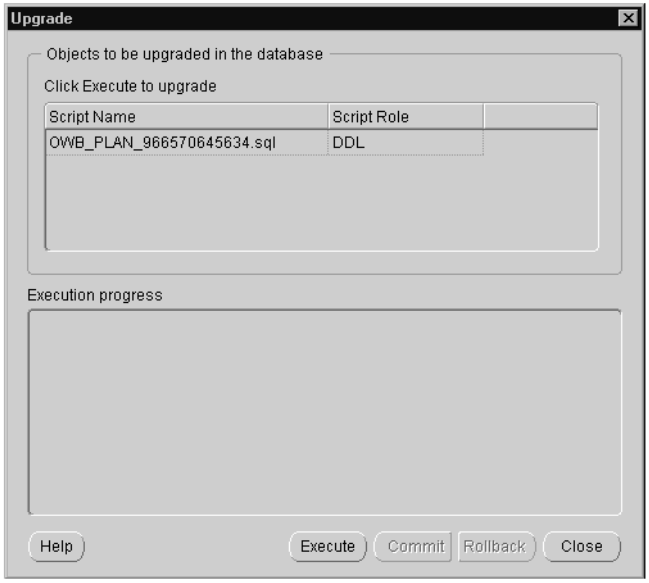
7. Click **Upgrade.**

   The Upgrade dialog displays the script that upgrades the objects that were selected.

8. Click **Execute** to run the script.

   While upgrading, the Execution progress area of the dialog reports the upgrade progress.

Figure 9–24 Upgrade Dialog



9. When the script completes, you can either click **Commit** to keep the changes you have made, or **Rollback** to undo the changes.

   The physical instance in your target warehouse is upgraded.

10. Click **Close.**

Note: Warehouse Builder drops and creates dimensions when upgrading a dimension. The drop and create statements appear along with the summary scripts.

These statements are executed only after you commit and close the Upgrade dialog.

The data warehouse is now ready to be loaded with data that matches the structure of the upgraded objects in your warehouse.

10

Managing Loads and Updates

This chapter describes how to load and refresh a data warehouse using scheduling and dependency management tools. After you generate and deploy scripts, register them as jobs with Oracle Enterprise Manager or another scheduling tool. You can use a dependency management tool such as Oracle Workflow to run multiple job processes with dependencies. Schedule these Workflow processes to load or update the warehouse. The Warehouse Builder Workflow Queue Listener monitors the processes and ensures that dependencies are handled in the correct order. After the processes have completed, view the results using the Warehouse Builder Runtime Audit Viewer.

For information about Oracle Workflow, see the *Oracle Workflow Guide.* For more information about Oracle Enterprise Manager, see the *Oracle Enterprise Manager Administrator's Guide.*

This chapter includes the following topics:

- Registering Tcl Scripts
- Creating an Oracle Workflow
- Scheduling a Workflow
- Viewing the Results

Registering Tcl Scripts

After the target warehouse module is deployed, the warehouse is created and the Tcl or mapping scripts are stored there. In order to run the Tcl scripts and load the warehouse, first register the scripts with Enterprise Manager or another scheduling tool.

To schedule jobs with Enterprise Manager, the following steps must be completed:

- Create a Windows NT user name.
- Configure a set of Preferred Credentials for Enterprise Manager.
- Start services for Oracle Agents and Enterprise Manager on the machine that hosts the target schema.

For more information, see the *Oracle9i Warehouse Builder Installation Guide.*

To register deployed scripts with Enterprise Manager:

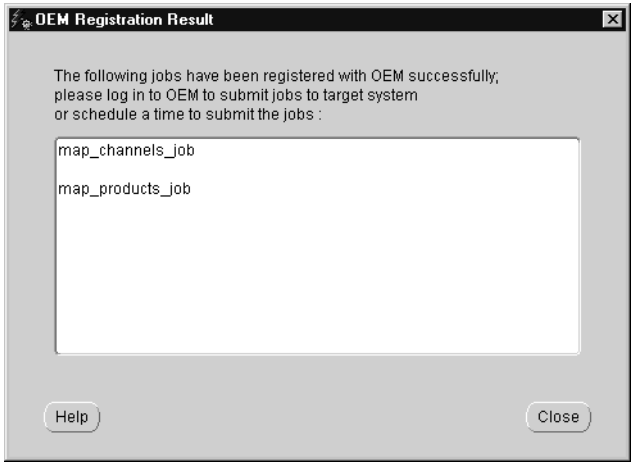
1. From the Generated Scripts dialog, select the deployed scripts and click **Save as File.** Use the Control Key to select multiple scripts.

   The scripts must be saved before you register them.

2. Select the Tcl script you want to register with Enterprise Manager and click **OEM Register**. You can only register Tcl scripts with Enterprise Manager.

   Warehouse Builder connects with Enterprise Manager, registers the load scripts, and displays a confirmation list.

Figure 10–1 OEM Registration Results Dialog



After the scripts have been registered with Enterprise Manager, you can either:

- Schedule the jobs from Enterprise Manager.
- Deploy the scripts to Workflow and create multiple job processes with dependencies.

These processes can then also be scheduled with Enterprise Manager. Workflow processes are effective because they contain all the job dependencies and ensure that the warehouse is loaded and refreshed in the correct sequence.

When you load data or update existing data into a data warehouse, you must run the scripts in strict sequence to ensure that all foreign key references are satisfied. The referenced tables must be loaded before the tables making the reference.

For example, dimension tables must be loaded before the related fact table. A materialized view cannot be refreshed until the related fact table and referenced dimensions have been loaded.

Note: If you modify the configuration of a warehouse module or mapping definition after the scripts have been deployed to the Generation Directories or registered with Enterprise Manager, you must:

- Re-generate the scripts.
- Re-deploy the scripts to the Generation Directories.
- Re-register the scripts with Enterprise Manager. You cannot register a job with Enterprise Manager if that job already exists in the Enterprise Manager Job Library. To update an existing job, remove the job from the Job Library first.

Creating an Oracle Workflow

You can create multiple job processes by defining a Workflow process in Oracle Workflow. When you schedule the processes, the Workflow server ensures that the jobs run in the proper sequence. If an exception occurs, the Workflow server terminates the process.

To define a Workflow process, the following steps must be completed:

- Configure, validate, and generate the mappings.
- Deploy the generated mappings to the Generation Target Directories and the PL/SQL packages to the target database.
- Register the generated mappings with Enterprise Manager.
- Define the Workflow Queue Listener directory and host for the warehouse module. These configuration parameters on the warehouse module specify where the Workflow Queue Listener executes.
- Deploy the mappings to the Workflow server using the Workflow Deployment Wizard.

  If you plan to schedule the workflow process with Enterprise Manager, deploy the process to the Enterprise Manager Job Library.
- Define the workflow process using Workflow Builder.
- Schedule the workflow process using Workflow Monitor or Enterprise Manager.

Deploying Scripts to the Workflow Server

After you register the Tcl scripts as jobs with Enterprise Manager, you can deploy the scripts to the Workflow server using the Workflow Deployment Wizard.

To deploy scripts to the Workflow server:

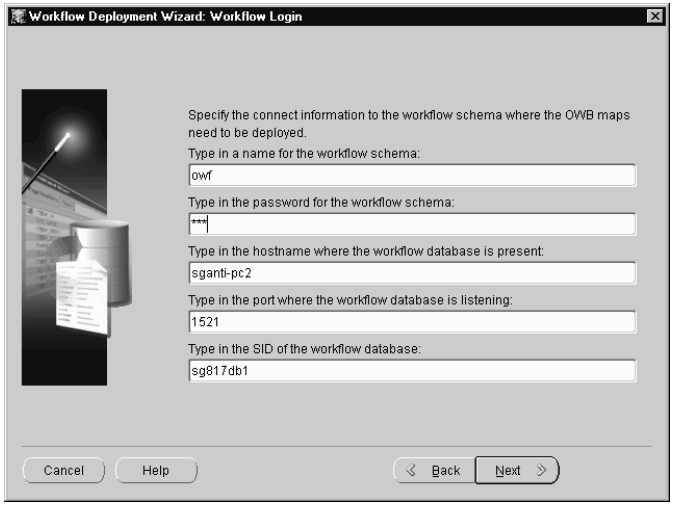
1. From the **Tools** menu, select **Wizards**, and then **Workflow Deployment Wizard** or expand the wizard drawer and click the Workflow Deployment Wizard icon.

   The Workflow Deployment Wizard Welcome page displays.

2. Click **Next.**

   The Workflow Login page displays.

Figure 10–2 Workflow Deployment Wizard Login Page



3. Specify the following connection information:

   - **User name:** User name for the Workflow schema where the mappings are deployed.
   - **Password:** Password for the Workflow schema where the mappings are deployed.
   - **Host name:** Computer that the Workflow schema is located on.
   - **Port number:** Port number that connects to the Workflow schema.

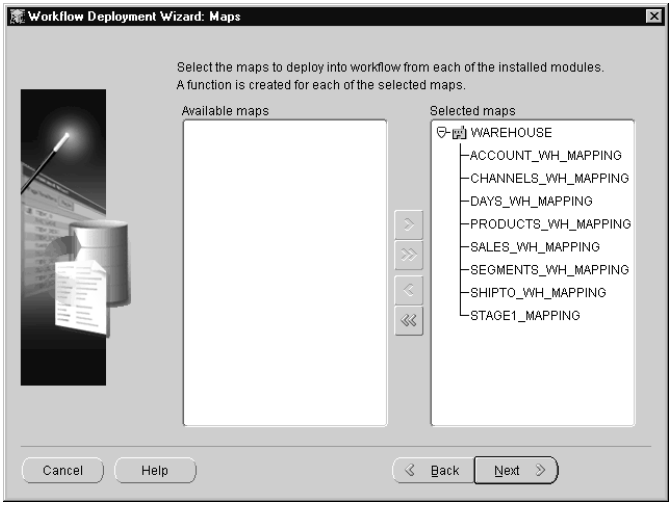
- **SID:** SID for the database instance of the Workflow server.

Warehouse Builder uses this information to establish a session with the Workflow server.

4. Click **Next.**

   The Maps page displays a list of available mappings to be deployed.

Figure 10–3 Workflow Deployment Wizard Maps Page

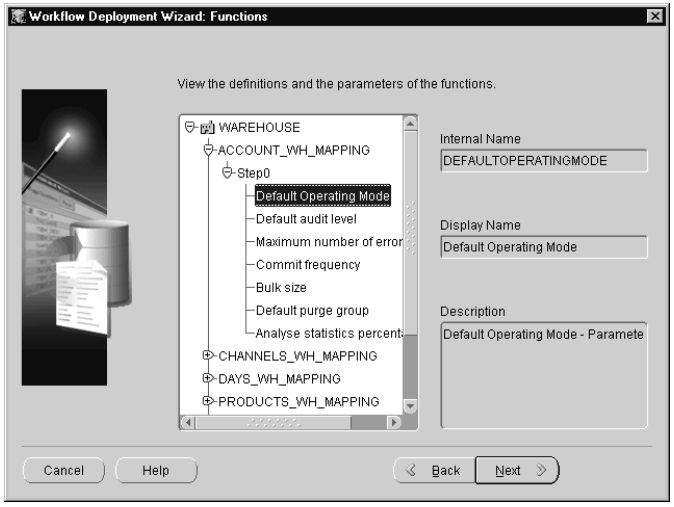


5. Select the maps you want to process. Use the arrow buttons to move the maps to Selected maps.

6. Click **Next.**

   The wizard displays the Functions page, which shows the function names assigned to the mappings.

Figure 10–4 Workflow Deployment Wizard Functions Page



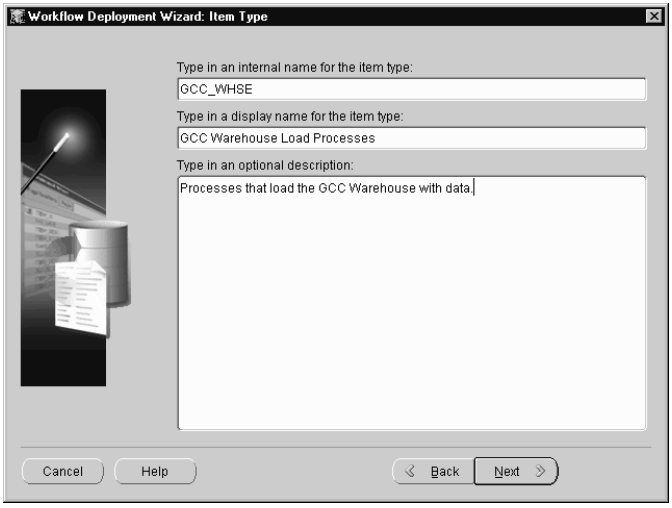
To display the internal function name for a mapping, select the mapping in the navigation tree.

You cannot modify names on this page.

7. Click **Next.**

   The Item Type page displays.

Figure 10–5 Workflow Deployment Wizard Item Type Page



Specify the following:

- An internal name of no more than eight characters.
- A display name of no more than eighty characters. The name can include spaces.
- A description of no more than 240 characters.

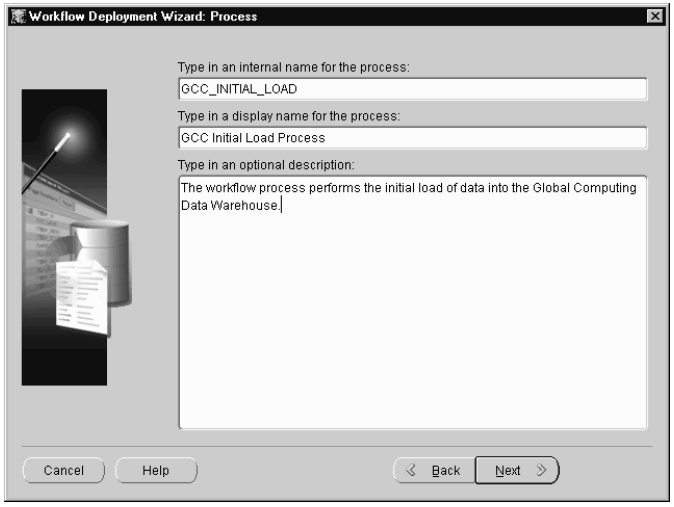
Note: If the item type you are defining already exists with the Workflow server, it is overwritten. This includes any modifications, such as attribute values and process diagram dependencies.

---

8. Click **Next.**

   The Process page displays.

Figure 10–6 Workflow Deployment Wizard Process Page



Specify the following information:

- An internal name of no more than thirty characters.
- A display name of no more than eighty characters. This name can include spaces.
- A description of no more than 240 characters.

9. Click **Next.**

   The Finish page displays. Verify the contents of this page. Use the **Back** button to make changes.

10. Check the **Deploy the workflow process to OEM?** box if you intend to schedule the workflow process using Enterprise Manager.

    The wizard deploys the name of the process to the Enterprise Manager Job Library only if the warehouse module is configured for Enterprise Manager.

11. Click **Finish.**

    The wizard deploys the mappings as a set of Workflow functions defined with the specified item type to the Workflow server.

You can now start the Workflow Builder and define a process to load or update the warehouse.

Defining the Workflow Process

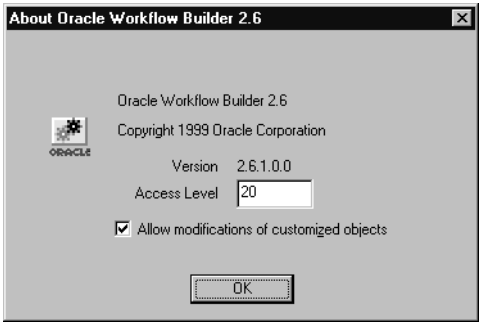
To define a Workflow process:

1. Start **Workflow Builder.**

   The Workflow Builder navigator window displays.

2. From the **Help** menu, select **About Oracle Workflow Builder** to set the client access level to a value less than or equal to twenty and click **OK.**

Figure 10–7 About Oracle Workflow Builder Dialog



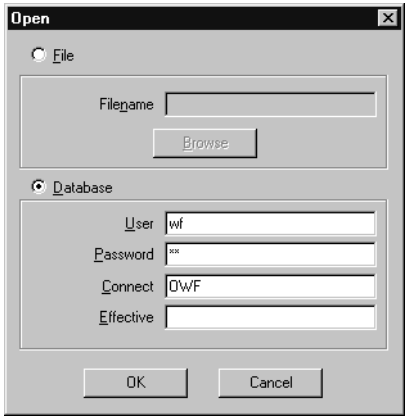
   The Workflow Deployment Wizard assigns each function an access level of twenty during the deployment operation.

   To edit these functions, you must set the access level of the Workflow Builder client to a value less than or equal to twenty.

3. From the **File** menu, select **Open** or click the Open icon.

   The Open dialog displays.

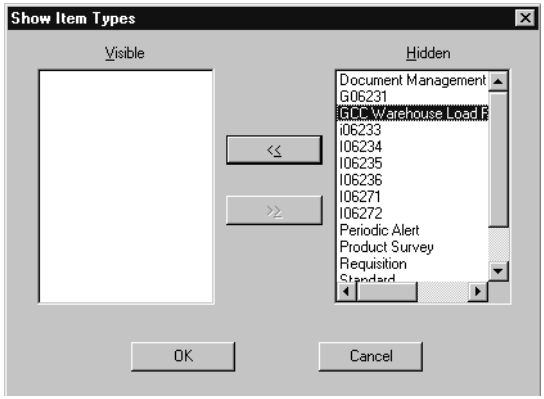
Figure 10–8 Workflow Builder Open Dialog



4. Select the Database radio button and enter the following connection information for the Workflow server:
   - User name and password for the server
   - SQL*Net connect string
5. Click **OK.**

   The Show Item Types window displays a list of item types.

Figure 10–9 Show Item Types Window

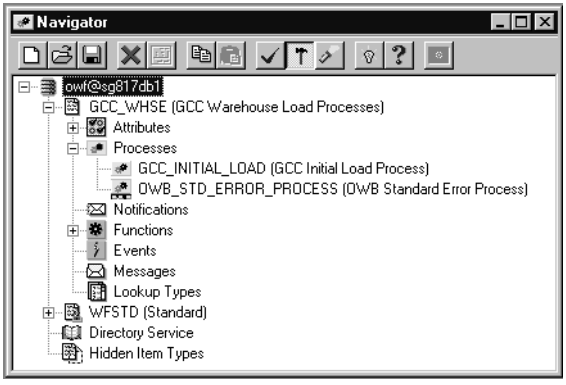


6. Move the item types to the Visible list using the arrow buttons.

7. Click **OK.**

   The Workflow Navigator window displays.

Figure 10–10 Workflow Navigator Window



8. Open your load process.

   Workflow Builder displays tree entries for the mappings deployed from Warehouse Builder.

9. Expand the navigation tree and open the deployed process.

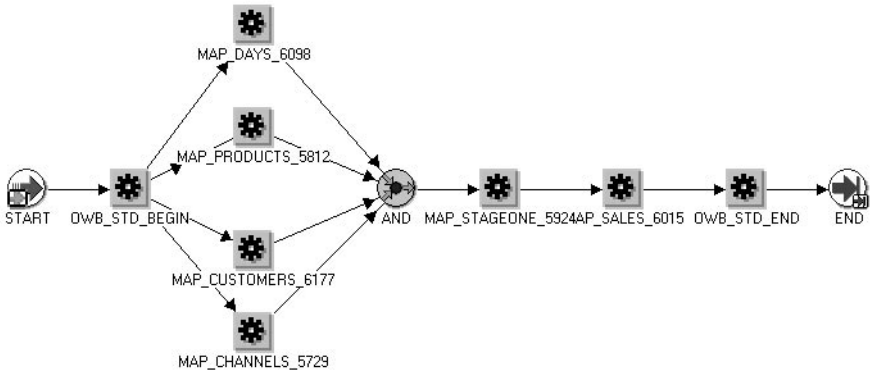
   The process displays in a separate window with the functions overlapped.

10. Design the Workflow process by dragging the functions into the order you want to run them.

11. Define the dependencies between the functions.

    Figure 10–11 shows the Workflow process that sequences the functions to load a data warehouse.

Figure 10–11 Workflow Process Diagram



After the Workflow server executes this job, the functions are processed so that the execution of a function depends on the successful completion of all its predecessors. The dimension tables are loaded in parallel but the remaining two jobs run sequentially.

Scheduling a Workflow

You can schedule a Workflow process using Enterprise Manager if the Workflow process has been deployed to the Enterprise Manager Job Library. Select the Deploy checkbox on the Finish page of the Workflow Wizard to deploy the process.

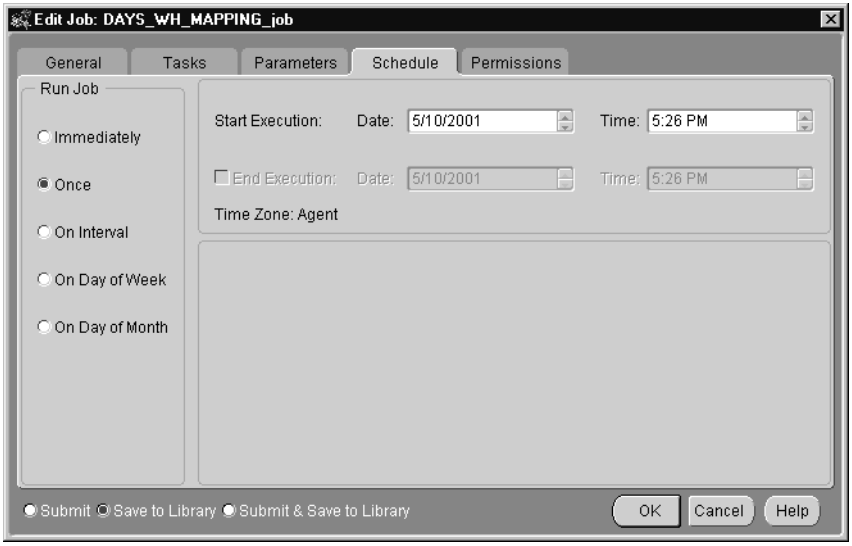
To schedule a Workflow process:

1. Open the Enterprise Manager Job Library, select the Start job of the Workflow process, and click **Edit.**

   The Edit Job window displays.

2. Click the **Schedule** tab.

Figure 10–12 Schedule Tab



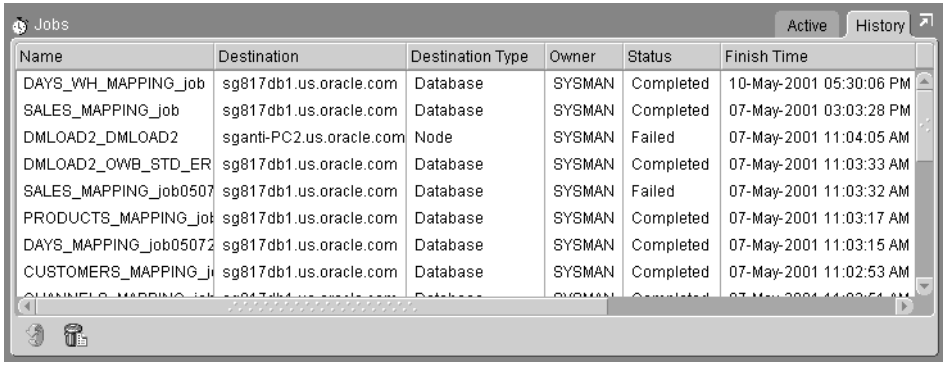
3. Specify when you want this process to run, select the **Save to Library** radio button, and click **OK.**

   If you are scheduling the process to begin immediately, select the **Submit** radio button.

Viewing the Job

You can view jobs in the Enterprise Manager Jobs window. You can track the status of a submitted job from the Active tab. After the job has completed, you can check its status in the History tab.

Figure 10–13 Job History



Right-click the job name on the History tab to display a pop-up list that enables you to view more details regarding the job, remove the job from the history log, or create another job like the selected job.

The Enterprise Manager log contains summary information about execution. For detailed information about the job, use the Warehouse Builder Runtime Audit Viewer. For more information on Enterprise Manager, see the *Oracle Enterprise Manager Administrator's Guide.*

Changing Job Parameters

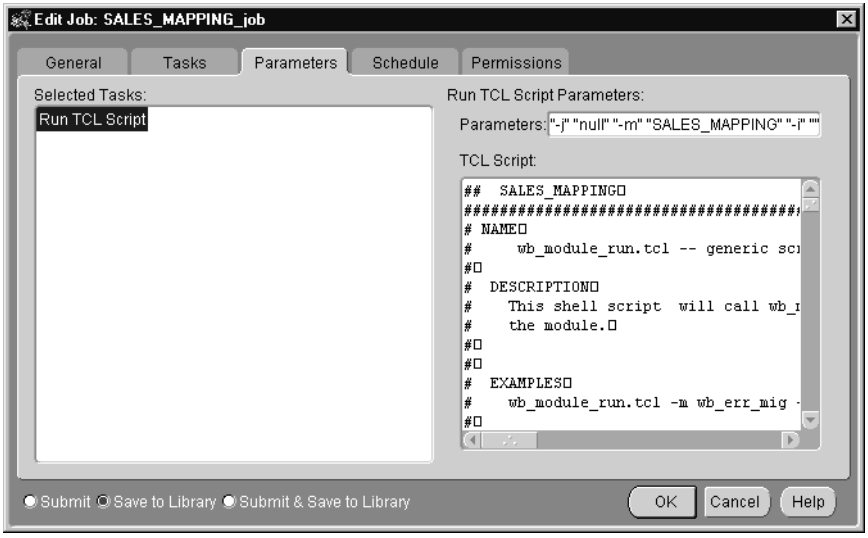
Parameters are set in the mapping configuration in Warehouse Builder. You can modify parameters in the Tcl script before you submit the job.

To modify a parameter value for a job:

1. Open the Enterprise Manager Job Library.
2. Select the job name and click **Edit.**

   The Job Editor window appears.
3. Click the **Parameters** tab.
4. Select the Tcl script and modify the parameter in the text box.
5. Submit the job.

Figure 10–14 Job Editor



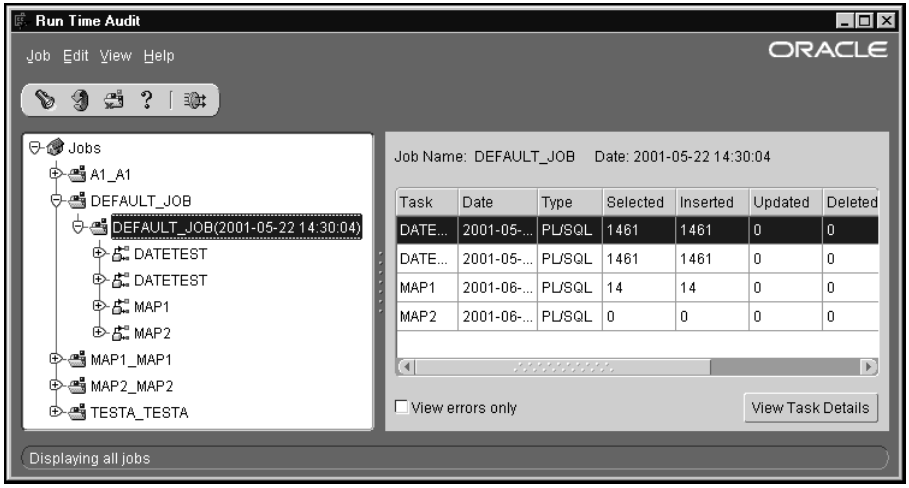
Viewing the Results

The Warehouse Builder Runtime Audit Viewer displays details of a job after it is run. This information can be useful when you are scheduling jobs. The Audit Viewer displays the contents of the Warehouse Builder Runtime Library for a load or refresh job. For example, you can display the number of records read, number of records inserted or updated, and detailed information about individual records when errors occur. This information helps you troubleshoot load errors.

About the Runtime Audit Viewer

The Runtime Audit Viewer window has two panes. The left pane contains a navigation tree with objects grouped by object type. The right pane displays information about the node that is currently selected in the navigation tree.

Figure 10–15 Runtime Audit Viewer Navigator

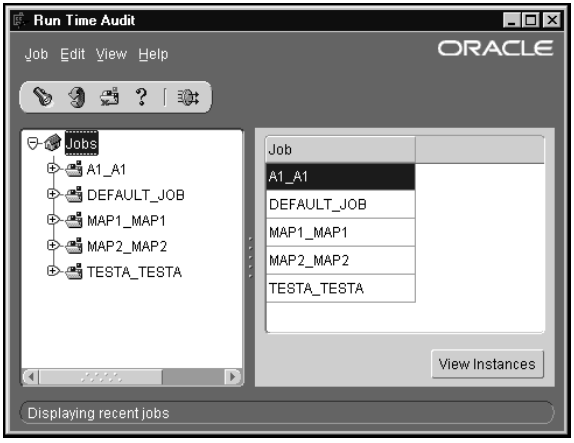


Viewing the Job Audit

The nodes in the navigation tree below the top Jobs node each represent a job. Jobs are listed in alphabetical order. To set up a mapping job, register it as a Workflow Process. The corresponding node appears in the navigation tree after the first run of the Workflow process.

Select the Jobs node at the top of the navigation tree to display the list of jobs. This includes DEFAULT_JOB and any other named jobs that have been run.

Figure 10–16 Job Audit Window



A mapping can be executed from either Warehouse Builder or Enterprise Manager, or it can be used as a component execution unit within a Workflow process.

When a mapping is executed from Warehouse Builder, the audit and error information is stored under the category Default Job. When a Workflow process is run, the audit and error information for the mappings is stored under the name of the Workflow process.

Viewing the Job Instance Audit

Expand a job node to display the job instance nodes in the navigation tree. Each time a job run starts, a new job instance is added. The text shows the name of the job and the time the run started.

Figure 10–17 Job Instance Audit Window

Selecting a job node in the tree or a job entry in the right hand pane and clicking the **View Instances** button displays details about job instances. A job instance represents a specific run of the job.

If the **View errors only** box is checked, a job instance is displayed only if it has errors.

Viewing the Task Audit

A node representing a job instance can be expanded to display Tasks that are part of the run. For example, the execution of a PL/SQL mapping or an SQL*Loader run is represented as a Task. If a Workflow Process consists of several tasks, each running a PL/SQL mapping, a node representing a run of the Workflow Process has a Task node for each of those mappings.

Figure 10–18 Task Audit Window

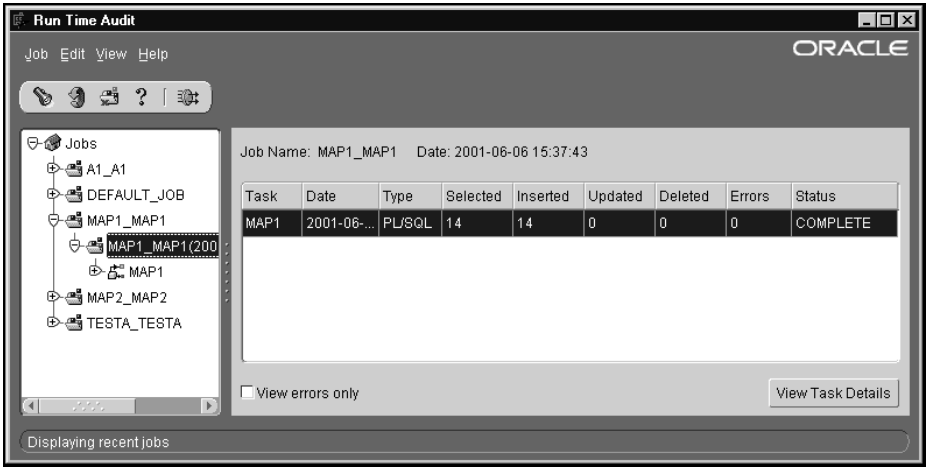


Table 10–1 lists the information displayed for each Task.

Table 10–1 Task Information

Column Name	Description
Task	Name of the task.
Date	Date and time the task was started.
Type	Type of the mapping (for example, PL/SQL).
Selected	Number of rows selected.
Inserted	Number of rows inserted into the target tables.
Updated	Number of rows updated in the target tables.
Deleted	Number of rows deleted in the target tables.
Errors	Number of errors detected.
Status	Status of the task: BEGIN indicates the task has started. COMPLETE indicates the task completed. FAILED indicates the task failed to complete.

Viewing the Task Details Audit

You can expand a task node to display a set of Detailed Mappings, which are also called target entries. In many cases there is only one Detailed Mapping for a task, but different scenarios can cause a task to have more than one Detailed Mapping.

Figure 10–19 Task Details Audit Window

Selecting a task node in the tree, or the Targets node below it, or selecting a task entry in the right-hand pane and clicking **View Task Details** displays information about its Detailed Mappings. A Detailed Mapping entry represents a mapping to a specific target table. A task that affects multiple target tables has a Detailed Mapping entry for each target table. Also, if a PL/SQL mapping is run in set-based fail over mode, and the set-based run detects errors for a specific target table, there are two Detailed Mapping entries for the table. One is for the set-based run and one is for the row-based run.

If the relevant detailed mapping cannot be started when processing a task, there is no Detailed Mapping entry for it. For example, a set-based run cannot be started with certain Loading Types, such as DELETE. In this case, although a set-based fail over run immediately switches to row-based mode, a pure set-based run causes the mapping to be abandoned. The task itself is not marked as COMPLETE, and it has a reduced or empty list of Detailed Mappings.

If the **View errors only** box is checked, only those Detailed Mapping entries with errors are displayed.

For PL/SQL mappings, the values of the statistics reported depend on the mode in which the mapping is run. For example, in:

- Set-based mode, if any errors are found, only the first error detected is logged, and the number of errors is set to 1.
- Row-based modes, multiple errors can be logged.
- Set-based mode, the value for the number of records selected is always the same as the number of records inserted in the target table. If any errors are detected, the transaction is rolled back, and the number of records selected and the number of records inserted are both zero.
- Row-based (target) mode, the value for the number of records selected is derived from a cursor that implements all the stages of a mapping. The value corresponds to the number of rows that are applied to the target table.
- Row-based mode, the cursor used to derive the number of records selected can omit the final stage of the mapping in some circumstances. For example, if the final stage involves a splitter, the number of records selected reflects the records selected as input to the split operation. The number of records can be larger in row-based mode than in set-based mode or row-based (target) mode.

Using the Warehouse Builder Runtime Audit Viewer

The Runtime Audit Viewer can only report on audit information that has been stored during the relevant mapping runs. The default Audit Level for a mapping is defined in the mapping configuration parameters. You can change this value when you configure the mappings. This value can be overridden in the runtime Tcl parameters.

For PL/SQL mappings, the audit levels are:

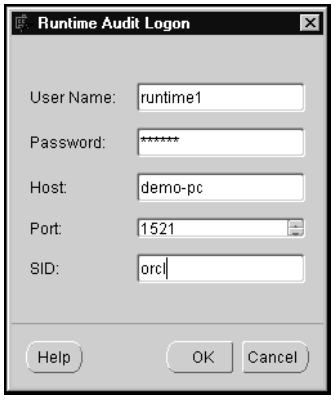
- **None:** No Task, Detailed Mapping, Error, or Error Detail information is available.
- **Statistics:** Task and Detailed Mapping information is available, but Error and Error Details are not.
- **Error Details:** All the information relevant to the Audit Viewer is available. This is the default value.
- **Complete:** This level stores additional information for all rows mapped. This extra information is not used by the Audit Viewer.

To use the Runtime Audit Viewer:

1. Select **Warehouse Builder Runtime Audit Viewer** from the **Start** menu.

   The Runtime Audit Logon dialog displays.

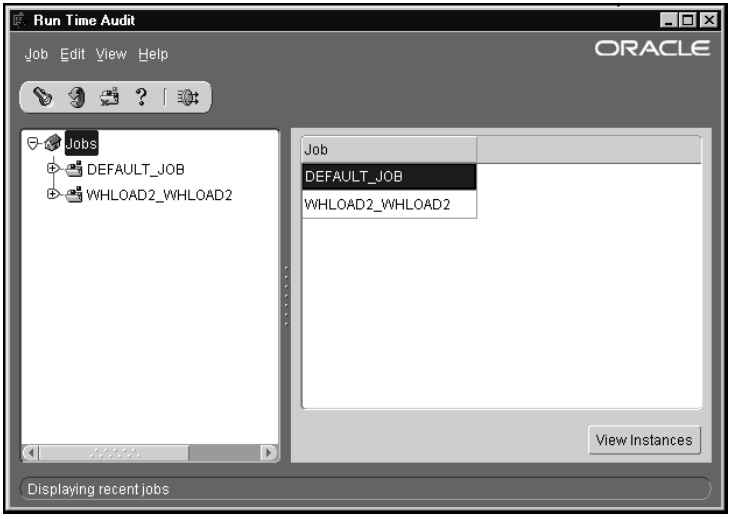
Figure 10–20 Runtime Audit Logon Dialog



2. Specify the runtime connection information and click **OK.**

   The Runtime Audit Viewer displays. The left pane contains a navigation tree. The right pane contains a list of details for the currently selected node.

Figure 10–21 Runtime Audit Viewer



3. Select objects in the left pane to display detailed information in the right pane.

   When the Audit Viewer first opens, all of the objects are rolled up under the Jobs node. As you expand the nodes, you see the following layers of objects:

   - **Jobs:** Lists the jobs that have been run.
   - **Job Instances:** Lists the runtime results of a particular job.
   - **Tasks:** Lists the results of the tasks within the jobs that have run.
   - **Detailed Mappings:** Lists the results of mappings that have been run.

Viewing Specific Results

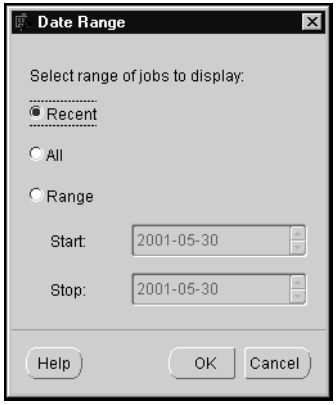
You can specify the results you can view by defining date range, or by performing a search. By default, the most recent job instance is displayed in the tree.

To select a date range:

1. From the **View** menu, select **Date Range.**

   The Date Range dialog displays.

Figure 10–22 Date Range Dialog



2. Choose a range option:
   - **Recent:** The most recent instance of each job is displayed. This is the default value.
   - **All:** All instances of each job are displayed.
   - **Range:** Instances run within a specific date range are displayed.
3. Click **OK.**

To search for objects by name:

1. From the **Edit** menu, select **Find.**

   The Find Tree Objects dialog displays.

Figure 10–23 Find Tree Objects Dialog

2. Choose from the options listed in Table 10–2.

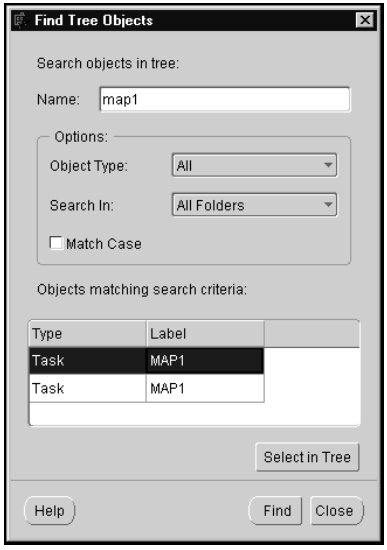
Table 10–2 Find Tree Options

Option	Description
Name	Type the name of the object you are searching for. Names of targets are enclosed in double quotes. You can use the asterisk (*) as a wildcard. For example: D* matches all names beginning with D. D*2 matches all names beginning with D and ending with 2.
Object Type	Search for either one or all object types.
Search In	Specify the scope of the search: Current Folder: searches the currently selected object folder in the navigation tree. All Folders: searches the whole navigation tree.
Match Case	Determine whether to match the case used in the Name field.

3. Click **Find.**

A list of objects matching the search criteria displays. You can click one of these entries and then click **Select in Tree** to select the relevant object in the navigation tree.

Figure 10–24 Find Tree Objects Dialog with Matching Items



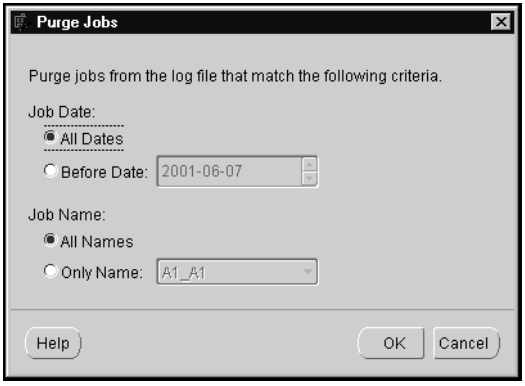
Refreshing the Audit Viewer

You can refresh the data displayed in the Audit Viewer by selecting **Refresh** from the **View** menu. This is useful if you are checking for errors or troubleshooting a job.

Purging Runtime Entries

You can purge jobs from the runtime tables using the Runtime Audit Viewer. To purge runtime entries, select **Purge** from the **Job** menu or the toolbar. The Audit Viewer displays the Purge Jobs dialog. You can purge jobs according to Job Date or Job Name.

Figure 10–25 Purge Jobs Dialog



11

Administration

This chapter covers information relating to the repository and metadata stored within it. When you first install or upgrade Warehouse Builder, you can use the following wizards to set up Warehouse Builder:

- Warehouse Builder Repository Assistant
- Warehouse Builder Runtime Assistant
- Warehouse Builder MDL File Upgrade Utility

You can either run these during the installation or upgrade, or run them later. See the *Oracle9i* Warehouse Builder *Installation Guide* for instructions.

This chapter includes the following topics:

- About the Warehouse Builder Metadata Loader
- About Batch Services
- About the Warehouse Builder Transfer Wizard
- About the Archive/Restore Utility
- Using Discoverer Workbooks for Metadata and Runtime Reporting

About the Warehouse Builder Metadata Loader

Use the Warehouse Builder Metadata Loader (MDL) to export objects from a Warehouse Builder repository to a file and to import objects from a file into Warehouse Builder repository. With this method, you can move metadata into a different repository. The MDL supports the exporting and importing of different types of objects, such as tables, facts, dimensions, materialized views, mappings, transformation categories, transformations, sequences, and information belonging to the objects including columns, constraints, parameters, and named attribute sets. You can export an entire project or a subset of objects from a project. You can execute the export and import utilities using a command line in an MS-DOS window.

Using the MDL With Multiple Users Warehouse Builder enables multiple users to access the same repository at the same time by applying locks to objects that are being modified. Locking affects metadata import. When you import metadata, Warehouse Builder applies locks to all individual objects you are importing. When the import is complete, the changes are committed.

Using the MDL with Large Files Large MDL exports and imports can require a large amount of memory. For large exports and imports, if the default -Xmx setting of 256M in exp.bat, imp.bat, or owbclient.bat is not enough, MDL fails due to lack of memory. Increasing the -Xmx setting in the appropriate bat file can fix this problem.

When importing a large MDL file that results in a large number of updates or adds, Warehouse Builder cannot have the capacity to lock all of the objects affected depending on the Warehouse Builder repository configuration. In this case, Warehouse Builder attempts to switch to single user mode. Also, if more than 300 locks or seventeen percent of the total enqueue resources for the Warehouse Builder repository, whichever is less, are required to import an MDL data file, Warehouse Builder attempts to automatically switch to single user mode. This allows the MDL to avoid performance degradation when using a large number of locks. This also helps limit the potential of running out of enqueue resources. If the MDL switches to single user mode, no other users are able to log on to the repository until after the MDL import completes.

Run large MDL exports and imports on the database server where the Warehouse Builder repository resides in order to get the best performance.

Note: To avoid potential loss of work, perform large imports when usage is at a minimum.

This section describes the components of the Metadata Loader:

- Metadata Export Utility
- Metadata Import Utility
- Metadata Loader Command Line Utility

Metadata Export Utility

The Metadata Export Utility exports objects from a Warehouse Builder repository to an operating system file. The utility can export an entire project, a set of modules, or a set of objects defined by a module. It can also export the global-shared transformation library.

Export File

The export file is a delimited character file which can be read with a text editor. Each record within the file begins with a keyword followed by one or more variable-length fields separated by a pipe (|) or caret (^). The separator character is specified when the file is exported. The pipe (|) is the default separator character.

Example 11–1 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
```

Subsets of Objects You can export all of the objects in a project or a subset of those objects. For example, you can export the entire warehouse project to a file, a few selected modules, or selected objects within a source or warehouse module.

When you export a subset of objects, the utility exports definitions for each object selected and all of that object's ancestors. For example, if you export only the dimension, the export file contains definitions for:

- The dimension

- The dimension's parent (the warehouse module)
- The warehouse module's parent (the warehouse project)

Table 11–1 lists the repository objects that can be exported and imported by the Metadata Loader.

> **Note:** If you are exporting a subset of objects, make sure you export all referenced objects and import them as well. The Metadata Import Utility allows you to import repository objects even if the object's references cannot be satisfied.

Table 11–1 Repository Objects

Projects	Modules	Module Objects
Warehouse Builder Project	Source	Files
		Sequences
		Tables
		Views
	Warehouse	Business Areas
		Dimensions
		Facts
		Mappings
		Materialized Views
		Sequences
		Tables
		Transformation Categories
		Transformations
		Views

Exported Values for Configuration Parameters By default, the Metadata Export Utility exports values for configuration parameters. To override this, run the Metadata Export from the command line and include the statement CONFIGPARAM=N in the Metadata Export Utility's parameter file. For more information, see "Metadata Loader Command Line Utility" on page 11-15.

Exporting Metadata

1. Select the names of the objects you want to export.

   To export a project:

   - Select the project name from the Project tree view.
   - From the **Project** menu, select **Metadata Exports** and then **File.**

   To export a module or set of modules from a project:

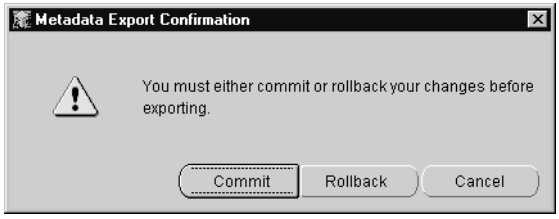
   - Select the names of the module(s) to be exported from the Project tree view. Press the Control button to select multiple modules.
   - From the **Project** menu, select **Metadata Exports** and then **File.**

   To export a subset of objects in a module:

   - Double-click a module in the Project tree view to open the Module Editor.
   - Select the names of the objects or object types to be exported. Press the Control button to select multiple modules.
   - From the **Module** menu, select **Metadata Export.**

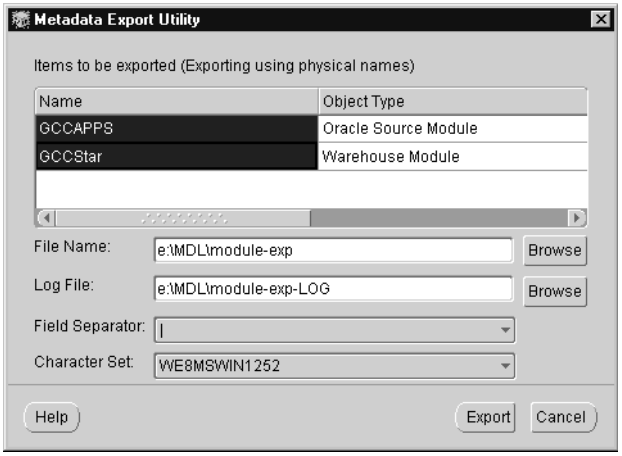
2. If you have made any changes and you want to run the export, the Metadata Export Confirmation dialog displays. Click **Commit** to save any changes or **Rollback** to ignore changes and revert to the previously saved version.

Figure 11–1 Metadata Export Confirmation



The Metadata Export Utility window displays. This window displays the name of the objects you are exporting

Figure 11–2 Metadata Export Utility



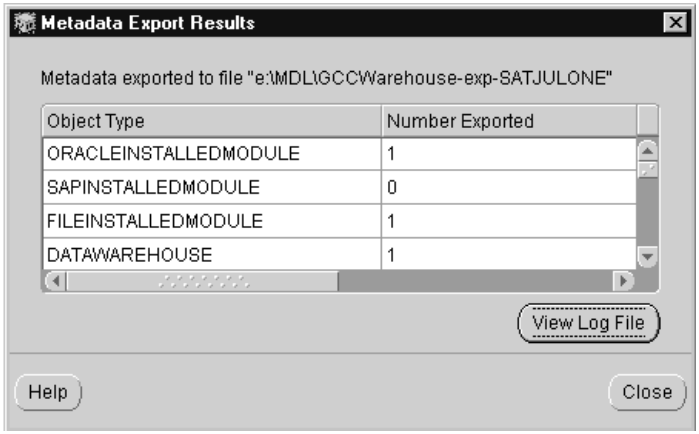
3. Specify the following:

   - **File Name:** Typethe name of the export file to create. Click **Browse** button to locate a directory or file.
   - **Log File:** Enter the name of the log file to create. The MDL writes diagnostic and statistical information to the log file. Click **Browse** to locate a directory or file.
   - **Field Separator:** Select a field separator. The MDL uses the field separator to separate the object and its attributes as shown in Example 11–1. The default is the pipe symbol (|). You can change it to a caret symbol (^) using the drop-down list.
   - **Character Set:** Select the character set to use in the export file. The default character set is defined by the Warehouse Builder client machine. Use the drop-down list to change the output character set.

4. Click **Export.**

   The Metadata Export Progress dialog displays the progress. When the export completes, the Metadata Export Results dialog displays.

Figure 11–3 Metadata Export Results



The Metadata Export Dialog displays the object types and the number of each type that were exported. For a detailed view of the export process, click **View Log File.** This displays the entire log file.

Metadata Import Utility

The Metadata Import Utility imports objects from an export file into a Warehouse Builder repository. The Metadata Import Utility operates in one of four modes: add, replace, add and replace, and add and merge.

Import Searching When you use the Metadata Import Utility, it searches the repository for any repository objects that exist in the repository and in the file you are importing. There are three methods to search for objects: Universal Identifiers, Logical Names, and Physical Names.

- **Universal Identifiers:** Universal identifier called the Universal Object Identifier or UOID are exported for each object to the export file. These identifiers can be used to determine whether an object needs to be created, replaced, or merged during an import operation.
- **Logical and Physical Names:** The physical and logical name of an object are exported to the export file. These names determine whether an object needs be created, replaced, or merged during an import operation.

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 the an object with the corresponding logical name is not found, the

resulting actions are based on the import mode you select. Table 11–2 lists and describes the available import modes.

Table 11–2 Import Mode without Matching Logical Names

Import Mode	Result
Add 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.
Add and Replace Mode	A new object is created.
Add and Merge Mode	A new object is created.

If the Metadata Import Utility finds the logical name, the following actions result based on the import mode. Table 11–3 lists and describes the available import modes.

Table 11–3 Import Mode with Matching Logical Names

Import Mode	Result
Add 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.
Add and Replace Mode	The object is replaced.
Add and Merge Mode	The object is merged.

When importing using the add and replace, and replace mode, the import completely replaces the existing object's children so that the final object is exactly the same as the source object. Any children of a repository object that are not replaced or added are deleted. This occurs regardless of whether a child occurs in a mapping or is a foreign, primary, or unique key column.

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 Warehouse Builder 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. Table 11–4 lists examples of repository objects and children.

> **Note:** A replace operation can lead to malformed constraints and mapping definitions.

Table 11–4 Examples of Repository Objects and Children

Repository Object	Children
Tables, Views, Materialized Views	Columns, Foreign Keys, Unique Keys, Primary Keys, Check Constraints, Named Attributes, Configurations
Dimensions	Levels, Level Attributes, Hierarchies, Level Relationships, Columns, Foreign Keys, Unique Keys, Primary Keys, Check Constraints, Named Attributes, Configurations
Facts	Measures, Columns, Fact Foreign Keys, Segmented Unique Keys, Foreign Keys, Unique Keys, Primary Keys, Check Constraints, Named Attributes, Configurations
Transformations	Parameters, Implementations, Configurations
Files	Records, Fields, Configurations
Business Areas	Business Area Relationships

Importing Metadata

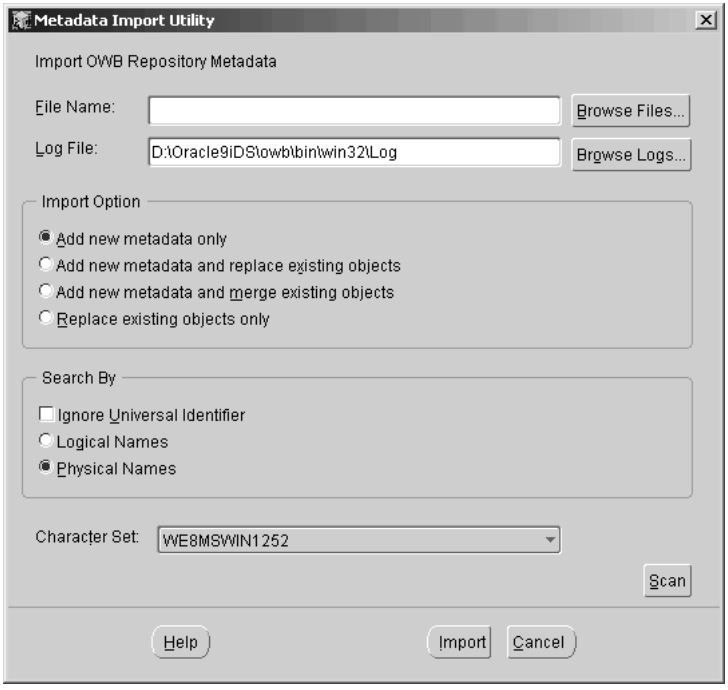
To import objects from an export file:

1. From the **Administration** menu, select **MetaData Import.**

   The MetaData Import Utility window displays.

   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.

Figure 11–4 Metadata Import Utility



2. Specify the following:

   File Name: Type in or browse to the export file you want to import.

   Log File: Create a log file for the import. Warehouse Builder reads and processes the exported metadata and writes status and diagnostic information in the log file.

   Import Options:

   - **Add new metadata only:** Adds new objects to a repository. If you import a file that contains objects that already exist in your repository, they are ignored. The old objects remain unchanged.
   - **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 existing objects in your repository. This reconciles any changes you

have made to your repository with changes or additions that were made elsewhere and then exported to an MDL file.

- **Replace existing objects only:** Replaces existing objects in your repository.

Search by:

- **Ignore Universal Identifier:** Does not use Universal Identifiers to search for objects you are importing. By default, the Universal Identifiers are used to determine if an object already exists or not.

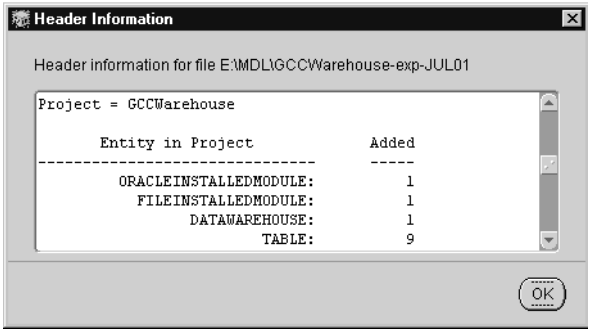
  Note: Warehouse Builder creates Universal Object IDentifiers or UOIDs for each object in the repository. MDL imports that run in merge mode must use UOIDs for the search criteria in order to merge into existing mappings. If the **Ignore Universal Identifier** box is checked, any mappings from the MDL file that already exist in the target repository are skipped. 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.

- **Logical Names:** Searches your repository using the logical names of the objects you are importing to make sure the objects do not already exist.
- **Physical Names:** 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.

3. Click **Scan** to display the exported metadata header information and a summary of the total number of object types contained in the exported metadata file.

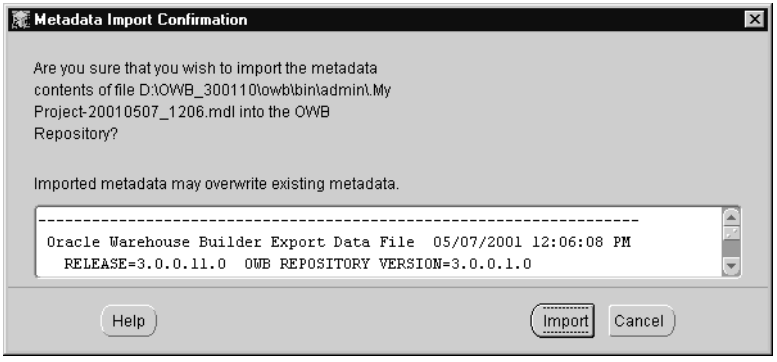
Figure 11–5 Header Information



4. Click **Import**.

   The Metadata Import Confirmation dialog displays only if the exported metadata data information has not been reviewed.

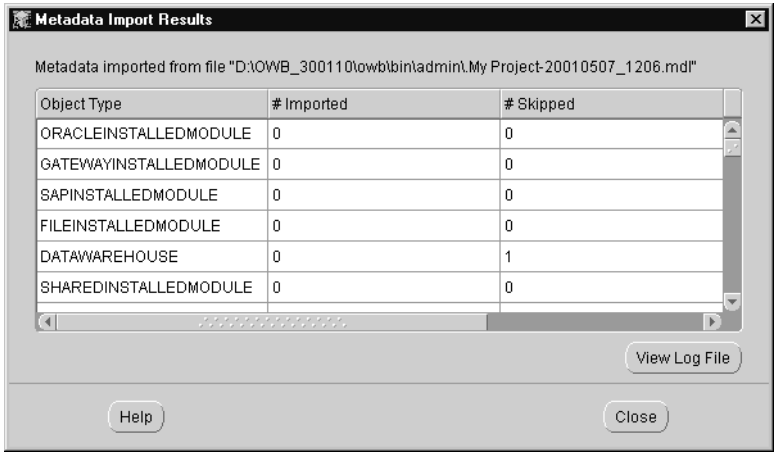
Figure 11–6 Metadata Import Confirmation



5. Click **Import** to continue.

   The Metadata Import Progress panel displays. When the import is complete, the Metadata Import Results dialog displays.

Figure 11–7 Metadata Import Results



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**. This displays the entire log file.

The Log File There are three types of status messages in the log file:

- **Informational:** Provides information about the import or export.
- **Warning:** Cautions you about the import or export of an object.
- **Error:** Indicates that the MDL export or import was aborted and did not complete successfully.

Example 11–2 displays the contents of an import log file. The import statistics display the total number of objects that have been added, replaced, and skipped.

Example 11–2 Sample Import Log File

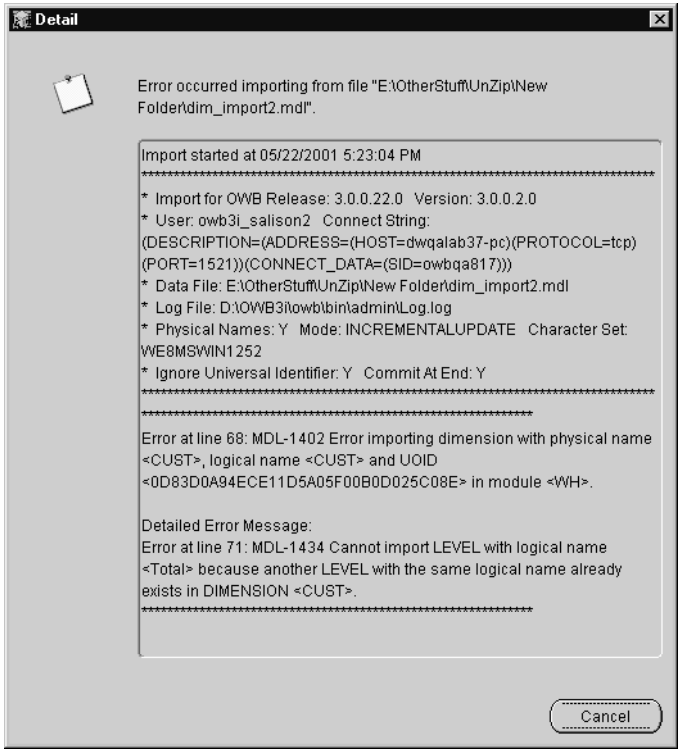
```
Import started at 04/25/2001 4:59:46 PM
*******************************************************************************
*  Import for OWB Release: 3.0.0.0.0   Version: 3.0.0.3.0
*  User: user30_3i   Connect String: epaglina-pc:1521:ora8i
*  Data File: d:\owb3000\sco_dim_time_phy_m_tgt.mdl
*  Log 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
```

Detailed Error Messages If you are running an MDL Import or Export and encounter an error, an error message displays.

You can click **Detail** to display a detailed error log that lists the repository object and the object line that the error occurred in. This is useful for troubleshooting. In the example below, the repository object is the CUST dimension and the object is the TOTAL level.

Figure 11–8 Detailed Error Message



Metadata Loader Command Line Utility

You can execute the Metadata Loader Utilities from an MS-DOS window. These two utilities allow you to disable the export of configuration values, vary the separator character within an export file, and maintain directive files for selected export operations. 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 `$OWBHOME\owb\bin\win32` directory for the Windows platform and `$OWBHOME/owb/bin/solaris` for UNIX platforms. Both the export and import utilities are driven by a set of directives. You can specify the directives in the following ways:

- Provide directives as a response to command-line prompts.

- Create a directives file.
- Provide directives as a response to command-line prompts, and create a directives file.

The following examples describe how to export and import Warehouse Builder definitions using these utilities with a directive file.

Directive Keywords for the Export Utility

The format for a directive is:

Keyword=Value

You can also form a directive 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)

The directives can be written in a simple text file. Table 11–5 summaries the keywords used to form export directives. You can use the comment indicator (#) to document your directive scripts. Put the indication in the first column of a record and follow it with text.

Table 11–5 Keywords for Export Utility Directives

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 password that matches USERNAME.
	HOST	Machine name for Warehouse Builder repository.
	PORT	Port for Warehouse Builder repository.
	SID	SID for Warehouse Builder repository.
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.

Table 11–5 Keywords for Export Utility Directives (Cont.)

Utility Prompt	Keyword	Description
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 contains keyword directives.
	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 use for the export data file.
	MODULES	If wildcard or multi-value format used for MODULE, no other object type keywords can follow. If 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	
	MATERIALIZEDVIEWS	
	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	
	HELP	Use HELP=Y for a complete list.

Table 11–5 Keywords for Export Utility Directives (Cont.)

Utility Prompt	Keyword	Description
	#	Comment line used in a parameter file.

Export a Project

This example describes how to export an entire warehouse project. The operation requires two steps: create the directives file and then execute the Metadata Export Utility.

Create the Directives File The directives file is a simple text file that contains a set of directives for the export utility.

```
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
```

Execute the Export Utility The following command invokes the Metadata Export Utility and specifies the above directive file:

```
w:\owb\bin\win32>exp parfile=e:\MDL\EXP_Directives
Processing ... Export successful.
```

The objects have now been exported to the file and can be imported into a repository using the Metadata Import Utility.

Directive Keywords for the Import Utility

You can direct the import utility to import objects from a file by answering prompts or by creating a file with a set of directives. Table 11–6 summaries the keywords used to form a directive. The format for each directive is Keyword=value.

Table 11–6 Keywords for Import Utility Directives

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.

Table 11–6 Keywords for Import Utility Directives (Cont.)

	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 contains keyword directives.
	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 directive is not included, then the default is CREATE.

Import Selected Modules

This example shows you how to import two modules using the Import Utility. The operation requires two steps: create a directive file and then execute the Metadata Import Utility.

Create the Directives File The directives file is a simple text file that contains a set of directives for the export utility.

```
USERID=GCCWH/GCCWH@dwdoc11-pc:1521:ora816
FILE=e:\MDL\gccstar-exp
LOG=e:\MDL\gccstar-imp-LOG
MODE=CREATE
CONFIGPARAM=N
```

Execute the Import Utility The following command invokes the Import Utility and specifies the above directive file:

```
w:\owb\bin\win32>imp parfile=e:\MDL\IMP_Directives.txt
Processing ...
Import successful.
```

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.

Mapping Definitions A mapping definition can be updated in a repository that is not identical to the mapping in the exported metadata file if any objects (tables, facts, transformations) that the mapping references cannot be found in the target repository. A warning message is written to the log file when this occurs.

Code Generation Before you generate scripts from imported definitions, first configure the definitions and then validate them. The validation identifies malformed definitions. For additional information on the configuration, validation and generation of scripts, refer to Chapter 9, "Configuring, Generating, and Deploying".

Foreign Key Definitions A foreign key definition can be updated in a repository that is not identical to the foreign key in the exported metadata file if its referenced unique or primary key does not exist in the target repository. A warning message is written to the log file that the foreign key does not contain a referenced key.

Splitter for Exporting and Importing OWB 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=*
- FACTS=*
- FILES=*
- DIMENSIONS=*
- VIRTUALTABLES=*
- TEMPORARYTABLES=*
- TRANSFORMCATEGORIES=*

The split utility splits the mappings within a module in a Warehouse Builder project. The size of the pieces are 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 below 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 a _imp.bat suffix is added to the import batch file.

The following example shows you 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 11–7.

Table 11–7 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.
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.

Keyword in Parameter File	Description
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

and so on.

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 Warehouse Builder is used to perform export, use multi-select to select and export objects other then 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 11–8 lists the files that are created.

Table 11–8 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.

About Batch Services

Use Batch Services to run bulk operations, such as the import and export of metadata directly from the command line. You can also use Batch Services to run Warehouse Builder operations such as validation, generation, and deployment of objects in a database. You can choose to run single operations, or you can use parameter and bulk parameter files to run multiple operations. When you run command line operations, log files are written and stored for each operation. Batch Services can also be scheduled to run independently from the Warehouse Builder client application.

The Batch Service command line scripts are in the following locations:

UNIX

- `$OWBHOME/owb/bin/solaris/batchservice.ksh`

Windows NT/2000

- `$OWBHOME\owb\bin\win32\batchservice.bat`

Using the Batch Services Command Line Syntax

The types of Batch Service command line scripts include:

- Validation
- Generation
- Deployment
- Metadata Import
- Metadata Export

You can run these scripts as single operations, or you can create a parameter file and a parameter file script that runs multiple operations using a single script. For more information, see "Running Multiple Command Line Operations" on page 11-30.

Table 11–9 provides descriptions for the notations used in the command line syntax for Batch Services.

Table 11–9 Batch Service Command Line Notations

Notation	Description
<user>	Enter the database user name to connect to the Warehouse Builder repository.
<duser>	Enter the database user name to connect to the runtime database.
<passwd>	Enter the password required by the user name to connect to the Warehouse Builder Repository.
<dpasswd>	Enter the password required by the user name to connect to the runtime database.
<host>	Enter the host machine name of the Warehouse Builder repository.
<dhost>	Enter the host machine name of the runtime database.
<SID>	Enter the SID of the Warehouse Builder repository.
<dSID>	Enter the SID of the runtime database.

Table 11–9 Batch Service Command Line Notations (Cont.)

Notation	Description
<port>	Enter the listener port number of the Warehouse Builder repository database.
<dport>	Enter the listener port number of the runtime database.
<project name>	Enter the name of the project that contains the object. **Note:** The Kornshell does not recognize strings enclosed in quotes as one argument. Use the standard typesetter ("_^") notation of inserting space to denote spaces. For example: Type "My Project" (with one space) as "My_^Project" in the command line.
<application name>	Enter the name of the module within the project that contains the object.
<element name>	Enter the exact name of the object. Case-sensitive.
<element type>	Enter the object type of any Warehouse Builder object. You can choose from one of the following: TABLE, VIEW, MATERIALIZEDVIEW, DIMENSION, FACT, MAPPING, SEQUENCE, TRANSFORMATION. **Note:** Use TRANSFORMATION to denote the object type for Transformation Categories.

The following tables contain the command line syntax to perform various operations. To perform these operations on transformations, you must use transformation-specific syntax.

- Table 11–10, "Validation Syntax"
- Table 11–11, "Generation Syntax"
- Table 11–12, "Deployment Syntax"
- Table 11–13, "Metadata Import Syntax"
- Table 11–14, "Metadata Export Syntax"

Note: The following tables provide the syntax using KSH scripts for UNIX systems. Use BAT scripts with Windows NT/2000 systems.

Table 11–10 Validation Syntax

Operation	Command Line Syntax
Validate	batchservice.ksh <user> <passwd> <host> <SID> <port> **-validate** <project name> <application name> <element name> <element type> [<logDir>] **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -validate "My_^Project" im1 tab1 TABLE /tmp/.owbObjects
Validate Transformation Categories	batchservice.ksh <user> <passwd> <host> <sid> <port > **-validate** <project> <module> {<transformation_library_folder>|<transformation_library_folder%%object_name>} TRANSFORM {<logdir>} **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -validate "My_^Project" IM1 Functions%%SortingTransform

Table 11–11 Generation Syntax

Operation	Command Line Syntax
Generate	batchservice.ksh <user> <passwd> <host> <SID> <port> **-generate** <project name> <application name> <element name> <element type> [<logDir>] **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -generate "My_^Project" im1 tab1 TABLE /tmp/.owbObjects
Generate Transformation Categories	batchservice.ksh <user> <passwd> <host> <sid> <port > **-generate** <project> <module> {<transformation_library_folder>|<transformation_library_folder%%object_name>} TRANSFORM {<logdir>} **For Example:** batchservice.ksh owb90004 owb sroychow-pc3 ora816 1521 -generate "My_^Project" IM1 Functions%%SortingTransform

Table 11–12 Deployment Syntax

Operation	Command Line Syntax
Deploy to File	batchservice.ksh <user> <passwd> <host> <SID> <port> **-deployfile** <project name> <application name> <element name> <element type> <deploy directory path> [<logDir>] **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -deployfile "My_^Project" im1 tab1 TABLE /tmp/scott /tmp/.owbObjects
Deploy to Database	batchservice.ksh <user> <passwd> <host> <SID> <port> **-deploydb** <project name> <application name> <element name> <element type> <duser> <dpasswd> <dhost> <dSID> <dport> [<logDir>] **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -deploydb "My_^Project" im1 tab1 TABLE owbrtusername owbrtpassword rtservername-pc ora817 1521 /tmp/.owbObjects
Deploy Transformation Category to File	batchservice.ksh <user> <passwd> <host> <sid> <port > -[deployfile] <project> <module> {<transformation_library_folder>|<transformation_library_folder%%object_name>} TRANSFORM {<logdir>} **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -deployfile "My_^Project" IM1 Functions%%SortingTransform /tmp/.username
Deploy Transformation Category to Database	batchservice.ksh <user> <passwd> <host> <sid> <port > -[deploydb] <project> <module> {<transformation_library_folder>|<transformation_library_folder%%object_name>} TRANSFORM <runtimeuser> <runtimepasswd> <runtimehost> <runtimesid> <runtimeport> **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -deploydb "My_^Project" IM1 Functions%%SortingTransform owbrtusername owbrtpassword rtservername-pc ora817 1521

Table 11–13 Metadata Import Syntax

Operation	Command Line Syntax
Import with Option to Use Physical Names	batchservice.ksh <user> <passwd> <host> <SID> <port> **-import** <import file name> <log file name> < mode> <use_physical_names > where <mode> = CREATE, REPLACE, UPDATE, or INCREMANTALUPDATE <use_physical_names> = true or false **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -import class_project.mdl log1.log CREATE true
Import with Option to Use Physical Names and Specify Character Set	batchservice.ksh <user> <passwd> <host> <SID> <port> -import <import file name> <log file name> < mode> <use_physical_names > <character set> where <mode> = CREATE, REPLACE, UPDATE, or INCREMANTALUPDATE <use_physical_names> = true or false **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -import class_project.mdl log1.log CREATE true WE8MSWIN1252
Import with Option to Use Physical Names, Ignore Universal ID, and Specify Character Set	batchservice.ksh <user> <passwd> <host> <SID> <port> -import <import file name> <log file name> < mode> <use_physical_names > <ignoreUniversalID> <character set> where <mode> = CREATE, REPLACE, UPDATE, or INCREMANTALUPDATE <use_physical_names> = true or false <ignoreUniversalID> = true or false **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -import class_project.mdl log1.log CREATE true true WE8MSWIN1252

Table 11–14 Metadata Export Syntax

Operation	Command Line Syntax
Export a Project	batchservice.ksh <user> <passwd> <host> <SID> <port> -exportproject <projectName> <export file name> <log file name> <character set> <physical names > <field separator> <physical name> {<config parameter>}. **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora816 1521 -exportproject "My_^Project" exportfile logfile WE8MSWIN1251 "|" true "Y"
Export a Module	batchservice.ksh <user> <passwd> <host> <SID> <port> -export <projectName> <application name> <export file name> <log file name> <character set> <physical names > <field separator> <physical name> {<config parameter>}. **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora816 1521 -export "My_^Project" IM1 exportfile logfile WE8MSWIN1251 "|" true "Y"

Note: You must include the single attached dash in front of an action. For example, -import is valid, but import and —import are not valid.

Running Multiple Command Line Operations

You can run multiple command line operations using a single script by creating parameter files. The parameter file is a separate text file that contains a batch of command line operation scripts with the same syntax used in the single operations. When you type the parameter file command line script to run the parameter file, you only need to specify the parameter file. Parameter files enhance performance by providing a single connection for all of the operations listed in the file. You can also choose to use a bulk parameter file for very large groups of operations to improve performance. Bulk parameter files refer to outside files such as data files and error files.

Table 11–15 contains the command line syntax for the parameter and bulk parameter file scripts.

Table 11–15 Parameter File Command Line Syntax

Operation	Command Line Syntax
Executing a Parameter File with Deployment to Database	batchservice.ksh <user> <passwd> <host> <SID> <port> -paramfile <paramfile> [<duser> <dpasswd> <dhost> <dSID> <dport>] **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -paramfile batchservce.param owbrun owbrun sever-pc3 ora817 1521
Executing a Parameter File without Deployment to Database	batchservice.ksh <user> <passwd> <host> <SID> <port> -paramfile <paramfile> **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -paramfile batchservce.param
Executing a Bulk Parameter File with Deployment to Database	batchservice.ksh <user> <passwd> <host> <SID> <port> -bulkparamfile <bulkparamfile> [<duser> <dpasswd> <dhost> <dSID> <dport>] **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -bulkparamfile bulkbatchservce.param owbrun owbrun sever-pc3 ora817 1521
Executing a Bulk Parameter File without Deployment to Database	batchservice.ksh <user> <passwd> <host> <SID> <port> -bulkparamfile <bulkparamfile> **For Example:** batchservice.ksh owbusername owbpassword servername-pc ora817 1521 -bulkparamfile bulkbatchservce.param

Example 11–3, Example 11–4, Example 11–5, and Example 11–6 display the contents of a sample parameter file, bulk parameter files, and data file. Use the comments in the example as guidelines for creating these files.

Example 11–3 Parameter File

```
=============================================================================
# Place a pound sign before your comments
# validate "My_^Project" <application Name> <element Name> <element Type> <logdir>
validate "My_^Project" IM1 TAB1 TABLE  /tmp/.owbObjects
# generate
# generate "My_^Project" <application Name> <element Name> <element Type> <logdir>
generate "My_^Project"  IM1 TAB1 TABLE /tmp/.owbObjects
#deploy
```

```
# similarly we do for deploy in file system and the database.
deploydb "My_^Project"  IM1 TAB1 TABLE <logdir>
import import.mdl importmdl.log CREATE true true WEMSWIN1251
=====================================================================
```

> **Note:** The keywords are in lowercase, and there is no dash in front of the keyword like in the command line option.

Example 11–4 Bulk Parameter File

```
=====================================================================
# pound sign is for comments
# The bulkErrorFile is the location of the error report.
bulkErrorFile = /somebulkErrorFile
# The defaultprojectAppResultDir is the directory the results are stored in.
# If you leave it empty then it takes the current directory.
defaultProjectAppResultDir  = ~/
# The project is where you enter the name of the Project.
project = "My_^Project"
# The application is the name of the module.
application = IM1
# For projectAppFCOData, specify the data file for this module with the path.
# The data file contains the actions to be performed on specified objects.
projectAppFCOData = /tmpA/Object/data1.dat
# projectAppFCODeployFileOK specifies whether to deploy to file or not.
# The default is false.
projectAppFCODeployFileOK = false
# projectAppFCODeployFileDir specifies the directory to deploy the generated scripts to.
projectAppFCODeployFileDir = /tmpA/Object/
# projectAppFCODeployFileType specifies the file type to deploy.
# There are three options: DDL,ANALYZE, or ALL. The default is ALL.
projectAppFCODeployFileType=  DDL
# projectAppFCODeployDataBaseOK specifies whether to deploy to database or not.
# The default is false.
projectAppFCODeployDataBaseOK = false
# projectAppFCODeployDatabaseType is the type of file you are deploying.
# There are two options: PLS and ALL. The default is ALL.
projectAppFCODeployDatabaseType = PLS
# The projectAppResultDir is the directory where Batch Services results are sent to.
projectAppResultDir  = /tmpA
# end of Bulk Setup parameter file
=====================================================================
```

Example 11–5 Import Bulk Parameter File

```
=====================================================================
# pound sign is for comments
```

```
bulkErrorFile = /somebulkErrorFile
defaultProjectAppResultDir  = ~/
# Provide a full path for the import. The default is the current directory.
importFullPath =
# Specify the MDL files to import. You can import multiple MDL exported files.
# See note following this example for more information.
importFiles = A*.mdl
# Specify whether or not to run import. The default is true.
importMode = true
# Specify whether or not to import by physical names.
importUsePhysicalName =
# Specify whether or not to ignore the UOID when importing. The default value is true.
importIgnoreUOID =
# Provide a character set string.
importCharacterSet =
=====================================================================
```

You can import files by:

- Specifying MDL files by name, such as A1.mdl, A2.mdl.
- Using wildcards, such as *.mdl, A*.mdl, and *_Mapp_*.mdl.

Example 11–6 Data File

```
=====================================================================
# pound sign is for comment
# OBJECT NAME, OBJECT TYPE ACTION SET
# <elementname>,<elementype> {<service> <, >}+
TAB1,TABLE validate generate deploydb
TAB2,TABLE validate generate deployfile
MAPP1,MAPPING generate deploydb
# end of FCO Data file
=====================================================================
```

Locating and Reviewing Log Files

The output for validation, generation, and deployment are stored in their respective log files. The log file has the following typical structure. The messages are shown in italics. The action and the date of activity are highlighted in bold. Example 11–7 shows a validation log file.

Example 11–7 Validation Log File

```
BEGIN ----------------------------------------------
Input:
owbusername owbpassword servername-pc ora817 1521  -validate  My_^Project  im1  tab1
TABLE
```

```
Output:
directory = /owb/owbhome/owb/bin/im1_tab1_validate.log

Action = validate   Date = Mon Apr 02 14:30:26 GMT-08:00 2001
project = im1,  element = tab1,  type    = TABLE,
"ERROR", "No Columns", "Add new Column"
END   ----------------------------------------------
```

About the Warehouse Builder Transfer Wizard

The Warehouse Builder Transfer Wizard enables you to synchronize, integrate, and use metadata stored in multiple sources and formats.

Warehouse Builder Transfer Wizard Overview

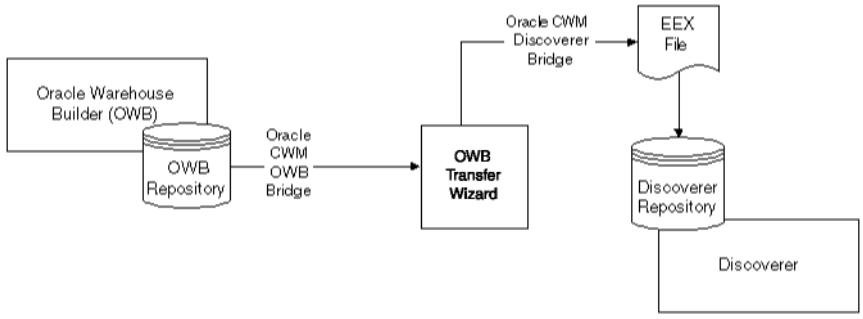
The Warehouse Builder Transfer Wizard allows you to import metadata from and export metadata to data warehousing tools. The Transfer Wizard performs two major tasks:

1. Exporting selected metadata from the Warehouse Builder repository using a bridge to a variety of targets. The target can be:
   - Object Management Group (OMG) file
   - Oracle Discoverer versions 3.1 and 4*i*
   - Oracle Express
   - Oracle 9*i* OLAP Server
2. Importing selected metadata from source tools into the Warehouse Builder repository using a bridge. The source can be:
   - Object Management Group (OMG) file
   - Computer Associates ERwin (3.5.1)
   - Powersoft PowerDesigner (version 6)
   - Oracle 9*i* OLAP Server

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 11–9 shows an example of the transfer process for exporting metadata from Warehouse Builder into Discoverer.

Figure 11–9 Warehouse Builder Transfer Wizard Model



Transfer Considerations

Before you transfer metadata between two data warehousing tools, you need to perform tasks within the source tool to ensure that the metadata transfers successfully and displays appropriately in the target tool.

For detailed information on transfer considerations for metadata import and export, refer to Appendix H, "Warehouse Builder Bridges: Transfer Parameters and Considerations".

Importing Metadata into Warehouse Builder

After you have prepared your source tools to ensure a successful transfer, you can use the Warehouse Builder Transfer Wizard to import the metadata. For more information on transfer considerations, refer to Appendix H, "Warehouse Builder Bridges: Transfer Parameters and Considerations".

The Warehouse Builder Transfer Wizard enables you to import metadata from the following types of sources:

- A flat file that conforms to the OMG standard
- Computer Associates ERwin
- Powersoft PowerDesigner
- Oracle9*i* database containing OLAP objects

This section contains instructions for using the Warehouse Builder Transfer Wizard to import metadata into Warehouse Builder:

For more information on transfer considerations, refer to Appendix H, "Warehouse Builder Bridges: Transfer Parameters and Considerations".

To launch the Transfer Wizard for an import:

1. Open Oracle Warehouse Builder.
2. Select the Administration view from the vertical tool bar at the left of the Warehouse Builder console window.
3. From the **Administration** menu, select **MetaData Import,** and then **Bridge.**

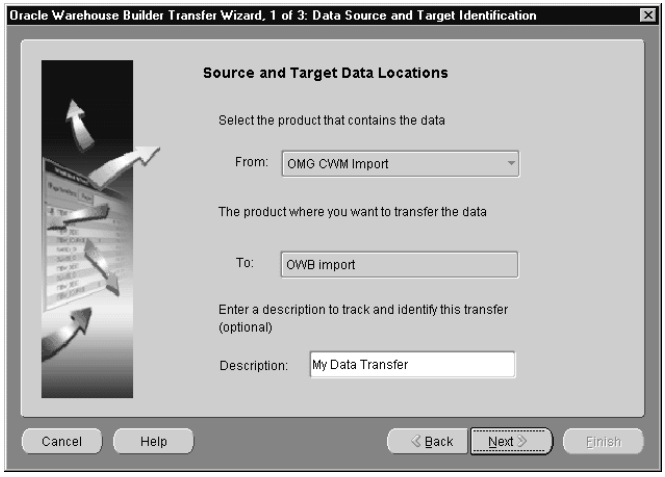
   The Oracle Transfer Wizard Welcome window displays, identifying the steps you perform while using the Transfer Wizard.

   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.
4. Click **Next.**

   The Data Source and Target Identification window displays.
5. In the **From** field, identify your metadata source (OMG, CA ERwin, PowerDesigner). In the **To** field, accept the default (Warehouse Builder import).
6. Optionally, enter a **Description** of the metadata to be transferred.

   This description displays in the progress bar during the transfer process and is useful when you are performing multiple transfers.

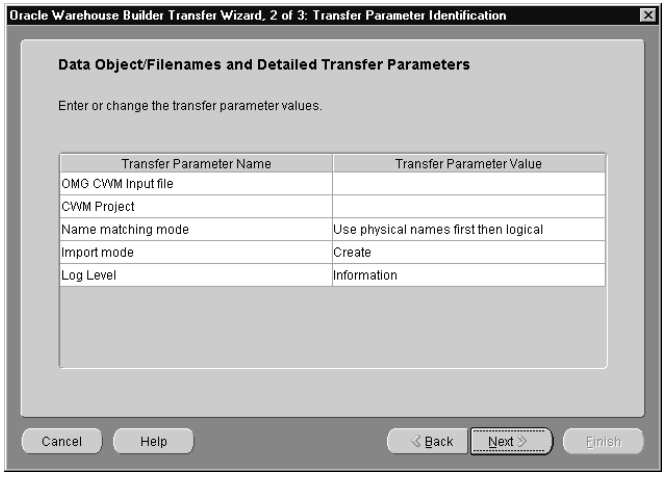
Figure 11–10 Data Source and Target Identification Window



7. Click **Next.**

The Transfer Parameter Identification window displays.

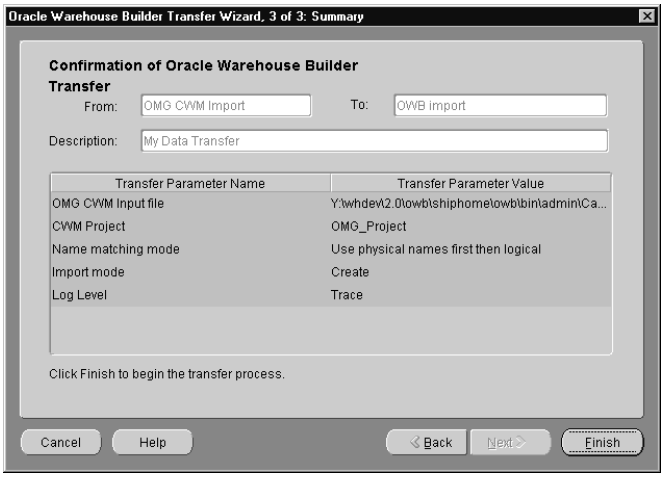
Figure 11–11 Transfer Parameter Identification Window



The Transfer Parameters window displays a different list of parameters based upon the metadata source you selected.

8. Click **Next.** The Summary window displays.

Figure 11–12 Summary Window



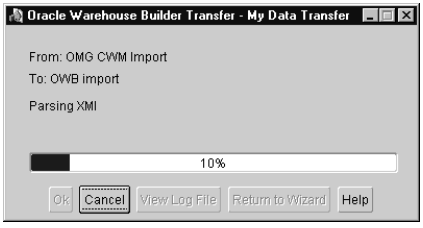
9. Review your entries.

   If any are incorrect, click **Back** to return to the previous screen and make the necessary changes.

10. Click **Finish** on the Confirmation window.

    The transfer window displays with a status bar.

Figure 11–13 Data Transfer Progress Panel



The title of the transfer window is the description you provided. If you did not provide a description, a title does not display.

> **Note:** The transfer can require several minutes or an hour or more to complete, depending on the amount of metadata you transfer.

11. 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.)

      > **Note:** 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.

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

After you have prepared your target tools to ensure a successful transfer, you can use the Warehouse Builder Transfer Wizard to export the metadata. For more information on transfer considerations, refer to Appendix H, "Warehouse Builder Bridges: Transfer Parameters and Considerations".

The Warehouse Builder Transfer Wizard enables you to export metadata to the following types of targets:

- A file that conforms to the OMG standard
- Oracle Discoverer 3.1
- Oracle Express
- Oracle Discoverer 4*i*

- Oracle 9*i* Database to store OLAP objects

This section contains instructions for using the Warehouse Builder Transfer Wizard to export metadata from a Warehouse Builder project to a target. For more information on transfer considerations, refer to Appendix H, "Warehouse Builder Bridges: Transfer Parameters and Considerations".

To export metadata using the Warehouse Builder Transfer Wizard:

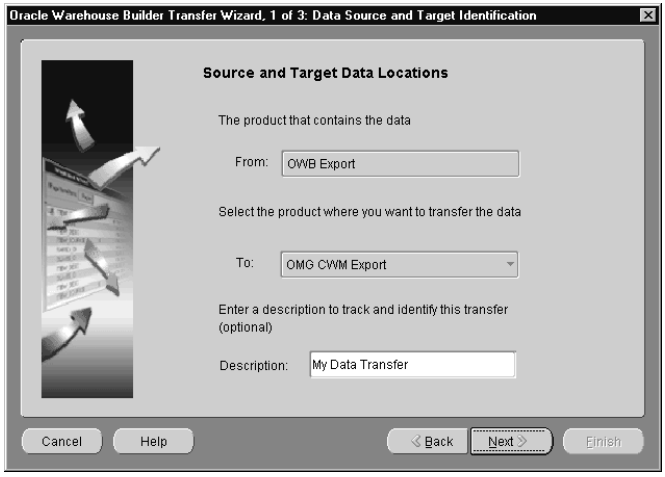
1. Open Oracle Warehouse Builder.
2. Select the Project mode from the vertical tool bar at the left of the Builder client window.
3. From the **Project** menu, select **MetaData Export,** and then **Bridge.**

   The Oracle Transfer Wizard Welcome window displays, identifying the steps you perform while using the Transfer Wizard.
4. Click **Next.**

   The Data Source and Target Identification window displays.
5. In the **From** field, accept the default (Warehouse Builder Export) and in the **To** field, select your target (OMG, Discoverer 3.1, Express, or Discoverer 4*i*) to identify the target for your export.
6. Optionally, enter a **Description** of the metadata to be transferred.

   This description displays in the progress bar during the transfer process and is useful when you are performing multiple transfers.

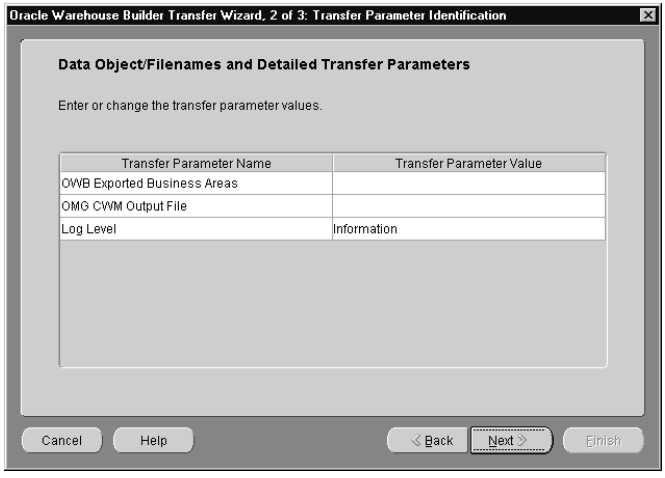
Figure 11–14 Source and Target Information Window



7. Click **Next.**

The Transfer Parameter Identification window displays.

Figure 11–15 Transfer Parameter Identification Window

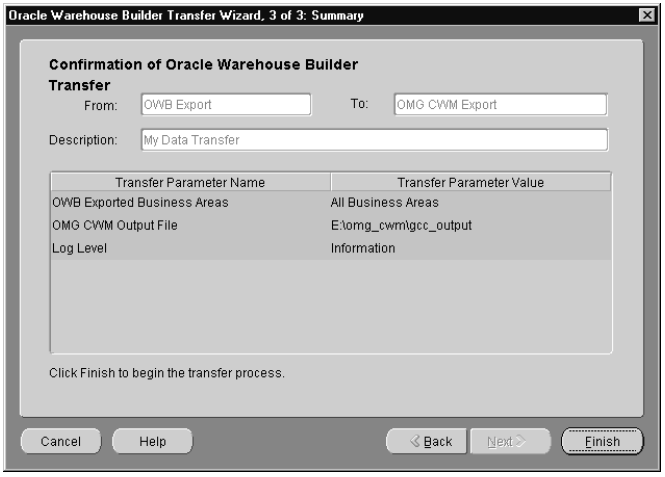


The Transfer Parameters window table lists parameters that you must enter or select. This window displays a different set of parameters depending on the target you selected in the previous step.

8. Click **Next.**

   The Confirmation of Warehouse Builder Transfer window displays.

Figure 11–16 Summary Window



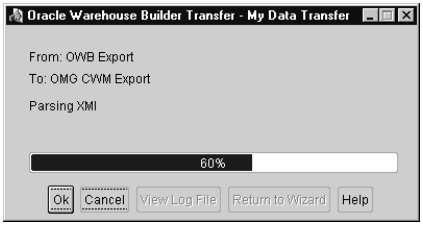
9. Review your entries.

   If any are incorrect, click **Back** to return to the previous screen and make the necessary changes.

10. Click **Finish.**

    The transfer window displays with a status bar.

Figure 11–17 Data Transfer Progress Panel



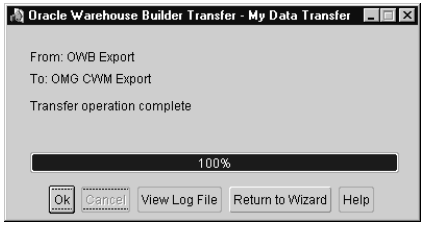
The title of the transfer window is the description you assigned to the transfer on the Choose Data Source and Target Types window. If you did not provide a description, a title does not display.

Note: The transfer can require several minutes or an hour or more to complete, depending on the amount of metadata you transfer.

11. Do one of the following:

- If the transfer completes successfully, click **OK.**

Figure 11–18 Data Transfer Complete Panel



- If a transfer failure message displays, click **View Log File** and review the log. You can also view the log of a successful transfer.

Note: 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**.

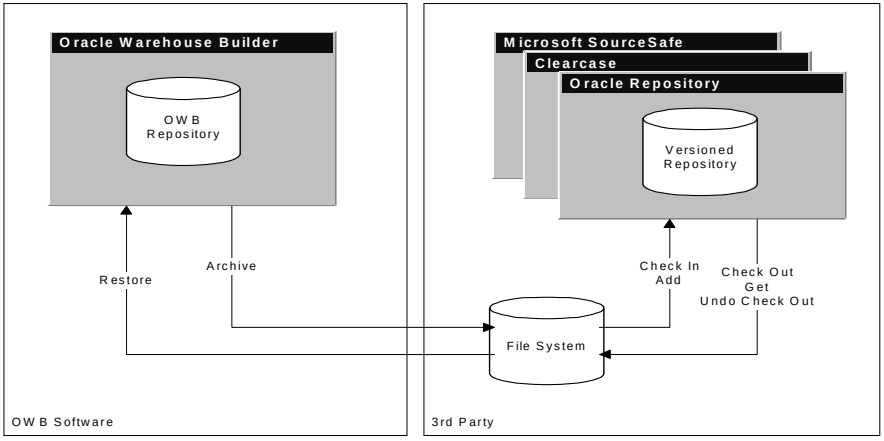
- If you determine the cause of the failure from the Information Log, note the data requiring update. Close the log by clicking **OK.** On the transfer window, click **Return to Wizard** and update the erroneous data on the Transfer Parameters window. Then transfer the data again.
- If you cannot determine 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.

About the Archive/Restore Utility

The Archive/Restore Utility allows you to save and load versions of your Warehouse Builder repository metadata. 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.

Figure 11–19 Archive/Restore



Use the following sections to assist you with your Archive/Restore:

- Setting up Preferences
- Archiving a Project
- Restoring a Project

> **Note:** 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.

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 allows 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.

Archive and Restore are different from Import and Export. Table 11–16 and Table 11–17 describe the differences between these features.

Table 11–16 Differences Between Archive and Export

Feature	Archive	Export
Character Set	UTF8	User Configured
Field Separator	Pipe Character (|)	User Configured
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

Table 11–17 Differences Between Restore and Import

Feature	Restore	Import
Character Set	UTF8	User Configured
Complete Project Replacement	Yes	User must first delete project
Dump Format	MDL	MDL
UniversalID Preservation	Always	User Configured
Name Preservation	Always	User Configured
Log File Name	Generated	Generated and User configured

Setting up Preferences

You can set up most of the archive and restore specifications using the Preferences dialog from the Administration or Project views. Use this dialog to specify labeling options, archive / restore directory locations and log folders.

To set up Archive/Restore preferences:

1. Select **Preferences** from the Administration or Project menus depending on the view you selected.

   The Preferences dialog displays.

2. Select the **Archive/Restore** tab if it is not displayed on top.

Figure 11–20 Archive/Restore Preferences

3. Browse to or create a default root folder for archiving.

   The archive service creates additional folders.

4. Specify how you want to handle the label. The options are listed in Table 11–18.

Table 11–18 Label Options

Label Option	When to Use	Generated Path Name
Do not include	For Source Control Management when the path name never changes.	The path is consistent and not dependent on the version label value.

Table 11–18 Label Options

Label Option	When to Use	Generated Path Name
Parent Folder	For Source Control Management when the label value changes slowly. For example, labels like Development, Alpha, Beta, and Production. Also good for dumps to a file system.	The generated path changes with each new version label.
File name	When dumping files to a file system.	The generated path changes with each new version label. This appends the label name as the base part of the terminal file name. For example, if the label is development, the file name is development.mdl. If there is no label and this mode is selected, the file name is nolabel.mdl.

5. Browse to or create a location for your Archive Log Folder.

   This folder contains the archive log files as they are created. The log files contain detailed information about the archive including how the file was created and the contents of what was archived. The name of each log file is automatically derived from the logical project name that is being archived. All archive log files start with A and incorporate the label, date, and time to make them unique.

6. Browse to or create a location for your Restore Log Folder.

   This folder contains the restore log files as they are created. The log files contain detailed information about the restore including how the file was created and the contents of what was restored. The name of each log file is automatically derived from the logical project name contained in the archive. All restore log files start with R and incorporate the label, date, and time to make them unique.

7. Click **OK.**

   The preferences have been set. You can now proceed to archiving and restoring your project.

Archiving a Project

Archiving a project allows you to copy metadata stored within a Warehouse Builder repository to an external location for the purpose of securing that data at a fixed

point in time. Warehouse Builder provides an Archive Wizard to assist you in this process.

> **Note:** Before you archive your project, you can update the project version label with the Project Properties dialog.

To archive a project:

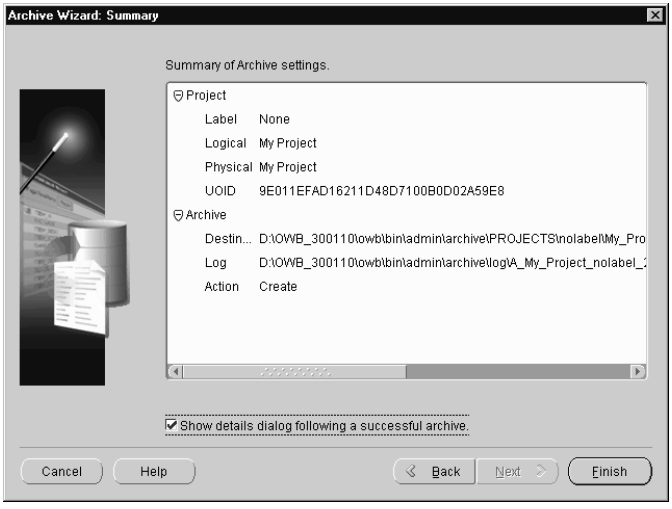
1. Select **Archive** from the Project menu or from the Administration menu depending on the view you selected. You can also select **Archive** from the right-click menu when a project is selected.

   The Archive Wizard Welcome page displays.

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.

Figure 11–21 Archive Wizard Summary Page



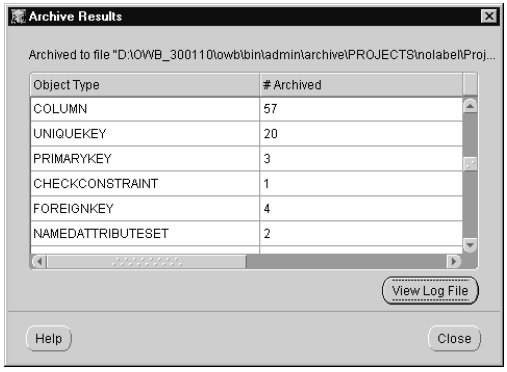
Note: No changes can be made from the wizard. If you notice an error in the Archive Wizard Summary page, click **Cancel** and make the appropriate changes to your Archive/Restore Preferences before continuing with the archive.

3. Click **Finish.**

   This begins the archive process. A progress window appears. When the progress bar reaches 100%, the archive process is complete.

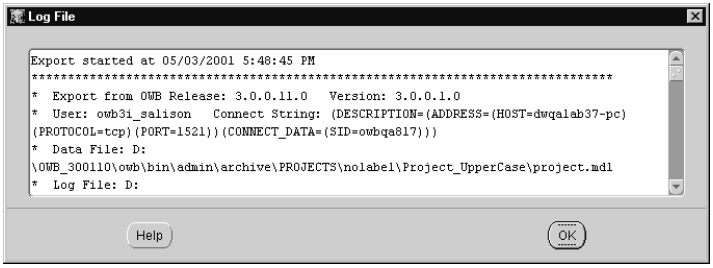
   If you checked the **Show details dialog following a successful archive** box, the Archive Results dialog displays. This dialog displays the name of each object type and how many of each were archived.

Figure 11–22 Archive Results



   For a more detailed look at the archive process, click **View Log File.** This displays the entire log file.

Figure 11–23 Archive Log File



Restoring a Project

Restoring a project allows you to recreate metadata within a Warehouse Builder repository from an external location.

To restore a project:

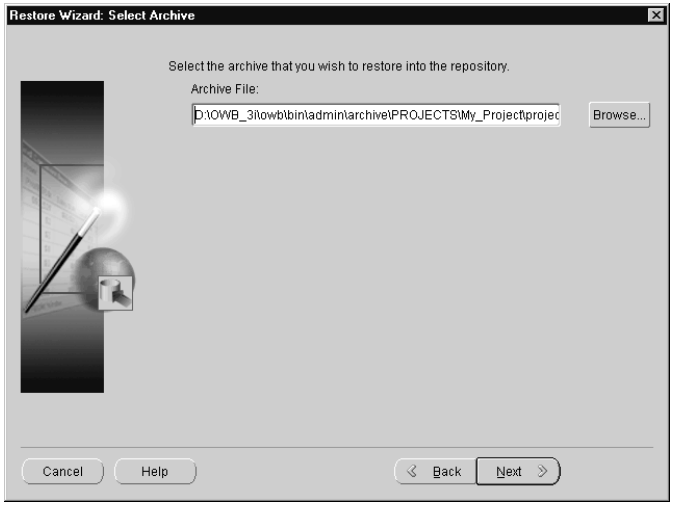
1. Select **Restore** from the Administration menu in the Administration View.

   The Restore Wizard Welcome page displays.

2. Click **Next.**

   The Select Archive page displays. Browse to or type the Archive File you want to restore.

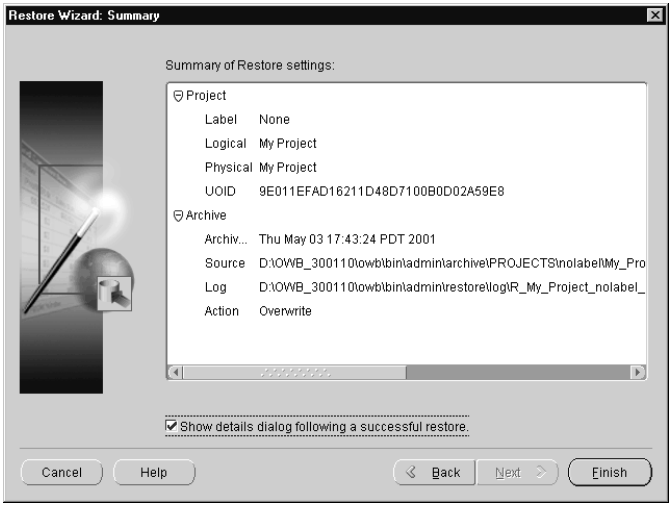
Figure 11–24 Select Archive Page



3. Click **Next.**

The Summary page displays a summary of the restore settings prior to running the restore process. 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 11–25 Restore Wizard Summary Page

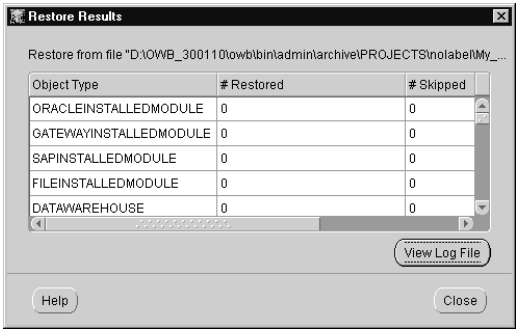


4. Click **Finish.**

   This begins the restore process. A progress window appears. When the progress bar reaches 100%, the restore process is complete.

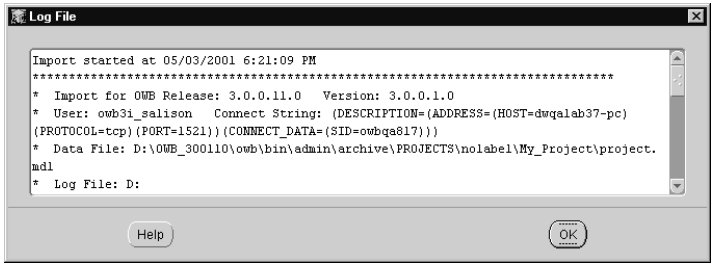
   If you checked the **Show details dialog following a successful restore** box, the Restore Results dialog displays. This dialog displays the name of each object type, how many of each were restored, and how many of each were skipped.

Figure 11–26 Restore Results



   For a more detailed look at the archive process, click **View Log File.** This displays the entire log file.

Figure 11–27 Restore Log File



Using Discoverer Workbooks for Metadata and Runtime Reporting

Warehouse Builder enables you to run reports on its design time metadata and runtime audit data using Oracle Discoverer workbooks. To run these reports, you must first install and access the Discoverer Workbooks packaged with Warehouse Builder. Discoverer workbooks can be used to run three types of reports: Definition reports, QA reports, and Runtime Reports.

- **Definition reports:** used to report on design objects within the Warehouse Builder repository. Drill paths within these reports enable you to drill down on related data.
- **QA reports:** used to report on incomplete and invalid object definitions. These reports are useful in tracking down logical errors in your design, such as facts without dimensions, dimensions without levels, mismatched data types in column mappings, or unused sources and targets within mappings.
- **Runtime reports:** used to report on runtime data and to track down runtime errors.

Warehouse Builder is packaged with a set of Discoverer files, including import End User Layer (EUL) templates (.eex) and Discoverer workbook files (.dis). The following sections describe how to use these files to install and set up the EULs in Discoverer and how to access the Discoverer workbooks.

To install and access the Discoverer workbooks:

1. If necessary, establish a database connection and grant user privileges to enable access between the Discoverer administrator database and Warehouse Builder database.
2. Create an .eex file using the files and templates packaged in Warehouse Builder.

3. Import the .eex file into Discoverer Administration tool using the Import Wizard.
4. Open the workbooks and run reports using the Discoverer Plus Edition or the Discoverer User Edition.

Installing the EUL Template and Workbooks

Before you can install the EUL templates and access the Discoverer workbooks, you must locate the necessary files packaged with your Warehouse Builder installation. You must also provide the required database connection information to enable the Discoverer administrator to access the Warehouse Builder data.

Prerequisites

- Install Oracle Discoverer. See the *Oracle Discoverer Administration Installation Guide* for details.
- Install Warehouse Builder and locate the following files in the DiscoMetaDataReports-Design and DiscoMetaDataReports-Runtime directories.

  The following files used to install the **Definition** and **QA** workbooks are located at: **\<OracleHome>\owb\misc\DiscoMetaDataReports-Design**

  - EUL Template: owbdefqa_template.eex
  - Definition Workbook: owbdef.dis
  - QA Workbook: qa.dis
  - Utility Files: eexfix.bat and eexfix.class

  The following files used to install the **Runtime** workbook are located at: **\<OracleHome>\owb\misc\DiscoMetaDataReports-Runtime**

  - EUL Template: owbruntime_template.eex
  - Runtime Workbook: runtime.dis
  - Utility Files: eexfix.bat and eexfix.class

Setting Database Connections

To report on Warehouse Builder metadata, the Discoverer EUL must point to the correct instance in that database. The Discoverer administrator has a user account on the database where it is installed. If the Warehouse Builder metadata resides in the same database, you must provide the administrator with the user account for

that instance and provide select privileges on public views. If the Warehouse Builder metadata resides in a different database, you must create a database link to enable the Discoverer administrator to connect to that database. Refer to the *SQL Reference Manual* for information on how to create a database link.

Creating Specific .eex Files

Once you provide this connection information and the administrator is able to connect to the Warehouse Builder metadata, you can create the .eex files by running the following scripts.

- To create an .eex file for **Definition** and **QA** workbooks, run the following script from a **DOS** prompt in the \OracleHome\owb\misc\DiscoMetaDataReports-Design directory:

  eexfix owbdefqa_template.eex mydesign.eex <OWBuser> [<OWBlink>]

  <OWBuser> is the user account and **<OWBlink>** is the database link required to access the Warehouse Builder metadata.

  Note: The user name and database link must be uppercase.

- To create an .eex file for **Runtime** workbook, run the following script from a DOS prompt in the \OracleHome\owb\misc\DiscoMetaDataReports-Runtime directory:

  eexfix owbruntime_template.eex myruntime.eex <OWBuser> [<OWBlink>]

Once you run this script, a .eex file is created and stored in the same directories as the packaged files.

- For Definition and QA Workbooks, the .eex file is stored in, **\OracleHome\owb\misc\DiscoMetaDataReports-Design.**
- For Runtime Workbooks the .eex file is stored in, **\OracleHome\owb\misc\DiscoMetaDataReports-Runtime.**

You are now ready to import these files into the Discoverer 4.0 and view and edit the EUL using Discoverer Administration Edition.

Importing .eex Files into Discoverer

You can import the .eex files into Discoverer by using the Import Wizard in the Administration tool. Refer to the *Discoverer Administration User's Guide* for details on importing .eex files.

Accessing the EUL within Discoverer Administration

After you import the .eex files, you can access the EUL for Warehouse Builder metadata and runtime audit data using the Discoverer Administration tool. This EUL is built using public views and contains high-level objects defined within Warehouse Builder projects. This reporting structure also facilitates user additions and extensions in future.

In the Administration tool, you can view a list of all the public views (grayed out) contained in the EUL that are not visible to the end user. These public SQL views, published with the Warehouse Builder installation, can also be used to construct new customized EULs.

Figure 11–28 EUL for Definition and QA Workbooks

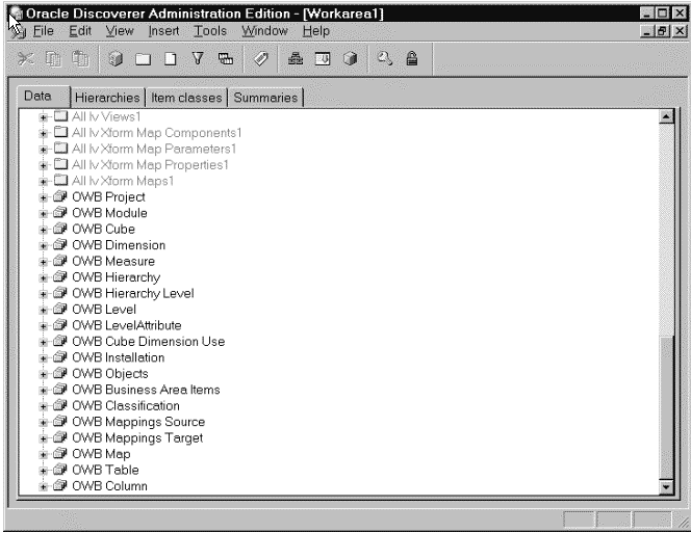


Table 11–19 lists all the folders within the EUL for Warehouse Builder Definitions and QA workbooks and describes their contents. For descriptions of all the public views (grayed out and prefixed by All), refer to Appendix H, "Warehouse Builder Bridges: Transfer Parameters and Considerations".

Table 11–19 Description of Warehouse Builder Design EUL Folders

EUL Folder	Description
OWB Project	The root node of Warehouse Builder design. A structure within the Warehouse Builder repository where you store definitions that describe your logical warehouse
OWB Module	Containers within the Warehouse Builder Projects that store definitions to define and populate your data warehouse.
OWB Cube	Core Fact entity in the Warehouse Builder dimensional model. They are central tables that hold the aggregate data from one or more data tables and contain foreign keys to dimension tables.
OWB Dimension	Entity in the Warehouse Builder dimensional model that organizes and indexes data for the fact tables.
OWB Measure	Part of the Warehouse Builder cube.
OWB Hierarchy	Hierarchies contained within each Warehouse Builder dimension that describe the parent-child relationship among a set of levels.
OWB Hierarchy Level	Levels contained within the Warehouse Builder hierarchy.
OWB Level	Levels contained within the Warehouse Builder dimensions. A combination of levels forms a hierarchy.
OWB Level Attribute	Attributes that define the level and the columns within the level.
OWB Cube Dimension Use	Describes how a cube relates to the dimensions.
OWB Installation	Contains information on this particular Warehouse Builder installation
OWB Objects	All the objects in the Warehouse Builder design.
OWB Business Area Items	Items within a Business Area in a Warehouse Builder Project.
OWB Classification	Used to generally classify objects (implements business areas).
OWB Mappings Source	The sources of Warehouse Builder Mappings.
OWB Mappings Target	The targets of a particular source.
OWB Map	Warehouse Builder mappings and their targets.

Table 11–19 Description of Warehouse Builder Design EUL Folders (Cont.)

EUL Folder	Description
OWB Table	Table objects in the Warehouse Builder design.
OWB Column	Columns contained within the tables in the Warehouse Builder design.

Figure 11–29 EUL for Runtime Workbook

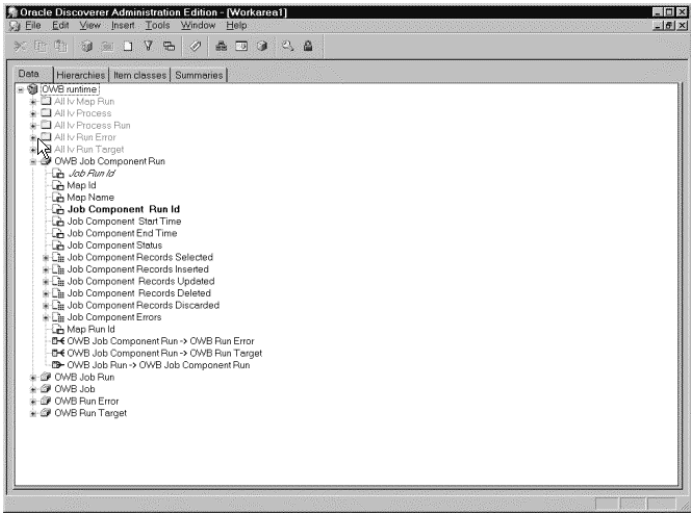


Table 11–20 lists all the folders within the EUL for Warehouse Builder Runtime workbook and describes their contents. For descriptions of all the public views (grayed out and prefixed by All), refer to Appendix H, "Warehouse Builder Bridges: Transfer Parameters and Considerations".

Table 11–20 Description of Warehouse Builder Runtime EUL Folders

EUL Folder	Description
OWB Job	The job or process that is executed from time to time.
OWB Job Run	The individual runs of a job.
OWB Job Component Run	The job components within a specific job run.
OWB Run Target	The target table of a particular job component run.
OWB Run Error	The errors encountered while running the job component.

Accessing Workbooks using Discoverer User Edition

To open the workbooks and run reports on Warehouse Builder metadata, connect to the Discoverer Plus Edition and open the workbooks **owbdef.dis** (for the Definition workbook) and **qa.dis** (for the QA workbook) from the following location:

\OracleHome\owb\misc\DiscoMetaDataReports-Design

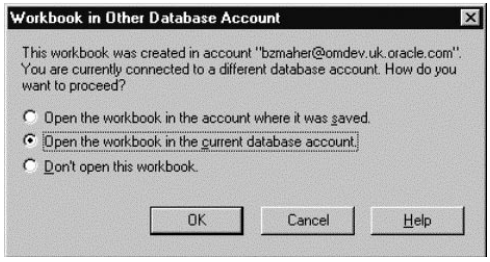
You can open the **runtime.dis** file (for the Runtime workbook) from the following location:

\OracleHome\owb\misc\DiscoMetaDataReports-Runtime

These workbooks connect to the EULs previously installed.

If the Workbook in Other Database Account dialog displays, choose the default **Open the workbook in current database account.**

Figure 11–30 Workbook in Other Database Account



If you receive a warning about missing objects, the workbook cannot connect to the EUL. Check for incorrectly specified .eex file generation parameters. To validate these parameters, cancel out of this workbook and follow the prompts to create a table query based on the Warehouse Builder definition metadata EUL. If you do not see any objects in the Warehouse Builder definition metadata EUL, then the user you entered does not exist or does not have access to the data. Check to see if you have used a wrong user or entered the user name in lower case when creating the .eex file.

To check the information entered in the .eex file, open Discoverer Administration, open the Business Area Warehouse Builder definition metadata, right-click on one of the grayed out folders, and check its properties for the user name.

Note: The user name must be entered in uppercase.

If you need to correct an error, you must rerun the .eex file generator with the correct user name and database link, delete this Business Area, and re-import it into Discoverer. You must also reconnect the Discoverer Plus Edition.

Navigating Workbooks

A workbook can contain multiple worksheets that enable you to document and manage your Warehouse Builder data. For example, the QA workbook contains worksheets for facts without dimensions, source objects not used, and mismatched data types. Reports or query results displayed in a worksheet can be edited and printed.

You can make your reports easier to read and more informative by reformatting them. To reformat the width of the columns, position your cursor in the column header between columns, so that it displays arrows pointing in both directions. Drag the column to make it wider or narrower.

When viewing Warehouse Builder metadata, you can choose objects from projects and modules defined within the Warehouse Builder repository. A down-pointing triangle to the right of the Page Items header signifies that you can click on the arrow and choose the Project Name, the Module Name, or the Table Name (when applicable).

Worksheets enable you to perform drill-down analysis on hierarchical data. A right-pointing triangle to the left of a column header signifies that you can drill down on that object. To perform a drill down:

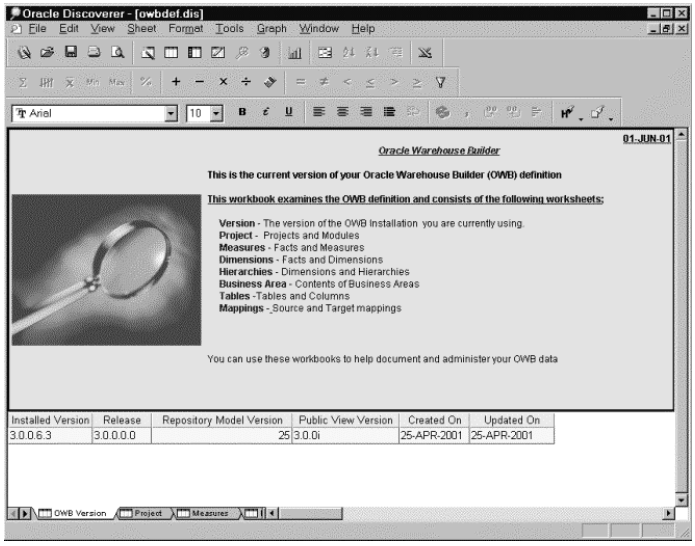
- Select the object you want to drill on.
- Right-click the object and select **Drill** from the pop-up menu.

The report displays the properties and hierarchical relationships for that object.

Definition Workbook

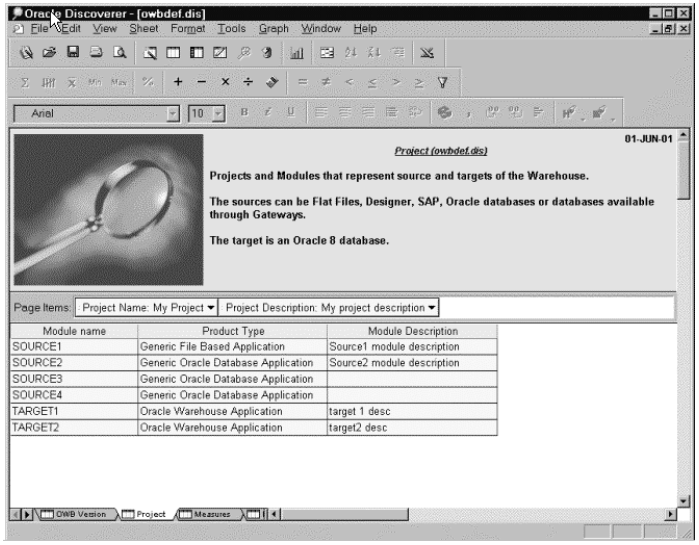
The definition workbook enables you to run reports on the objects defined within the Warehouse Builder repository, for example, Project, Measures, Dimensions, Hierarchies, Business Area, Tables, and Mappings.

Figure 11–31 Workbook Definition



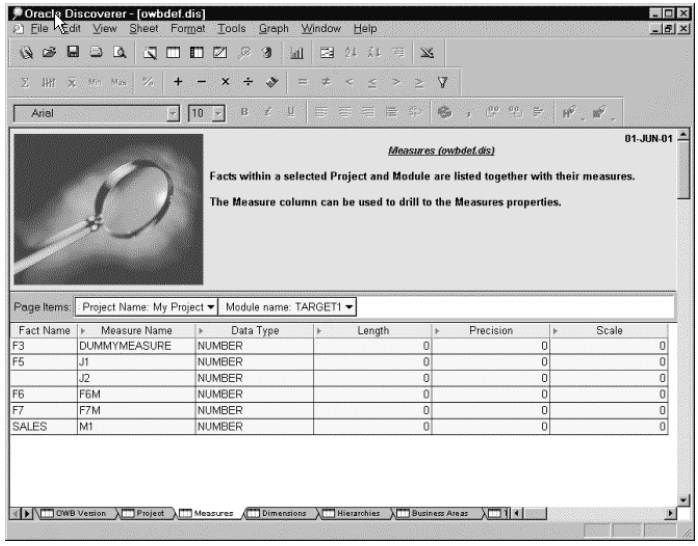
The **Version** worksheet within the Definitions Workbook displays all the worksheets available within this workbook.

Figure 11–32 Project Worksheet



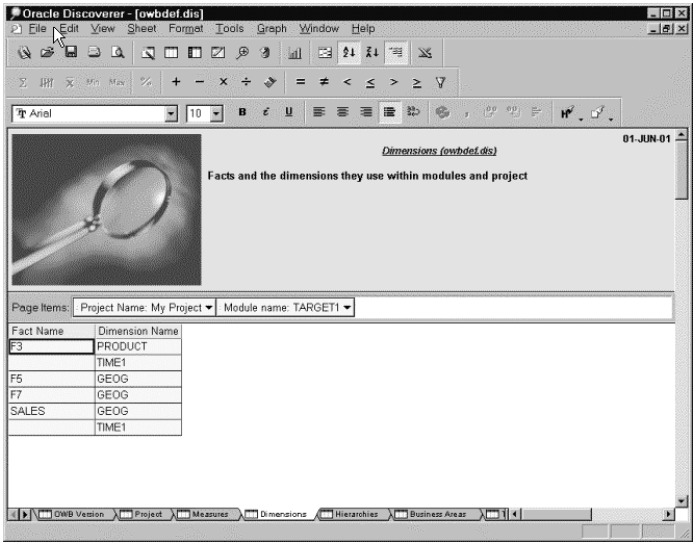
The **Project** worksheet contains the projects and modules defined within a logical Warehouse Builder design. The worksheet displays both source (Flat File, SAP, Designer, Oracle Database) and target modules.

Figure 11–33 Measures Worksheet



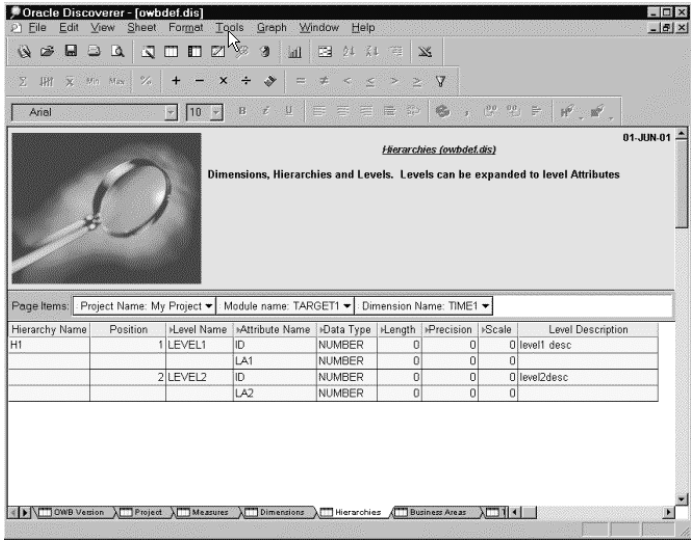
The **Measures** worksheet lists the Fact tables defined within the selected project and module, along with their measures. You can further drill down on the Measure Name column to view measure properties such as data type, length, precision, and scale.

Figure 11–34 Hierarchy Worksheet



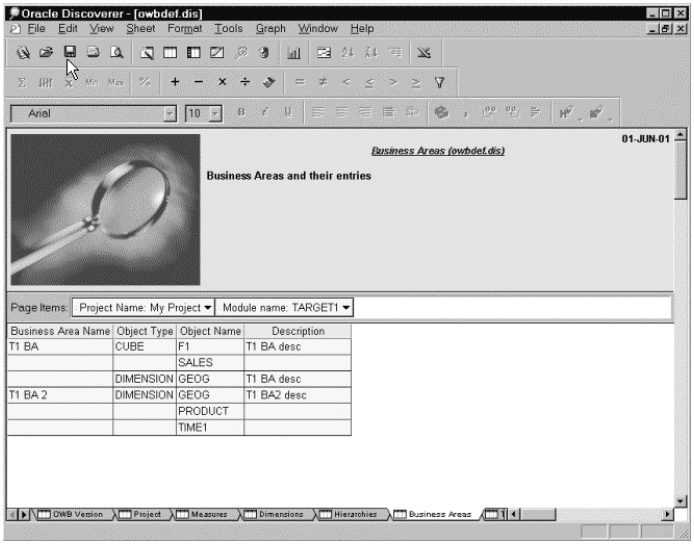
The **Hierarchy** worksheet lists all the dimensions, hierarchies, and levels defined within the selected project and module. You can drill on level names to view the level attributes and further on to view the attribute properties.

Figure 11–35 Dimensions Worksheet



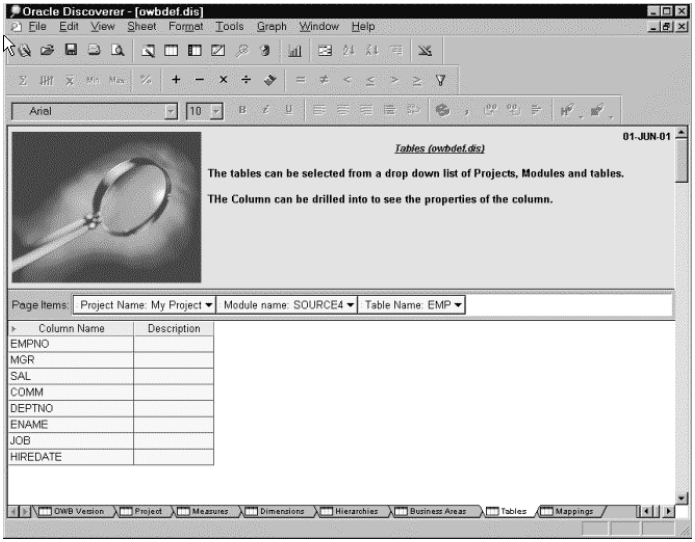
The **Dimensions** worksheet lists all the fact tables within the selected project or module, and the dimensions they reference.

Figure 11–36 Business Area Worksheet



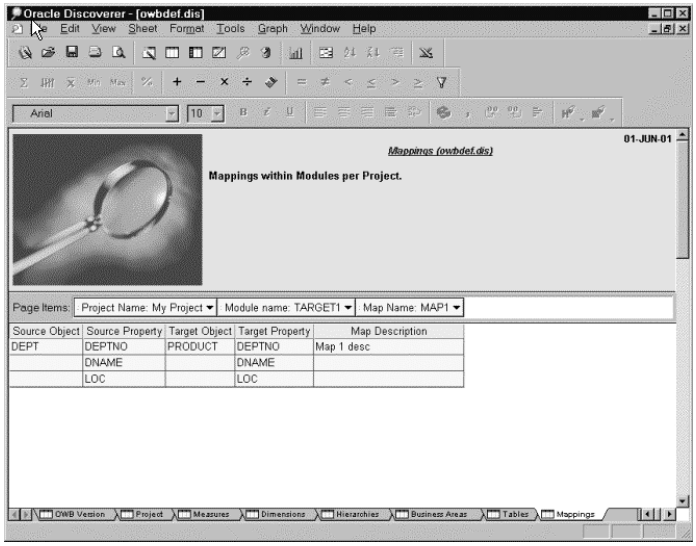
The **Business Area** worksheet lists all the Business Areas defined within the selected project. It also lists the objects defined within each business area.

Figure 11–37 Tables Worksheet



In the **Tables** worksheet, you can choose a table from a specific project and module. The worksheet then displays the columns defined within the selected table. The columns can be drilled on to view the column properties.

Figure 11–38 Mappings Worksheet

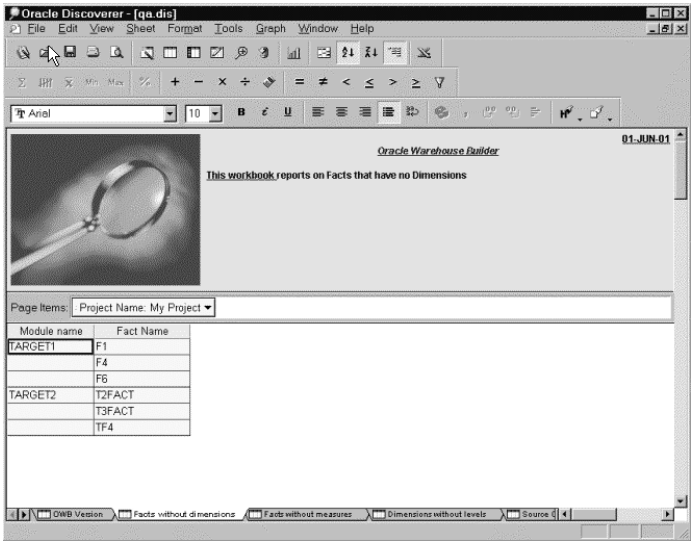


In the **mappings** worksheet, you can choose a mapping from a specific project and module. The worksheet then displays the source and target objects defined within that mapping.

QA Workbooks

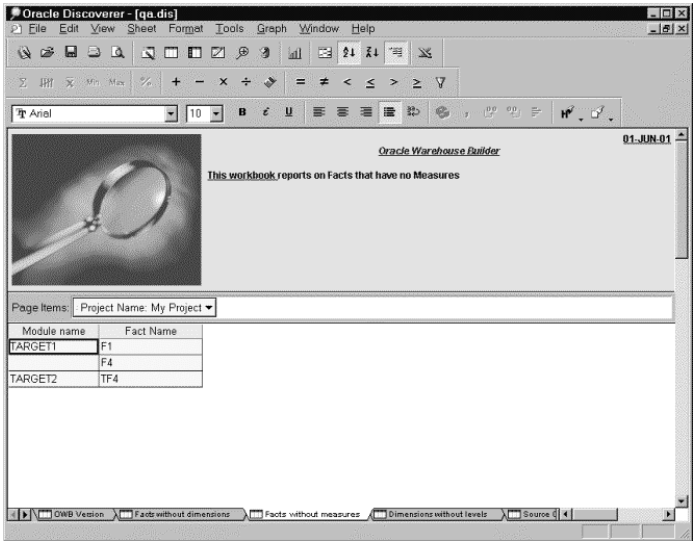
The QA Workbooks enable you to report on incomplete and invalid object definitions. These reports are useful in tracking down logical errors in your Warehouse Builder design. The QA workbook contains the following types of worksheets.

Figure 11–39 Facts without Dimensions Worksheet



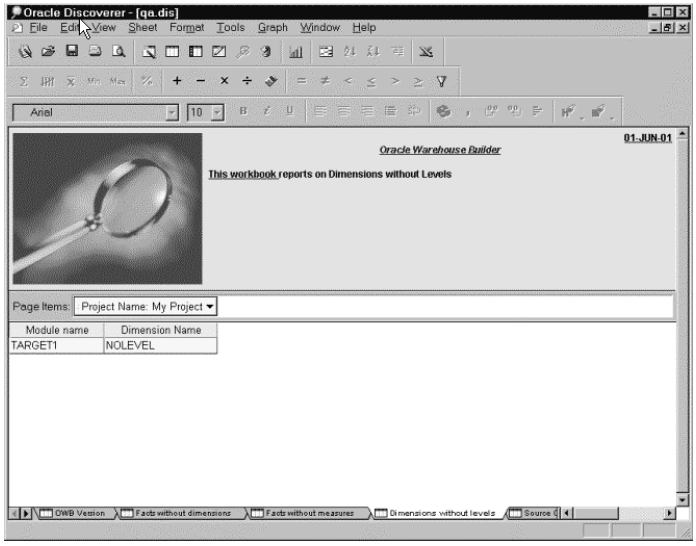
The **Facts without Dimensions** worksheet reports on erroneous fact tables, within the selected project, that do not reference any dimensions.

Figure 11–40 Facts without Measures Worksheet



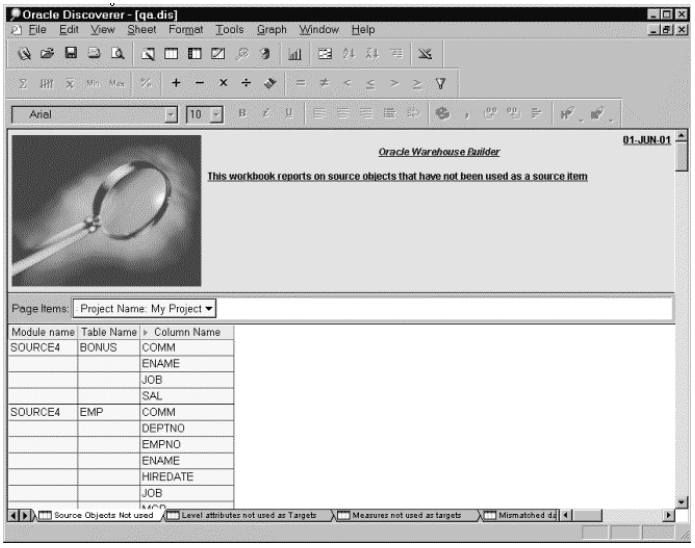
The **Facts without Measures** worksheet reports on fact tables, within the selected project, that do not contain defined measures.

Figure 11–41 Dimensions without Levels Worksheet



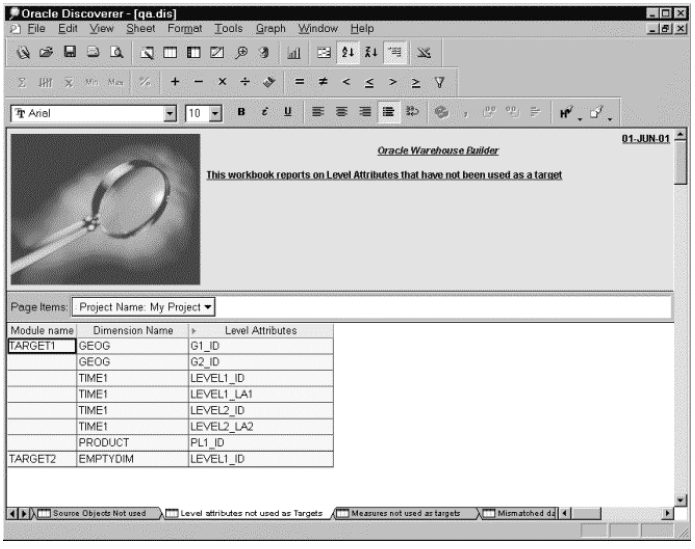
The **Dimensions without Levels** worksheet reports on the dimensions, within the selected project, that do not contain any defined levels.

Figure 11–42 Source Objects Not Used Worksheet



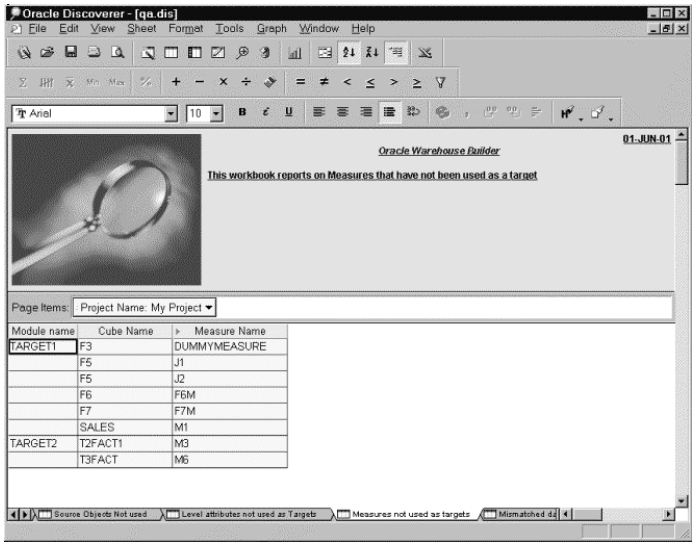
The **Source Objects Not Used** worksheet reports on the source objects that have not been used as source items within mappings in the selected project.

Figure 11–43 Level Attributes Not Used as Targets Worksheet



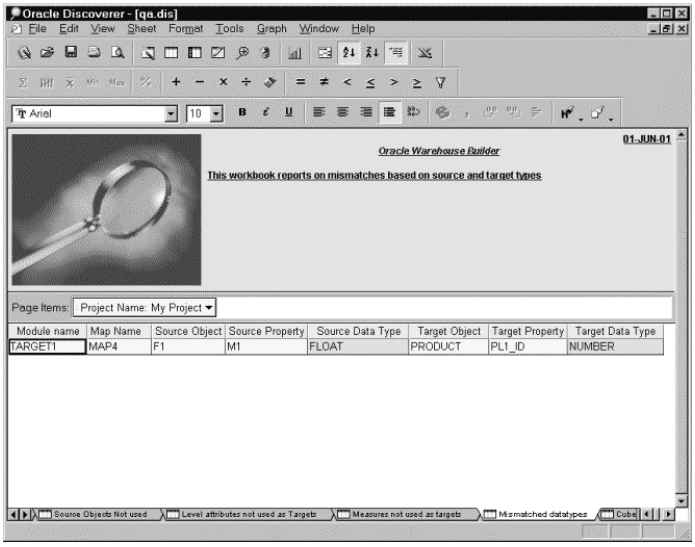
The **Level Attributes Not Used as Targets** worksheet reports on the level attributes that have not been used as a target within mappings in the selected project.

Figure 11–44 Measures Not Used as Targets Worksheet



The **Measures Not Used as Targets** worksheet reports on the measures that have not been used as a target in the selected project.

Figure 11–45 Mismatched Data Types Worksheet

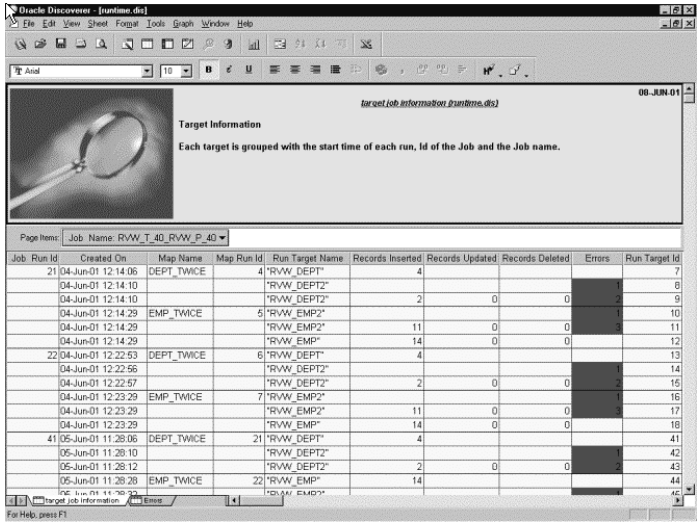


The **Mismatched Data Types** worksheet reports on mismatches based on the source and target types in the selected project.

Runtime Workbooks

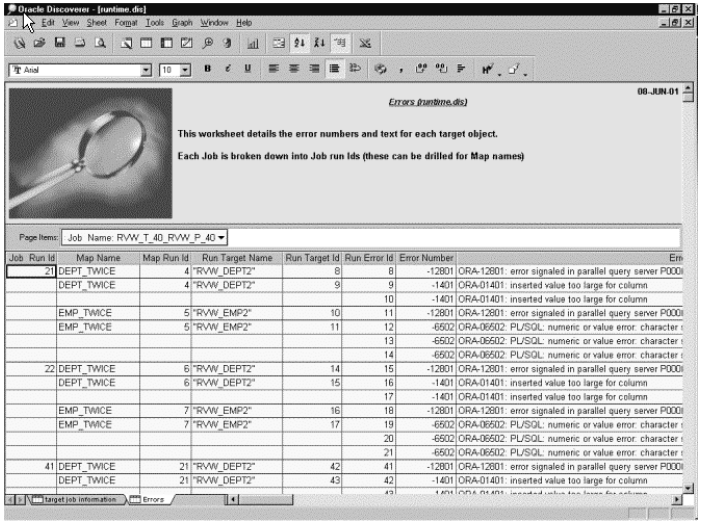
The Runtime workbooks enable you to report on runtime data and track down runtime errors. The following worksheets are available within the runtime workbooks.

Figure 11–46 Target Job Information Worksheet



The **Target Job Information** worksheet reports on the job runs for each job selected from the Page Items field. A job includes several Warehouse Builder mappings that can be run separately. This report lists information on the runs along with any errors that were encountered.

Figure 11–47 Errors Worksheet



The **Errors** worksheet displays the error numbers and error messages for each target object within a Warehouse Builder mapping within the selected job.

Improving Performance of Workbooks

You can use the Query prediction feature within Discoverer to measure the cost of running Discoverer Workbooks.

Since the Warehouse Builder design data resides in one physical table (CMPALLCLASSES) with numerous views, the default settings for the Discoverer Query prediction are not optimal for increasing performance. To increase performance, you can either disable query prediction by updating the registry, or ANALYZE the table to gather information as indicated in the Discoverer documentation.

Below is an extract from the Discoverer Administration documentation. For details refer to the *Discoverer Administration User's Guide.*

For Windows NT, Discoverer 4.1 registry settings are stored under:

\\HKEY_CURRENT_USER\Software\Oracle\Discoverer 4\Database

For Windows 2000, Discoverer 4.1 registry settings are stored under:

\\HKEY_CURRENT_USER\Software\Oracle\Webdisco 4\Default\Database

QPPEnable =1 (0 = false, 1 = true): This setting enables you to turn Query Prediction on or off. Uses query prediction / performance (QPP) if set to 1.

An alternative is to analyze the Warehouse Builder table to improve the efficiency of the query prediction.

SQL> analyze table <username.tablename> compute statistics for all columns;

SQL> analyze table <username.tablename> compute statistics;

For example, if the user name is Warehouse Builder, then the statement is:

analyze table OWB.CMPALLCLASSES compute statistics for all columns;

analyze table OWB.CMPALLCLASSES compute statistics;

This analyze statement only needs to be run occasionally.

12

Metadata Reporting Using Warehouse Builder Browser

Metadata reporting allows you to examine the data stored in your Warehouse Builder repository. The Warehouse Builder Browser integrates with Oracle Portal and makes it possible to add metadata reporting functionality to Oracle Portal. There are five Warehouse Builder portlets that you can add to customize your Warehouse Builder Browser. 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. You can access these reports from the Warehouse Builder client which opens Oracle Portal, or directly through Oracle Portal without using the client.

This chapter includes the following topics:

- Accessing Reports from the Warehouse Builder Client
- Accessing Reports from the Warehouse Builder Browser
- About Warehouse Builder Reports
- Using Warehouse Builder Browser
- Creating Custom Reports
- Using the Administration Pages

Accessing Reports from the Warehouse Builder Client

You can view a report on an item using the Warehouse Builder clients if Oracle Portal is installed. You must set up your Warehouse Builder Preferences to view reports.

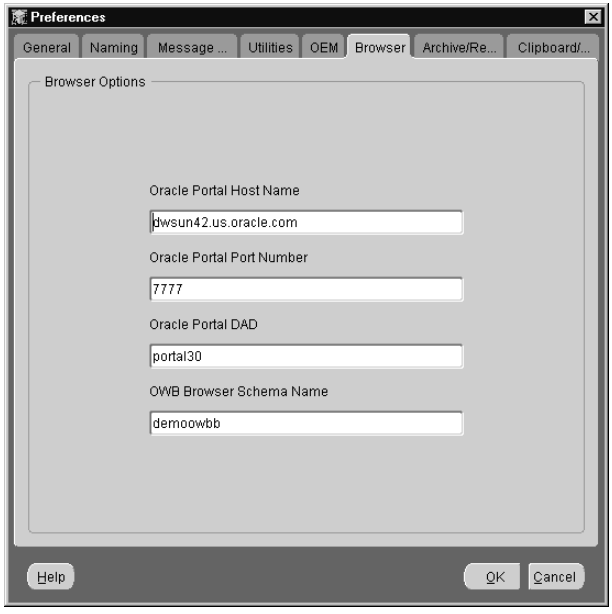
Setting up Browser Preferences

To set Browser Preferences:

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

   The Preferences window for the project displays.

Figure 12–1 Warehouse Builder Preferences



2. Click the **Browser** tab of the Preferences window.
3. Specify the following information:
   - Oracle Portal Host Name
   - Oracle Portal Port Number
   - Oracle Portal DAD

- Warehouse Builder Browser Schema Name

4. Click **OK**.

Opening the Reports

To open reports from within the Warehouse Builder client:

1. Open a project in Warehouse Builder and select an object to run a report on. You can select a module, or you can open the Module Editor and select any object in the module.
2. From the **View** menu, select **Reports**.

   The Oracle Portal site you defined in the Preferences window opens and displays the logon.
3. Enter your Internet Application Server (IAS) Single Sign-On user name and password the first time you run a report during this session.

   The Warehouse Builder Browser displays a list of reports available for the selected object.

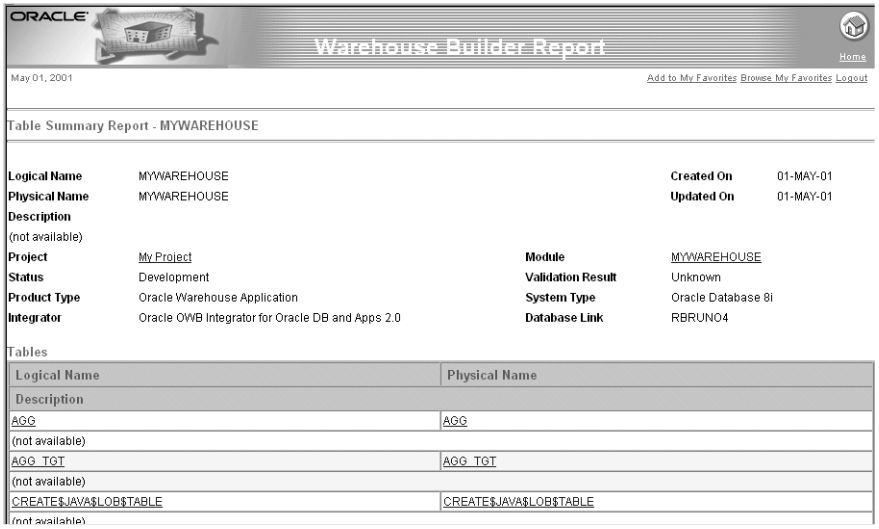
Figure 12–2 Available Warehouse Builder Reports

Name	Actions
Business Area Summary Report	view
Detailed Module Report	view
Dimension Summary Report	view
Fact Summary Report	view
File Summary Report	view
Function Library Summary Report	view
Materialized View Summary Report	view
Sequence Summary Report	view
Table Summary Report	view
Transform Map Summary Report	view
View Summary Report	view

4. Click the **view** link for a report.

   The report displays in a separate window. Figure 12–3 shows an example of the Table Summary Report.

Figure 12–3 Warehouse Builder Report



ORACLE

Warehouse Builder Report

Home

May 01, 2001

Add to My Favorites Browse My Favorites Logout

Table Summary Report - MYWAREHOUSE

Logical Name	MYWAREHOUSE	**Created On**	01-MAY-01
Physical Name	MYWAREHOUSE	**Updated On**	01-MAY-01
Description			
(not available)			
Project	My Project	**Module**	MYWAREHOUSE
Status	Development	**Validation Result**	Unknown
Product Type	Oracle Warehouse Application	**System Type**	Oracle Database 8i
Integrator	Oracle OWB Integrator for Oracle DB and Apps 2.0	**Database Link**	RBRUN04

Tables

Logical Name	Physical Name
Description	
AGG	AGG
(not available)	
AGG_TGT	AGG_TGT
(not available)	
CREATE$JAVA$LOB$TABLE	CREATE$JAVA$LOB$TABLE
(not available)	

To print the report, select the print operation from the browser menu or toolbar.

Accessing Reports from the Warehouse Builder Browser

You can view reports from the Warehouse Builder Browser without installing Warehouse Builder. You start by launching Oracle Portal. From there you can select the Warehouse Builder Browser portlets.

Starting the Warehouse Builder Browser

To start the Warehouse Builder Browser:

1. Open your Internet browser and set the URL to your Oracle Portal address.
2. Click **Login**.

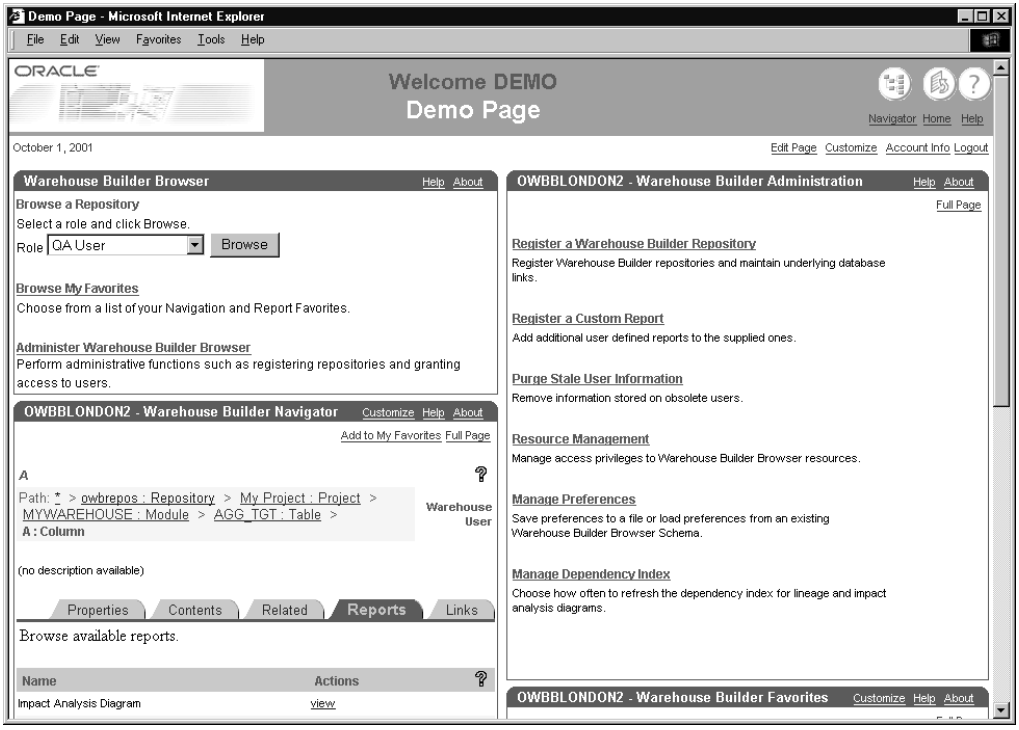
   The Warehouse Builder Browser uses a standard Oracle Portal login screen. This is not specific to the Warehouse Builder Browser.

Figure 12–4 Single Sign-On Page

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.

Your default Oracle Portal page displays. To display the Warehouse Builder Browser portlets, you must add them to your Oracle Portal home page. Use the Edit Page option to add portlets.

Figure 12–5 Oracle Portal Home Page with Warehouse Builder Portlets



Adding Portlets to the Warehouse Builder Browser

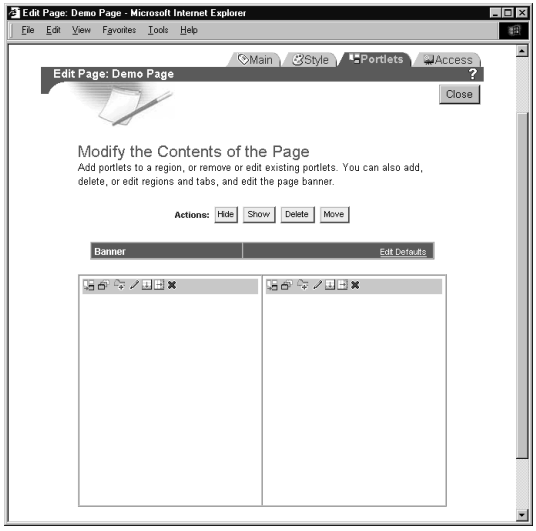
Warehouse Builder Browser portlets can be added to Oracle Portal after they have been installed on the machine running Oracle Portal. For more information about these portlets, see "Available Warehouse Builder Browser Portlets" on page 12-9. To install and configure the Warehouse Builder Browser Portlets, see the Warehouse Builder Installation and Configuration Guides. 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.

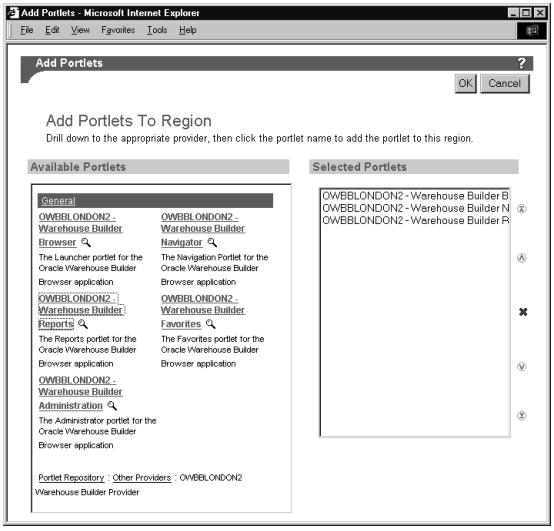
Figure 12–6 Oracle Portal Edit Page



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. See "Available Warehouse Builder Browser Portlets" on page 12-9 for more details.

Figure 12–7 Add Portlets Page



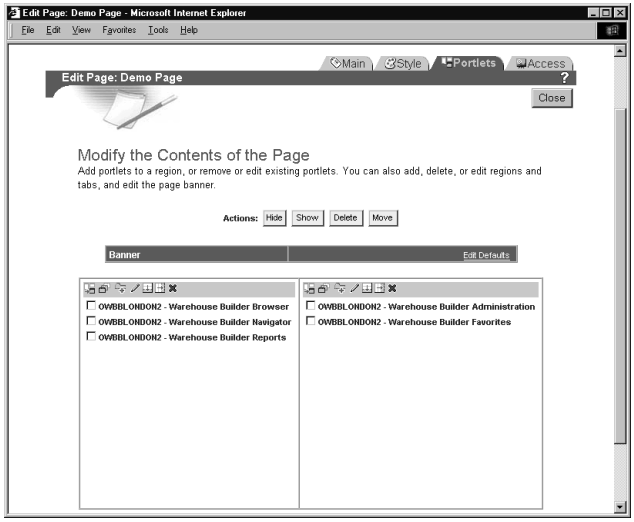
3. Select the portlets to add to the Oracle Portal home page.

   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.

Figure 12–8 Edit Page



Available Warehouse Builder Browser Portlets

You can add the following Warehouse Builder portlets to Oracle Portal. You can also add more than one of each portlet. For example, you can add two Reports portlets with each running a report on a different repository.

- Launcher Portlet
- Administration Portlet
- Favorites Portlet
- Navigator Portlet
- Reports Portlet

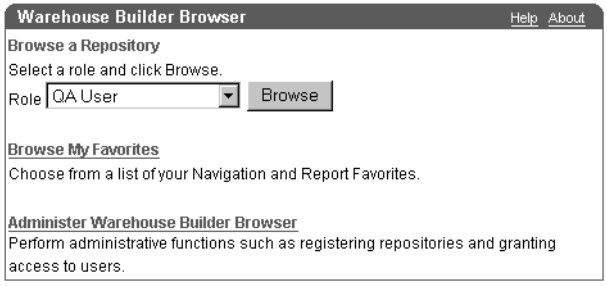
Launcher Portlet

The Launcher Portlet provides access to the following actions from your Oracle Portal home page:

- **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 12–9 Launcher Portlet

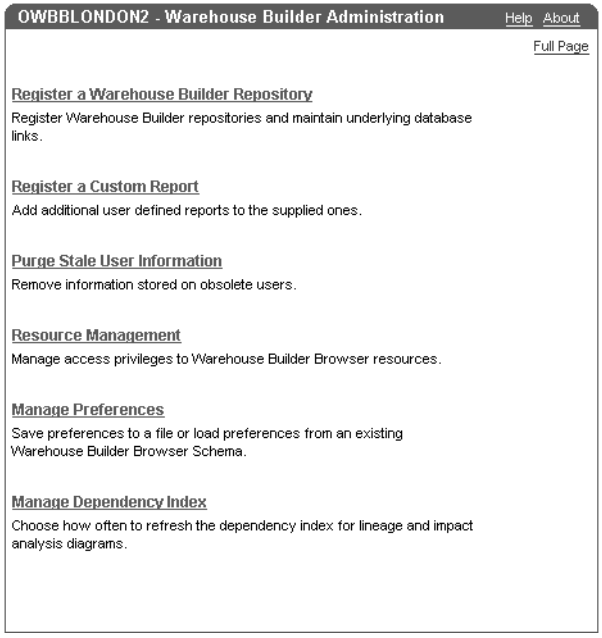


Administration Portlet

The Administration Portlet provides access to the following Administration features from your Oracle Portal home page:

- **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

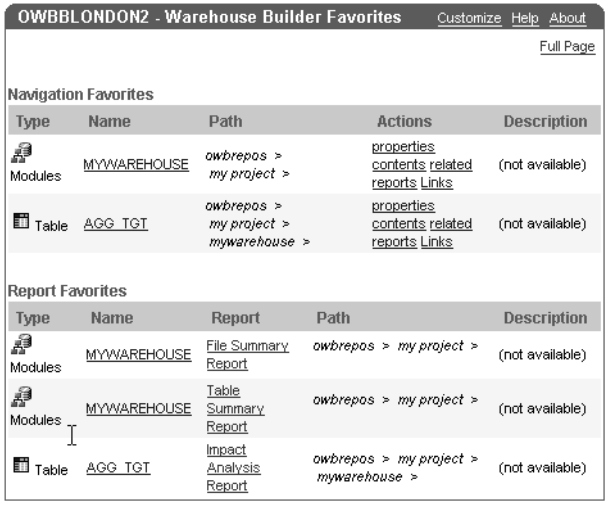
Figure 12–10 Administration Portlet



Favorites Portlet

The Favorites Portlet provides access to your favorite Navigator pages and Reports from your Oracle Portal home page. This portlet does not contain any default favorites. You 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 12–11 Favorites Portlet



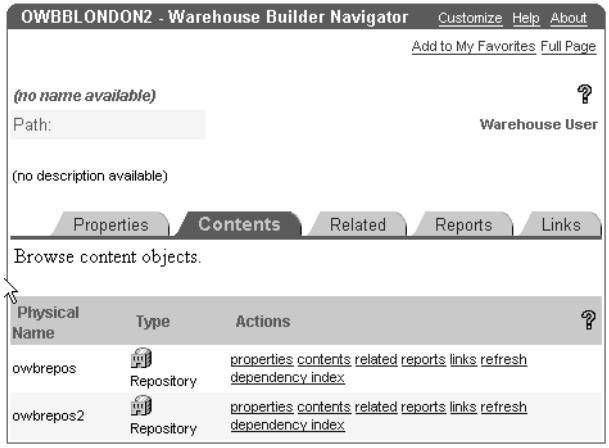
To view your favorite Navigator pages, select an item name under the column Name. You can also use the action links to view the properties, contents, related items, reports, or links associated with this item.

To view your favorite Reports, select a report name under the Report column. If you click on the item name under the column Name, the Navigator page for that item displays.

Navigator Portlet

The Navigator Portlet enables you to browse repositories accessible to the current user. 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.

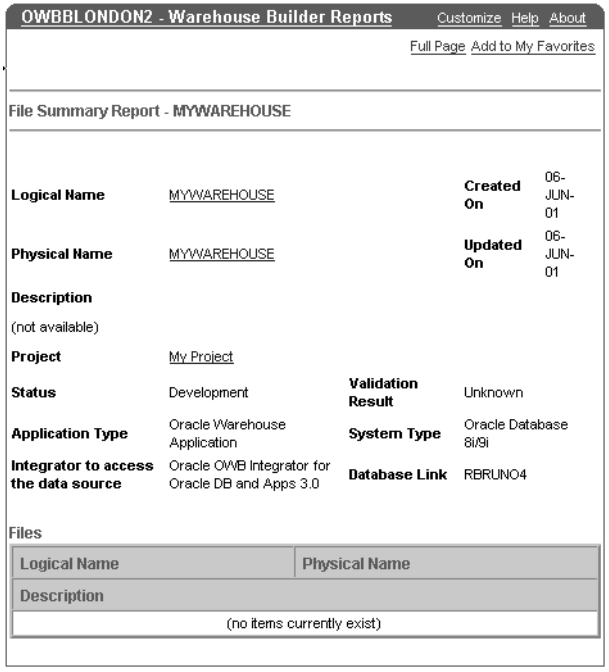
Figure 12–12 Navigator Portlet



Reports Portlet

The Reports Portlet displays a Warehouse Builder Report on your Oracle Portal home page. This portlet does not contain a default Report. In order to display a Report, add a report to your favorites and use the Customize link on this portlet to add it.

Figure 12–13 Reports Portlet



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.

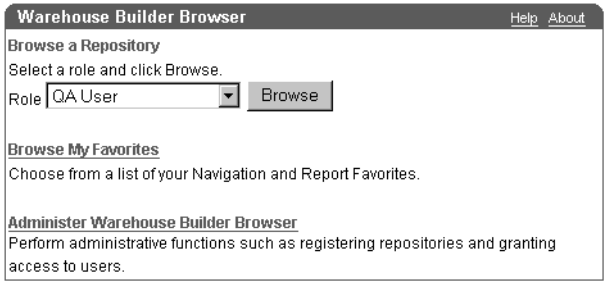
Using Warehouse Builder Browser to View Reports

To view Warehouse Builder Reports:

1. Select a Warehouse Builder Repository from the Repository drop-down list in the Warehouse Builder Browser Launcher portlet.

   The Repository list contains repositories that have been previously set up by your Warehouse Administrator using the Warehouse Builder Browser Administration pages.

Figure 12–14 Selecting a Repository



2. Select the role you want to use:
   - Warehouse Engineer
   - QA User
   - Warehouse User

   The role determines the available reports. Roles are set up by your Warehouse Administrator in the Warehouse Builder Browser Administration pages. For more information, see "Understanding Roles" on page 12-56.
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 to display the available reports for that item.
5. Select the **view** link to view the report.

About Warehouse Builder Reports

When you first install or upgrade Warehouse Builder, you can use the Warehouse Builder Browser Assistant to set up your Warehouse Builder Browser. You can either run this wizard during your installation process, or defer it to a later time. See the Warehouse Builder Installation Guide for instructions on how to use this wizard.

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.

Note: In order to use the Warehouse Builder Browser you must have access to an Oracle Portal site with the Warehouse Builder Browser portlet installed. For more information about installing the Warehouse Builder Browser portlet, see the Warehouse Builder Installation Guide.

Available Warehouse Builder Reports

Warehouse Builder provides a set of standard metadata reports. Use these pre-built reports to examine your metadata. The following types of reports are available:

- Summary Reports
- Detailed Reports
- Implementation Reports
- Lineage and Impact Analysis Reports
- Lineage and Impact Analysis Diagrams

Summary Reports

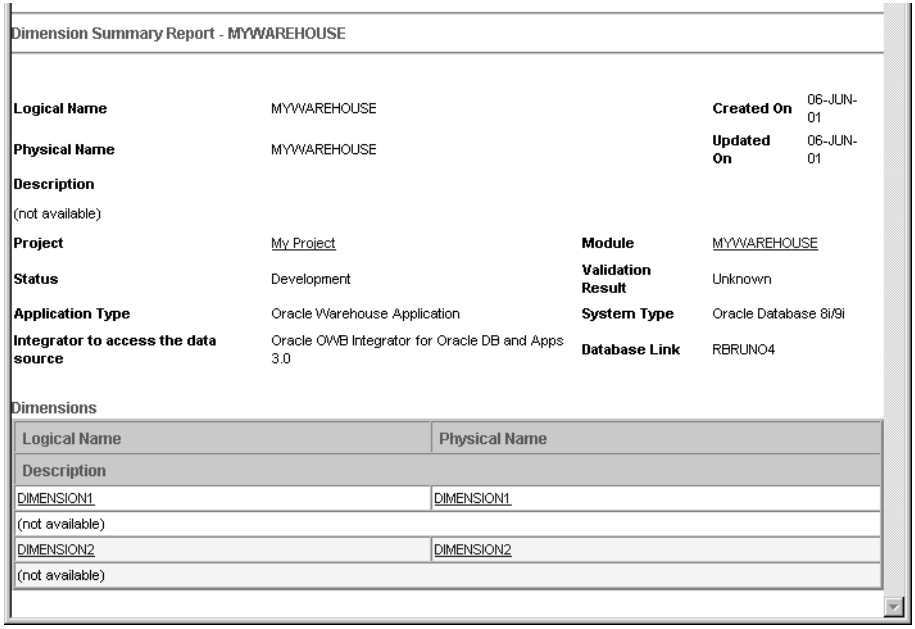
Summary Reports are run against a module. Each Summary Report presents a list of items contained within the module. The type of items displayed is determined by the Summary Report you select. For example, a Table Summary Report lists all tables in the module. A Materialized View Summary Report lists 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.

Available Summary Reports for a Module include:

Business Area Summary Report	Dimension Summary Report
Fact Summary Report	File Summary Report
Function Library Summary Report	Materialized View Summary Report
Sequence Summary Report	Table Summary Report
Transform Map Summary Report	View Summary Report

Figure 12–15 show a section from a Dimension Summary Report.

Figure 12–15 Dimension Summary Report



Dimension Summary Report - MYWAREHOUSE

Logical Name	MYWAREHOUSE	**Created On**	06-JUN-01
Physical Name	MYWAREHOUSE	**Updated On**	06-JUN-01
Description			
(not available)			
Project	My Project	**Module**	MYWAREHOUSE
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**	RBRUNO4

Dimensions

Logical Name	Physical Name
Description	
DIMENSION1	DIMENSION1
(not available)	
DIMENSION2	DIMENSION2
(not available)	

Detailed Reports

Detailed reports provide comprehensive information about an item. For example, a Detailed Table Report will list information about the table columns, keys, foreign keys and physical configuration parameters. Table 12–1 list the Detailed Reports and their contents.

Table 12–1 Detailed Reports

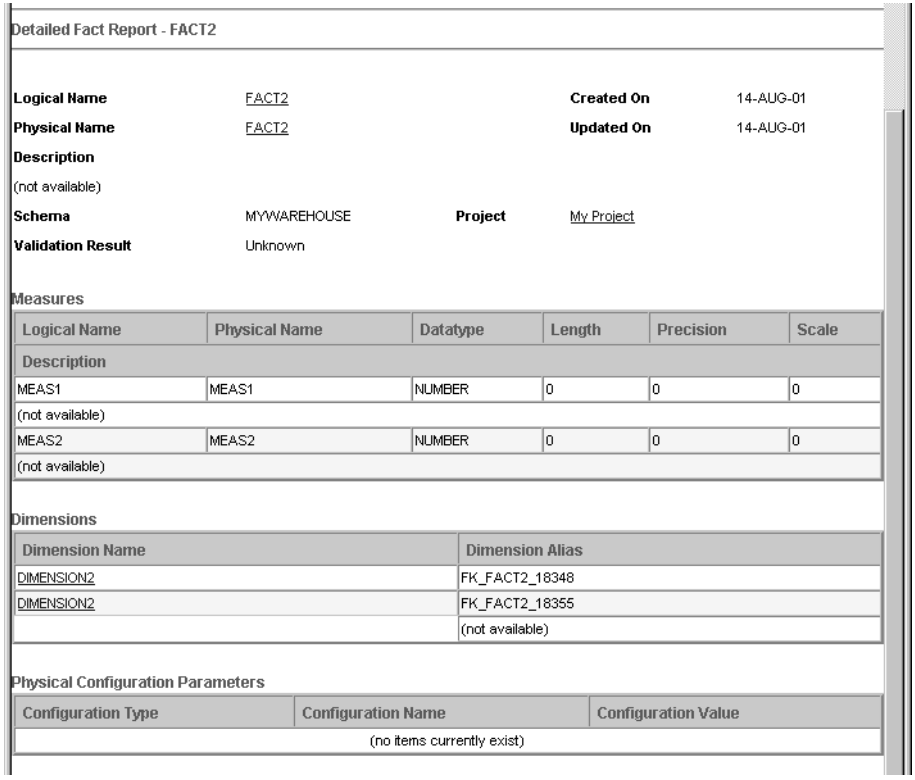
Report	Contents
Detailed Installation Report	Projects
Detailed Project Report	Modules and Business Areas
Detailed Module Report	Facts, Dimensions, Function Libraries, Materialized Views, Sequences, Tables, Views, Transform Maps, Files and physical configuration parameters
Detailed Dimension Report	Hierarchies, Levels, Level Attributes and physical configuration parameters

Table 12–1 Detailed Reports (Cont.)

Report	Contents
Detailed Fact Report	Measures, Dimensions and physical configuration parameters
Detailed Table Report	Columns, Keys, Foreign Keys and physical configuration parameters
Detailed View Report	Columns, Keys, Foreign Keys and physical configuration parameters
Detailed Materialized View Report	Columns, Keys, Foreign Keys, and physical configuration parameters
Detailed Sequence Report	Columns and physical configuration parameters
Detailed File Report	Records and physical configuration parameters
Detailed Record Report	Fields
Detailed Business Area Report	Items in the Business Area
Detailed Function Library Report	Functions and physical configuration parameters
Detailed Function Report	Parameters and Function Implementations
Detailed Transform Map Report	Object Uses, Map Components, Item Maps and physical configuration parameters

Figure 12–16 shows a section from a Detailed Fact report.

Figure 12–16 Detailed Fact Report



Detailed Fact Report - FACT2

Logical Name FACT2 **Created On** 14-AUG-01

Physical Name FACT2 **Updated On** 14-AUG-01

Description

(not available)

Schema MYWAREHOUSE **Project** My Project

Validation Result Unknown

Measures

Logical Name	Physical Name	Datatype	Length	Precision	Scale
Description					
MEAS1	MEAS1	NUMBER	0	0	0
(not available)					
MEAS2	MEAS2	NUMBER	0	0	0
(not available)					

Dimensions

Dimension Name	Dimension Alias
DIMENSION2	FK_FACT2_18348
DIMENSION2	FK_FACT2_18355
	(not available)

Physical Configuration Parameters

Configuration Type	Configuration Name	Configuration Value
(no items currently exist)		

Implementation Reports

Implementation Reports can be run on Dimensions and Facts. They provide information on how physical objects are used to implement logical objects. Table 12–2 lists the available Implementation Reports.

Table 12–2 Implementation Reports

Report	Content
Fact Implementation Report	Tables that implement the Fact, Columns that implement the Measures, Foreign Keys that implement the Fact to Dimension Use

Table 12–2 Implementation Reports (Cont.)

Report	Content
Dimension Implementation Report	Levels and the tables that implement them, Level Attributes and the Columns that implement them

Lineage and Impact Analysis Reports

Lineage and Impact Analysis Reports are available for Facts, Dimensions, Materialized Views, Tables, Views, and Records. These reports should not be confused with the Lineage and Impact Analysis Diagrams.

Impact Analysis Reports Impact Analysis Reports list all items belonging to the subject of the report that are used as a source of a mapping. The name of the mapping and the name of the item that it is mapped to is also displayed. The report gives a one step impact analysis for all items related to the selected item.

For example, if you list all the columns in a table that is used as a source in any maps, use this report.

Lineage Reports Lineage Reports are similar to Impact Analysis Reports. They list items that are the targets of a map.

Figure 12–17 shows a section from an Impact Analysis report.

Figure 12–17 Impact Analysis Report

Table Impact Analysis Report - ITEM_TABLE1

Logical Name	ITEM_TABLE1	**Created On**	06-JUN-01
Physical Name	ITEM_TABLE1	**Updated On**	06-JUN-01
Description			
(not available)			
Schema	MYWAREHOUSE	**Project**	My Project
Validation Result	Unknown		

Impact Analysis Dependency

Column IT1_C1

Dependencies

Source Dependency	Type	Mapping
My Project.MYWAREHOUSE.ITEM_TABLE2.IT2_C1	Column	ITEM_TABLETABLETEST

Column IT1_C2

Dependencies

Source Dependency	Type	Mapping
My Project.MYWAREHOUSE.ITEM_TABLE2.IT2_C2	Column	ITEM_TABLESPLITTABLETEST
My Project.MYWAREHOUSE.ITEM_TABLE3.IT3_C2	Column	ITEM_TABLESPLITTABLETEST

Column IT1_C3

Dependencies

Source Dependency	Type	Mapping
My Project.MYWAREHOUSE.ITEM_TABLE3.IT3_C3	Column	ITEM_TABLEJOINTABLETEST

Lineage and Impact Analysis Diagrams

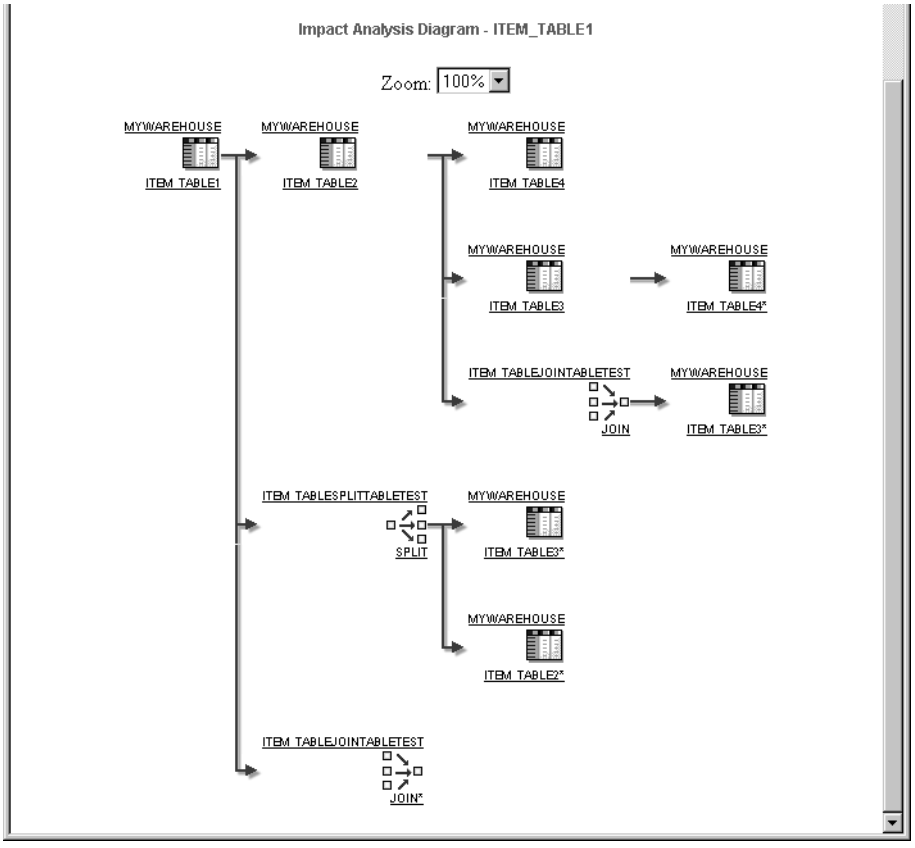
A Lineage Diagram graphically displays all the objects and transformations that are used to make up the subject of the Diagram. Lineage can be performed at either the object level or the item level. At the Object Level, the diagram can contain Tables, Views, Materialized Views, Dimensions, Facts, Records, 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 12–18 shows an example of an Impact Analysis Diagram.

Figure 12–18 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. For more information, see "Managing the Dependency Index" on page 12-60.

Using Warehouse Builder Browser

Use Warehouse Builder Browser to browse metadata reports and search for specific information by using the various functions of the pages including:

- Navigating the Warehouse Builder Browser,
- Browsing Favorites, and
- Customizing Your Favorites Page

Navigating the Warehouse Builder Browser

You can navigate the Warehouse Builder Browser in many ways. The main type of page is the Navigator page that includes the following parts:

- General Page Information
- Current Item Information
- Tabbed sections containing the Properties Tab, Contents Tab, Related Tab, Reports Tab, and Links Tab

Figure 12–19 Warehouse Builder Navigator

General Page Information

The top of every Warehouse Builder Browser page defines the type of page you are looking at and has links to other common pages.

Figure 12–20 Navigator Header

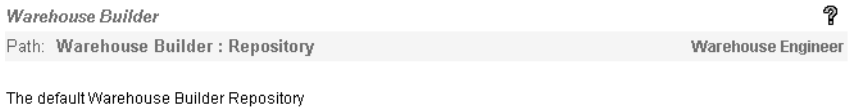
The header contains the following:

- **Title:** The title defines the type of page you are looking at. For example, the header in Figure 12–20 shows a Warehouse Builder Navigation page. There are also Warehouse Builder Administration and Favorites pages.
- **Home Link:** Select this link to go to your default home page.
- **Date:** The date shows you when the page is displayed. This is typically the current date. If you print a report, this can be helpful to keep track of when the report was run.
- **Add to My Favorites:** Select this link to add the current page to your Favorites pages. Use this to bookmark important reports.
- **Browse My Favorites:** Select this link to view the page containing your favorite Navigation pages and Reports that you previously bookmarked.
- **Customize:** Select this link to go to the Customize Warehouse Builder Navigation Pages page and customize your page display.
- **Logout:** Select this link to log out of the Warehouse Builder Browser and Oracle Portal.

Current Item Information

The center of the Navigator contains the current item definition and provides information to identify the item and its type.

Figure 12–21 Item Definition



The current item information section contains the following:

- **Item Name:** This is the name of the item that you have navigated to. In Figure 12–21, this is the Warehouse Builder Repository.
- **Help/Question Mark:** The Help link opens online help for the Navigation Page.
- **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. In Figure 12–21, the path is Path: Warehouse Builder: Repository, and the Warehouse Builder Repository is the current item. You can click an item in the path to make that the current item.
- **Role:** The role that you are logged in as is shown. In Figure 12–21, it is Warehouse Engineer.

Properties Tab

The Properties tab displays property name-value pairs for the current item. The property names are different depending on the type of the current item.

Figure 12–22 Properties Tab

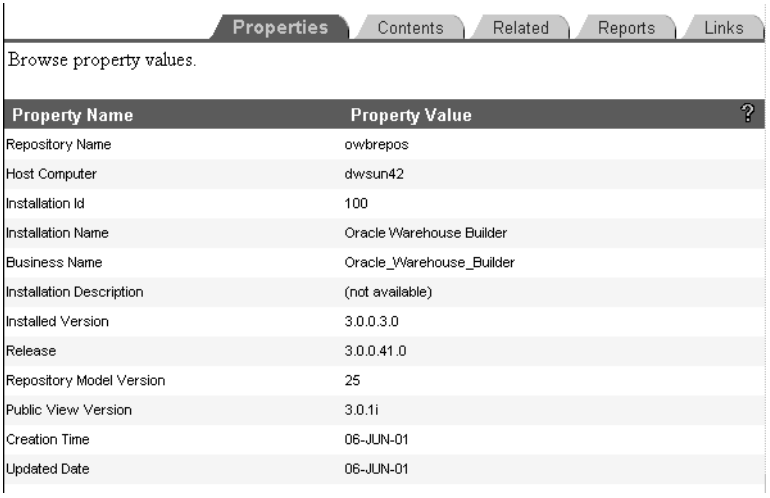


Table 12–3 Property Column Values

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 Other Custom Reports" on page 12-40.
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 12–23, the current item is the Warehouse Builder repository and the list of items that the repository contains. Table 12–4 lists the columns in the Contents tab and brief descriptions of their values.

Figure 12–23 Contents Tab

Table 12–4 Contents 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.
Actions	A set of links that go to the tab named on the link for the current item. ■ **Properties:** Selecting this makes the item on that row the current item and goes to the Properties tab for that item. ■ **Contents:** Selecting this makes the item on that row the current item and goes to the Contents tab for that item. ■ **Related:** Selecting this makes the item on that row the current item and goes to the Related tab for that item. ■ **Reports:** Selecting this makes the item on that row the current item and goes to the Reports tab for that item. ■ **Links:** Selecting this makes the item on that row the current item and goes to the Links tab for that item.

Related Tab

The Related tab allows you to view relationships between items. 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 12–24 shows the foreign key associations with a table. Table 12–5 lists the columns in the Related tab and brief descriptions of their values.

Figure 12–24 Related Tab

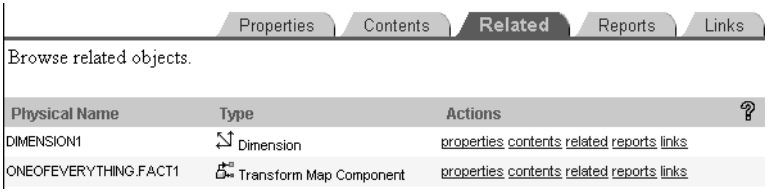


Table 12–5 Related Column Values

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:** Selecting this makes the item on that row the current item and goes to the Properties tab for that item. ■ **Contents:** Selecting this makes the item on that row the current item and goes to the Contents tab for that item. ■ **Related:** Selecting this makes the item on that row the current item and goes to the Related tab for that item. ■ **Reports:** Selecting this makes the item on that row the current item and goes to the Reports tab for that item. ■ **Links:** Selecting this makes the item on that row the current item and goes to the Links tab for that item.

Reports Tab

The Reports tab shows you the available reports for the current item. Each role has its own set of reports available for each item type. A Warehouse User sees a different set of reports than a Warehouse Engineer. Table 12–6 lists the columns in the Reports tab with brief descriptions of their values.

Figure 12–25 Reports Tab

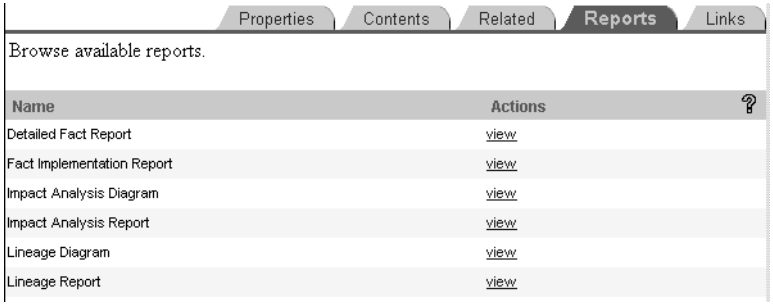


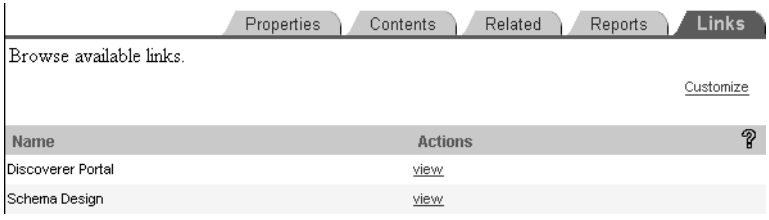
Table 12–6 Reports Column Values

Column Name	Value
Name	The name of the report.
Actions	**View:** Displays the report.

Links Tab

The Links tab allows you to associate other sources of information with an object or type of object. 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 12–26 Links Tab



Use the **Customize** link on the Links tab to:

- Edit Links
- Delete Links
- Add Links

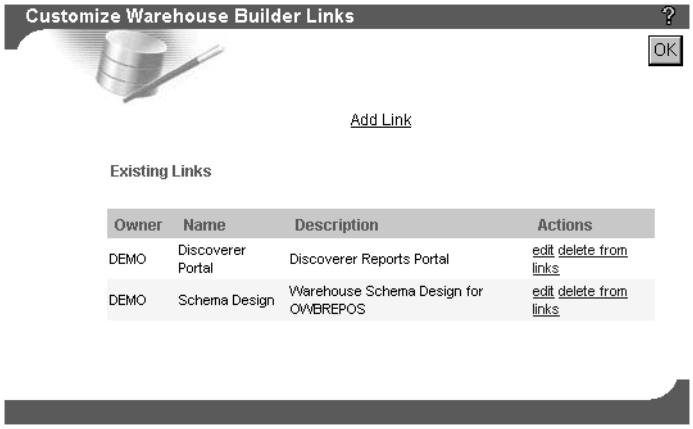
Note: You can only edit or delete links you created.

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.

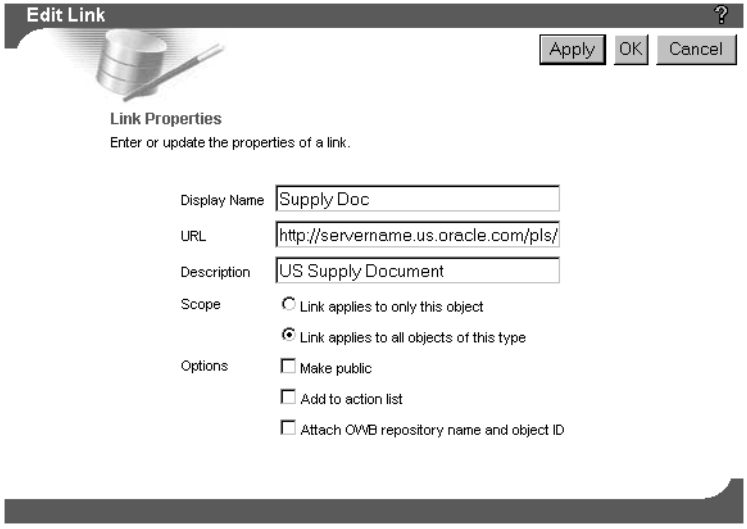
Figure 12–27 Customize Warehouse Builder Links Page



3. Select **edit** for the link you want to edit.

The Edit Link page displays the current link settings.

Figure 12–28 Edit Link Page



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:

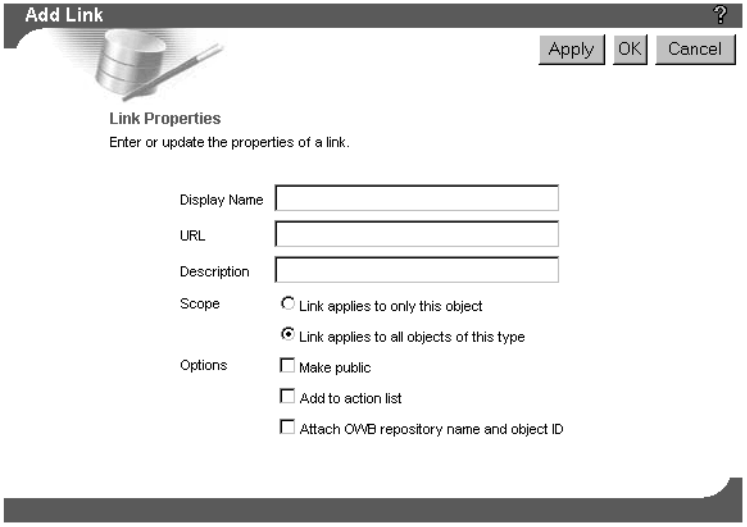
1. Navigate to the object, or object type that you want the new link associated with and select the Links tab.
2. From the Links tab, select **Customize.**

The Customize Warehouse Builder Links page displays the Existing Links.

3. Select **Add Link.**

   The Add Link page displays.

Figure 12–29 Add Link Page



4. Specify the link properties using Table 12–7and click **OK**.

Table 12–7 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 on the file and copy the URL location. Then return to the Add Link page and paste the URL into the URL field.
Description	Type a description for the link.

Table 12–7 Link Properties (Cont.)

Field	Description
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 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.

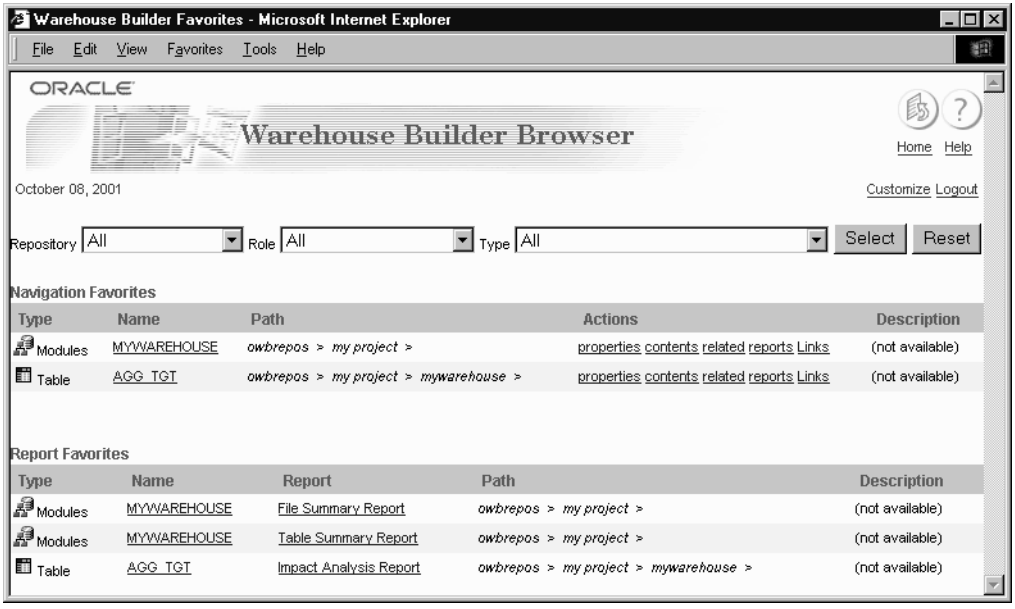
Browsing Favorites

Selecting the Browse My Favorites link goes to 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 consists of the following:

- General Page Information
- Filter
- Navigation Favorites
- Reports Favorites

Figure 12–30 Warehouse Builder Favorites Page



General Page Information

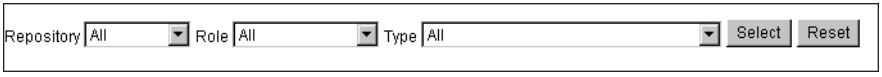
The top of every Warehouse Builder Browser page has a header that defines the type of page you are looking at, and has links to your default home page, your Favorites customization page, and log out.

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 12-36.

Filter

You can narrow down 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 12–31 Favorites Filter

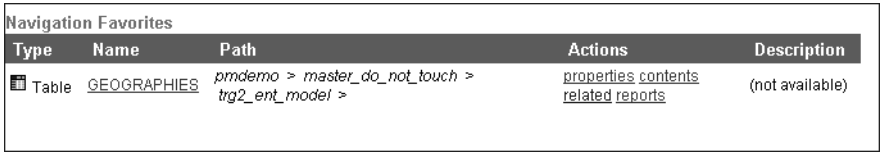


Navigation Favorites

This list contains items added from the Navigation pages. The table contains these columns:

- **Type:** The type of the item with an icon representing the type.
- **Name:** The name of the item. Selecting the name goes to the Navigation content page of 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:** You can add a description to a favorite in the Favorites Customize page. It is displayed here.

Figure 12–32 Navigation Favorites



Reports Favorites

This list contains items added from the Report pages. The table contains these columns:

- **Type:** The type of the item together with an icon representing the type.
- **Name:** The name of the item. Selecting the name will take you to the Navigator page with this item as the current item.
- **Report:** The name of the Report. Selecting the report name goes to the report.
- **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 Options page.
- **Description:** You can add a description to a favorite in the Favorites Customize page. It is displayed here.

Figure 12–33 Reports Favorites

Report Favorites

Type	Name	Report	Path	Description
InformationSystem	MODULE1	Detailed Information System Report	*agscott > my project >*	(not available)
Table	GEOGRAPHIES	Detailed Table Report	*pmdemo > master_do_not_touch > trg2_ent_model >*	(not available)

Customizing Your Favorites Page

To customize your favorites page, click the **Customize** link in the Warehouse Builder Favorites header. Table 12–6 lists the available actions and brief descriptions.

Figure 12–34 Customize Favorites Page

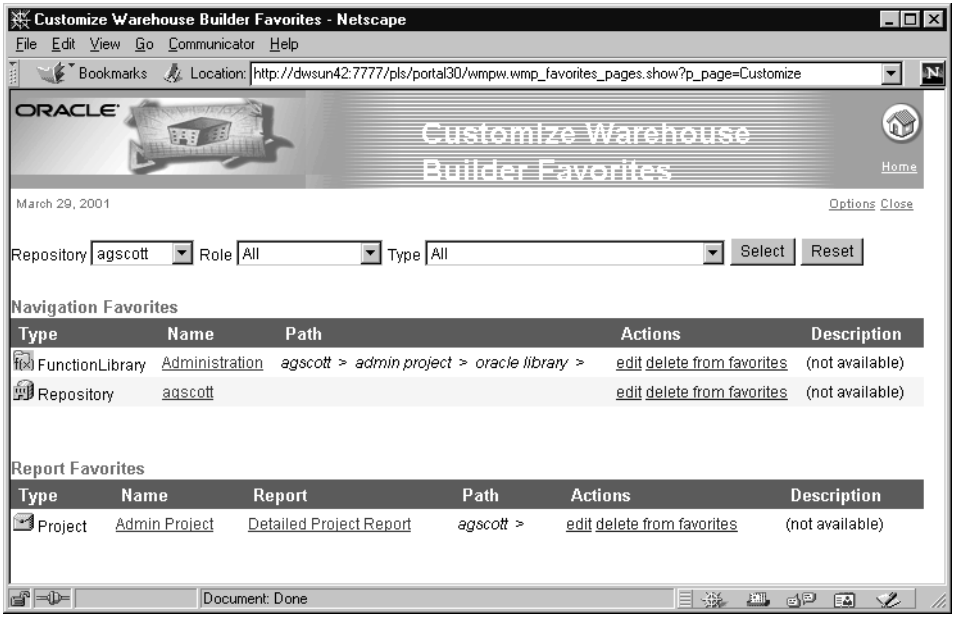
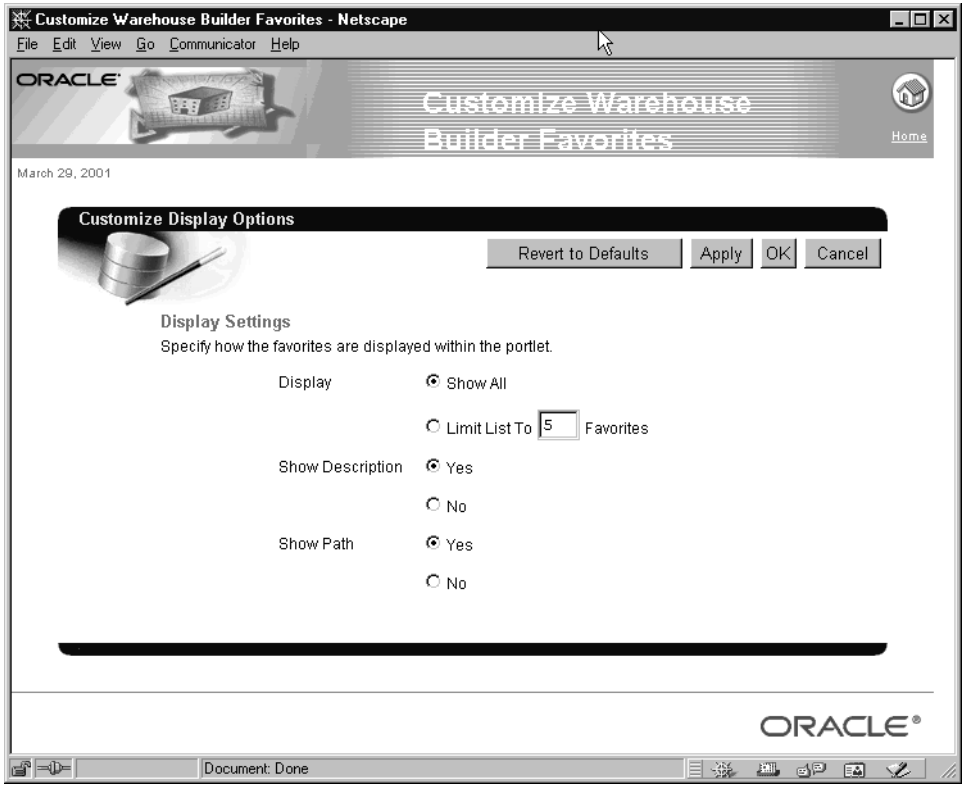


Table 12–8 Customization Actions

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. It does not remove the item from the Warehouse Builder Repository.

Selecting the Options link on the Favorites Customize page goes to the Customize Display Options page. Use this page to change the layout of your favorites on the Favorites page.

Figure 12–35 Favorites Customize Display Options Page



This page contains the following options:

- **Revert to Defaults:** Returns the display settings to the install defaults.
- **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 All:** Shows all the added entries in the Navigation list and the Reports list.
- **Limit List to n Favorites:** 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 / Hide Descriptions:** Yes, shows the description column. No, hides the column.
- **Show / Hide Path:** Yes, shows the path column. No, hides the column.

Creating Custom Reports

You can create custom reports on your metadata using the Warehouse Builder Public Views. 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 reports.

To create a custom report that can be viewed in the Warehouse Builder Browser, follow the steps in these sections:

- Creating a Custom Report
- Registering a Custom Report
- Adding a Custom Report to a Role
- Viewing a Custom Report

Creating a Custom Report

To create a custom report:

1. Log on to Oracle Portal and select the Database Objects tab from the Navigator page.

2. Find the schema where the Warehouse Builder Browser is loaded, and edit the schema details to ensure that the Application Schema check box is selected.
3. Select the Applications tab, and click on the **Create New Application** link.
4. Create a new application and click **OK.**
5. Select the application you just created and select the **Create New Report** link.
6. Select the **Report from SQL Query** link from the page that displays next.
7. Type the report name and display name and click **Next.**
8. 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 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**.

Registering a Custom Report

To register a custom report with the Warehouse Builder Browser:

1. From the launcher portlet, click the **Administer Warehouse Builder Browser** link, and then select the **Register a Custom Report** link.
2. Select the type and repository from the drop-down lists.
3. Enter a display name for the 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 **OK**.

   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.

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.

Creating Other Custom Reports

Use the tables below to determine which Public Views to report from. Table 12–9 lists repository object types and their associated public views. Table 12–10 lists runtime object types and their associated views. For detailed information on each view, including what objects they contain, see Appendix G, "OWB Public View Tables Management".

Table 12–9 Repository Object Types and Associated Views

Object Type	Views
Folder	ALL_IV_INFORMATION_SYSTEMS
	ALL_IV_INSTALLATIONS
	ALL_IV_PROJECTS, ALL_IV_SCHEMAS
	ALL_IV_FILES
Project	ALL_IV_PROJECTS
Module	ALL_IV_INFORMATION_SYSTEMS
Dimension	ALL_IV_DIMENSIONS
Level	ALL_IV_DIM_LEVELS
Level Attribute	ALL_IV_DIM_LEVEL_ATTRIBUTES
Hierarchies	ALL_IV_DIM_HIERARCHIES
	ALL_IV_DIM_HIERARCHY_LEVELS
Foreign Keys	ALL_IV_DIM_IMPLS
	ALL_IV_KEYS
	ALL_IV_FOREIGN_KEYS
Unique Keys	ALL_IV_DIM_IMPLS
	ALL_IV_KEYS
	ALL_IV_FOREIGN_KEYS
Key Column	ALL_IV_KEY_COLUMN_USES
	ALL_IV_COLUMNS
Configurations	ALL_IV_OBJECT_CONFIGURATIONS
Fact (Cube)	ALL_IV_CUBES
Fact Attribute	ALL_IV_CUBE_MEASURES
Foreign Keys	ALL_IV_CUBE_IMPLS
	ALL_IV_KEYS
	ALL_IV_FOREIGN_KEYS
	ALL_IV_CUBE_MEASURE_DIM_USES
Unique Keys	ALL_IV_DIM_IMPLS
	ALL_IV_KEYS
	ALL_IV_FOREIGN_KEYS

Table 12–9 Repository Object Types and Associated Views (Cont.)

Object Type	Views
Key Column	ALL_IV_KEY_COLUMN_USES
	ALL_IV_COLUMNS
Configurations	ALL_IV_OBJECT_CONFIGURATIONS
Table	ALL_IV_TABLES
Columns	ALL_IV_COLUMNS
Foreign Keys	ALL_IV_FOREIGN_KEYS
Unique Keys	ALL_IV_KEYS
Key Column	ALL_IV_KEY_COLUMN_USES
	ALL_IV_COLUMNS
Configurations	ALL_IV_OBJECT_CONFIGURATIONS
Indexes	
Partitions	
View	ALL_IV_VIEWS
Columns	ALL_IV_COLUMNS
Foreign Keys	ALL_IV_FOREIGN_KEYS
Unique Keys	ALL_IV_KEYS
Key Column	ALL_IV_KEY_COLUMN_USES
	ALL_IV_COLUMNS
Configurations	ALL_IV_OBJECT_CONFIGURATIONS
Materialized View	ALL_IV_VIEWS
Columns	ALL_IV_COLUMNS
Foreign Keys	ALL_IV_FOREIGN_KEYS
Unique Keys	ALL_IV_KEYS
Key Column	ALL_IV_KEY_COLUMN_USES
	ALL_IV_COLUMNS
Configurations	ALL_IV_OBJECT_CONFIGURATIONS
Indexes	
Partitions	
Sequence	ALL_IV_SEQUENCES

Table 12–9 Repository Object Types and Associated Views (Cont.)

Object Type	Views
Column	ALL_IV_COLUMNS
Configurations	ALL_IV_OBJECT_CONFIGURATIONS
File	ALL_IV_FILES
Record	ALL_IV_RECORDS
Field	ALL_IV_FIELDS
Unique Key	ALL_IV_KEYS
Configurations	ALL_IV_OBJECT_CONFIGURATIONS
Business Tree	ALL_IV_CLASSIFICATIONS
	ALL_IV_CLASSIFICATION_ITEMS
Business Area Shortcut	ALL_IV_CLASSIFICATIONS
	ALL_IV_CLASSIFICATION_ITEMS
Transform Category	ALL_IV_FUNCTION_LIBRARIES
Transforms	ALL_IV_FUNCTIONS
Transform	ALL_IV_FUNCTIONS
Parameters	ALL_IV_FUNCTION_PARAMETERS
Implementation	ALL_IV_FUNCTION_IMPLS
Map	ALL_IV_XFORM_MAPS
Stage	ALL_IV_XFORM_MAP_COMPONENTS
	ALL_IV_XFORM_MAP_PROPERTIES
	ALL_IV_XFORM_MAP_PARAMETERS
Data Source (DS)	ALL_IV_TABLES
	ALL_IV_VIEWS
	ALL_IV_MATERIALIZED_VIEWS
	ALL_IV_RECORDS, ALL_IV_SEQUENCES
DS Parameter Groups	ALL_IV_XFORM_MAP_COMPONENTS
DS Parameters	ALL_IV_XFORM_MAP_PARAMETERS

Table 12–9 Repository Object Types and Associated Views (Cont.)

Object Type	Views
Data Target (DT)	ALL_IV_TABLES
	ALL_IV_VIEWS
	ALL_IV_MATERIALIZED_VIEWS
	ALL_IV_RECORDS, ALL_IV_SEQUENCES
DT Parameter Groups	ALL_IV_XFORM_MAP_COMPONENTS
Operator	ALL_IV_XFORM_MAP_COMPONENTS
Operator Parameter Groups	ALL_IV_XFORM_MAP_PARAMETERS
Variable	ALL_IV_XFORM_MAP_COMPONENTS
Parameter	ALL_IV_XFORM_MAP_PARAMETERS
Configurations	ALL_IV_OBJECT_CONFIGURATIONS
Configuration Usage	ALL_IV_OBJECT_CONFIGURATIONS
Configuration Parameters Usage	ALL_IV_OBJECT_CONFIGURATIONS
Configuration Parameter Values	ALL_IV_OBJECT_CONFIGURATIONS
Line	ALL_IV_OBJECT_CONFIGURATIONS
Generated Object	ALL_IV_OBJECT_CONFIGURATIONS
Impact Analysis	ALL_IV_ALL_OBJECTS
	ALL_IV_IMPACT_DEPENDENTS
	ALL_IV_LINEAGE_DEPENDENTS

Table 12–10 lists the Public Runtime Views.

Table 12–10 Runtime Objects and Associated Views

Object Type	Views
Process Runs	ALL_IV_PROCESS
	ALL_IV_PROCESS_RUN
Audit Detail	ALL_IV_MAP_RUN
	ALL_IV_RUN_TARGET
Audit Error Log	ALL_IV_RUN_ERROR

Using the Administration Pages

From the Oracle Portal Home page, select the **Administer Warehouse Builder Browser** link to access the Warehouse Builder Administration pages.

Figure 12–36 Warehouse Builder Administration Portlet

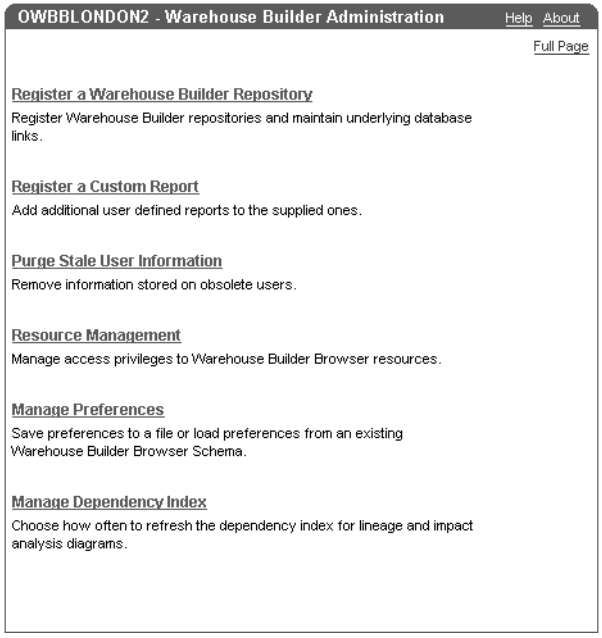
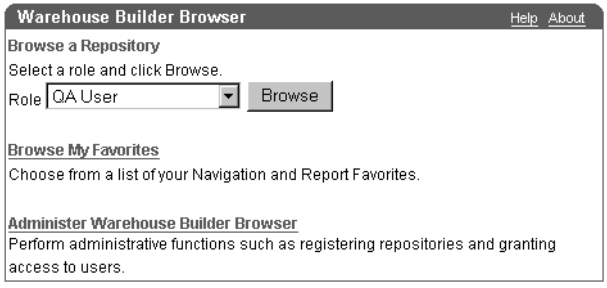


Figure 12–37 Warehouse Builder Launcher Portlet



The Warehouse Builder Administration pages can only be accessed if your Oracle Portal user name has full administrator privileges.

The Administration page contains the following administration actions:

- **Register an OWB Repository,** for more details, see "Registering a Warehouse Builder Repository" on page 12-46.
- **Register a Custom Report,** for more details, see "Registering a Custom Report" on page 12-52.
- **Purge Stale User Information,** for more details, see "Purging Stale User Information" on page 12-53.
- **Resource Management,** for more details, see "Resource Management" on page 12-54.
- **Manage Preferences,** for more details, see "Managing Preferences" on page 12-57.
- **Manage Dependency Index,** for more details, see "Managing the Dependency Index" on page 12-60.

Registering a Warehouse Builder Repository

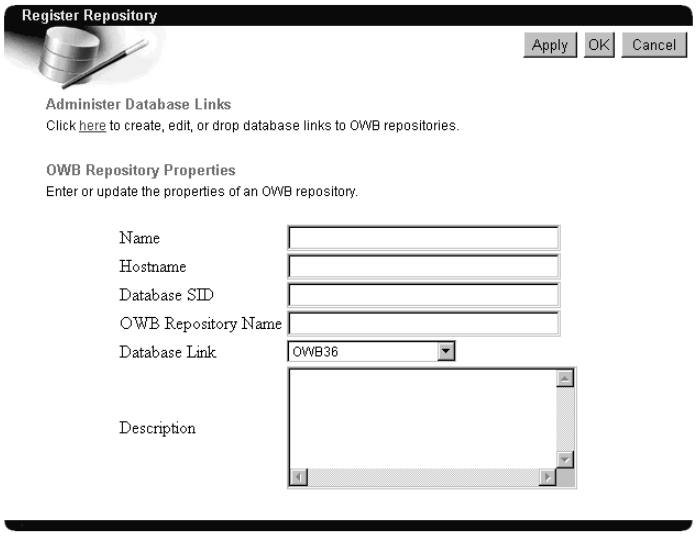
Before you can report on metadata in a repository, that repository must be registered with the Warehouse Builder Browser. Registrations creates the link to the repository.

> **Note:** When you register a Warehouse Builder repository, the database link to that repository must already exist. If it does not, you must create one. Skip to Administering Database Links on page 12-48 to create a database link before continuing.

To register a Warehouse Builder Repository:

1. Click **Register an OWB Repository** on the Warehouse Builder Administration home page. This goes to the Register Repository page.

Figure 12–38 Register Repository Page



2. Specify the Warehouse Builder repository properties. Table 12–11 lists the properties.

Table 12–11 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.
Host Name	The name of the server machine for the repository. **Note:** If you used the default host name LocalHost when you ran the Repository Assistant, enter LocalHost here even if Oracle Portal is running from another machine.
Database SID	The SID of the repository database.
OWB Repository Name	The name of the repository schema.
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.

Table 12–11 Warehouse Builder Repository Properties (Cont.)

Field	Description
Description	The user-defined descriptive text. This appears in the navigation pages for the repository.

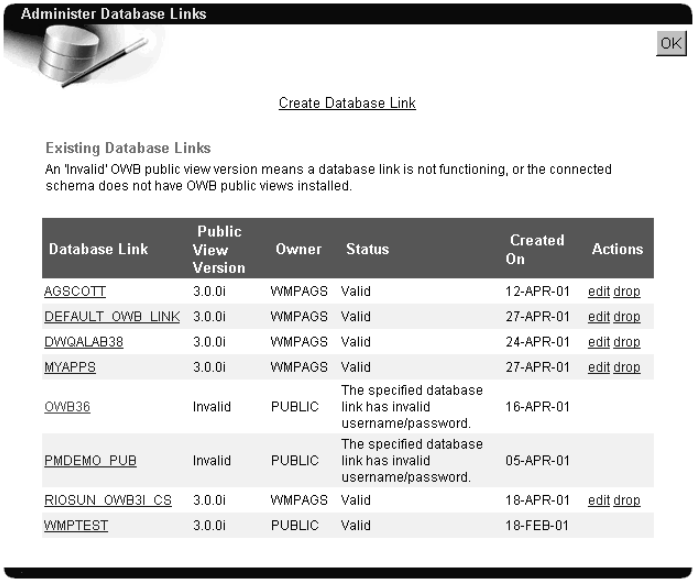
3. Click **Apply** to register the repository.
4. Click **OK.**

   The repository displays in the Warehouse Builder Administration home page.

Administering Database Links

You can create, view, edit, or drop database links to Warehouse Builder repositories from the Administer Database Links page. This section describes each of these actions.

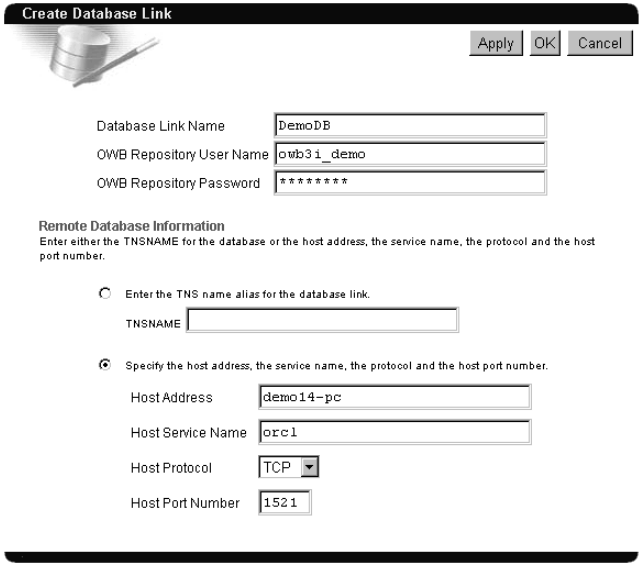
Figure 12–39 Administer Database Links Page



To create a database link:

1. Select **Create Database Link** from the Administer Database Links page.

Figure 12–40 Create Database Links Page



2. Specify the database link name.
3. Specify the Warehouse Builder repository user name and password.
4. Specify the remote database information. You can either:
   - Enter the TNS name for the database.
   - Enter the host address, service name, protocol, and host port number.
5. Click **Apply** to connect the link.
6. Click **OK.**

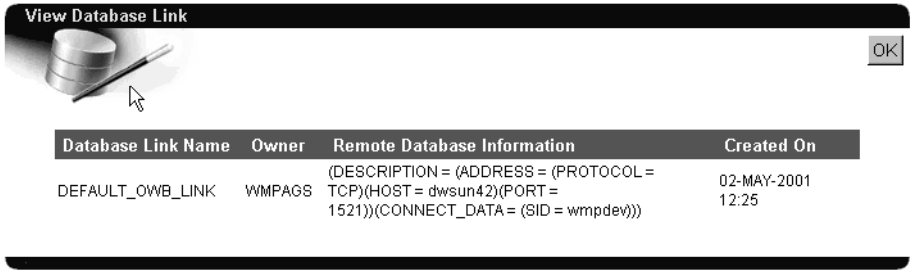
   The new link displays on the Administer Database Links page.

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.

Figure 12–41 View Database Link Page



2. Click **OK.**

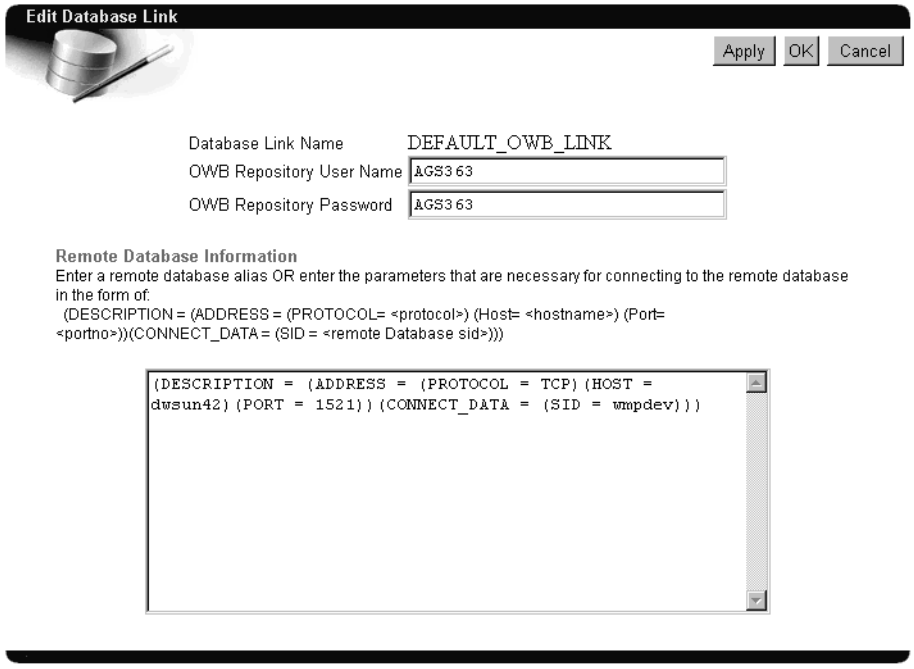
   The browser returns to the Administer Database Links page.

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 under the Actions column.

   The Edit Database Link page displays.

Figure 12–42 Edit Database Link Page



2. Edit the database link and click **Apply**.
3. Click **OK.**

To drop a database link:

> **Note:** Dropping a database link deletes it permanently. You must create a new link to use it again.

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.

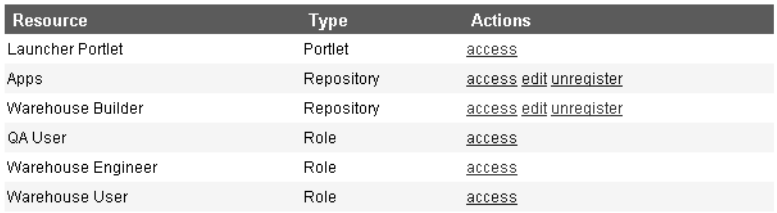
Unregistering a Repository

To unregister a Repository:

1. Select the Administer Warehouse Builder Browser link from the Browser home page.

   The Warehouse Builder Administration page displays with the registered repositories listed in the table at the bottom of the page.

Figure 12–43 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

2. Select the repository to unregister and click on 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.

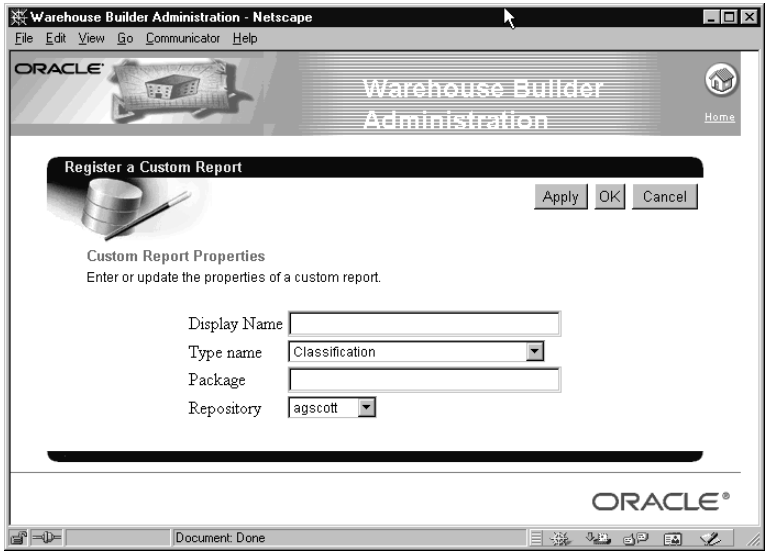
Registering a Custom Report

A custom report is an application component created using 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 12-38.

To register a custom report:

1. Select **Register a Custom Report** from the Warehouse Builder Administration page. The Register a Custom Report page displays.

Figure 12–44 Register Custom Report Page



2. Enter the report properties. Table 12–12 lists the custom report properties.

Table 12–12 Custom Report Properties

Field	Description
Display Name	This is the name of the report which is seen in the reports list page.
Type Name	This is the name of the data type reported on by this report.
Package	This is the full name of the PL/SQL package which implements this report.
Repository	This is the name of the repository containing target objects for this report.

3. Click **Apply** or **OK** to complete the registration.

Purging Stale User Information

If you add users through the Warehouse Builder Browser and remove them through Oracle Portal, the user information remains in the Access pages of the Warehouse

Builder Browser. Use the **Purge Stale User Information** link to remove these users and groups.

> **Note:** When you click **Purge Stale User Information,** the information is purged. No confirmation message displays.

Resource Management

The Resource Management page contains a table listing all resources that have been registered in the Warehouse Builder Browser. The following sections describe how to modify these resources.

Figure 12–45 Resource Management

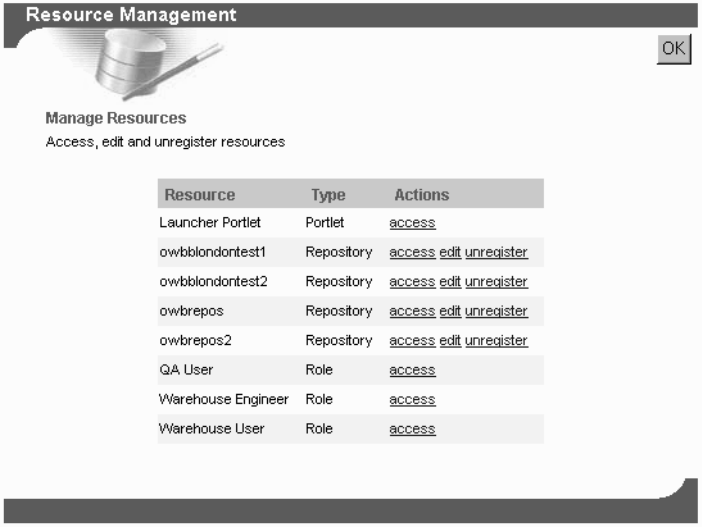


Table 12–13 lists the types of resources.

Table 12–13 Browser Resources

Resource Type	Description
Portlet	A portlet provided by the browser system. The Launcher portlet provides access to repository navigation, reporting, favorites, and administration.

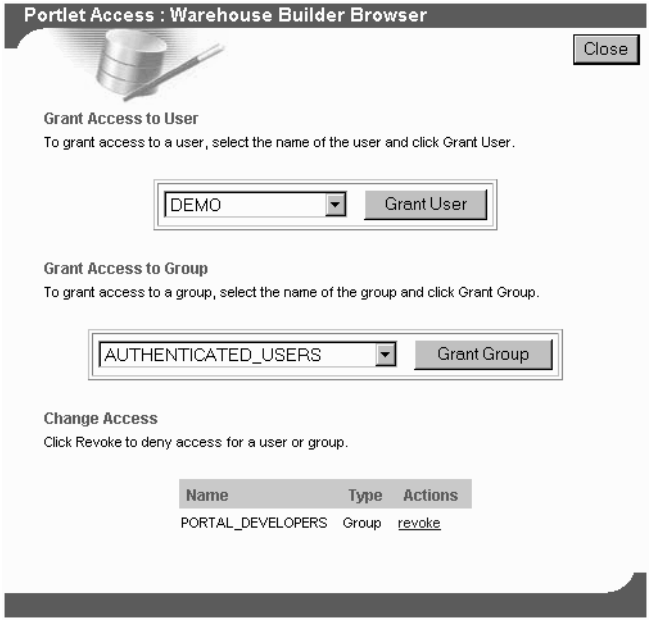
Table 12–13 Browser Resources (Cont.)

Resource Type	Description
Repository	Warehouse Builder Repository that can be browsed using the Warehouse Builder Browser.
Role	A view of the data in a Warehouse Builder Repository. For more information, see "Understanding Roles" on page 12-56.
Custom Report	A custom report on a Warehouse Builder repository object.

Managing Portlet Access

Use the Portlet Access pages to grant and revoke access to Warehouse Builder portlets. The users and groups that have access are listed under Change Access.

Figure 12–46 Portlet Access



From the Portlet Access page, you can:

- Grant an access right to a Warehouse Builder Browser user by selecting the name of the user from the drop-down list and clicking **Grant User**.

- Grant an access right to a Warehouse Builder Browser 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.

Managing Repositories

The Resource Management page lists all registered repositories. The actions listed next to the repository are described in Table 12–14.

Table 12–14 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 unregistration, the repository can no longer be browsed using the browser system. You must re-register the repository if you want to browse it again.

Managing Custom Reports

The Resource Management page lists all registered custom reports. The actions listed next to the repository are described in Table 12–15.

Table 12–15 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 unregistration, the report 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.

Understanding Roles

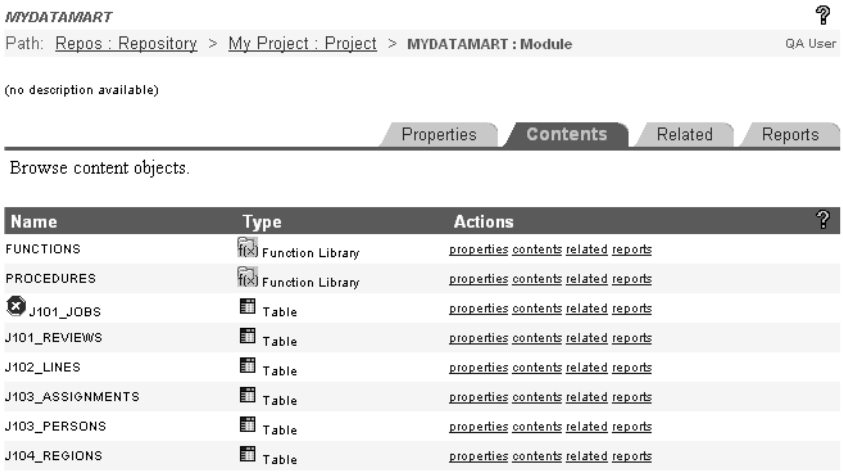
In order to browse the Warehouse Builder Repository, choose one of the pre-defined roles:

- **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 or register repositories, 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.

Figure 12–47 Validation Error



Managing Preferences

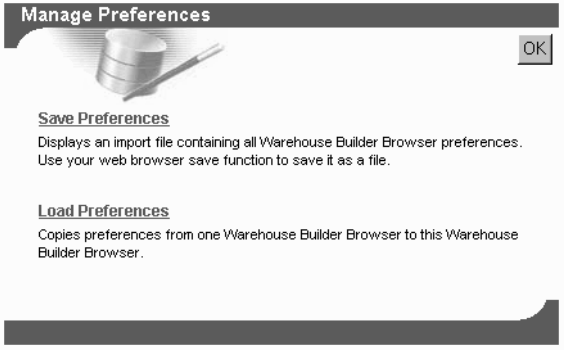
Use the Manage Preferences page to save and load Warehouse Builder Browser preferences. This lets you retain your preferences when you upgrade to a new version of Warehouse Builder Browser. You can copy schema preferences across schemas.

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 12–48 Manage Preferences Page



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.

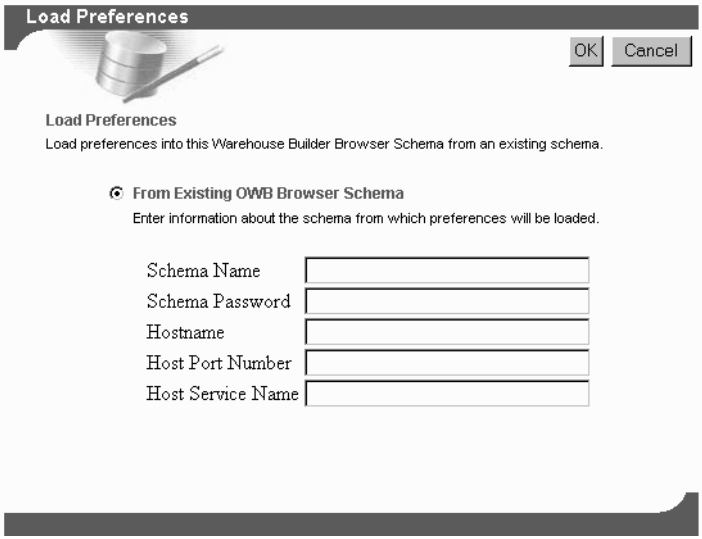
This file can be loaded into Warehouse Builder Browser using a tool such as SQL* Plus.

Loading Preferences

To load preferences from an existing schema:

1. Select **Load Preferences** from the Manage Preferences page.

Figure 12–49 Load Preferences Page



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.

   Note: 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.

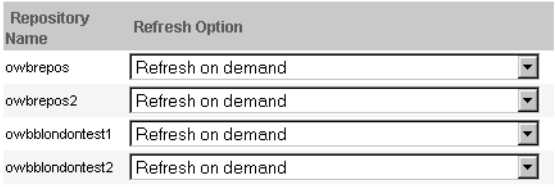
Setting the Refresh Options

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.

Figure 12–50 Setting Refresh Options



2. Choose one of the options from the drop-down list and click **OK**. Table 12–16 describes each option.

Table 12–16 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.
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.

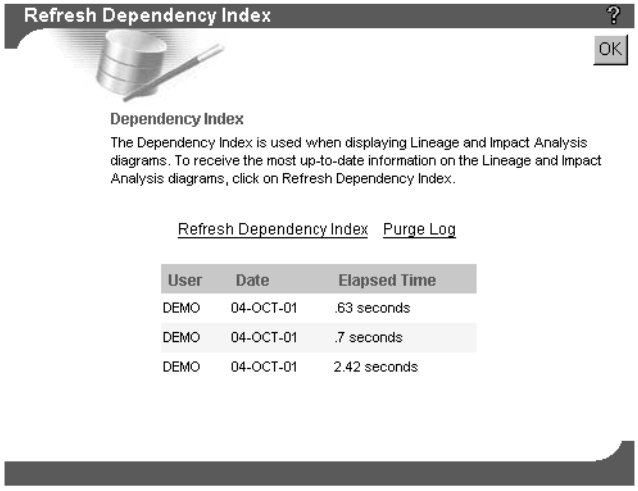
   The Contents tab that displays lists the available repositories.

Figure 12–51 Refreshing the 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. The elapsed time helps you determine how long the operation will take.

Figure 12–52 Dependency Index Refresh Log



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.

A

Keyboard Shortcuts

This appendix lists the keyboard options for accessing Warehouse Builder commands.

Keyboard Command Access

The following tables list Warehouse Builder keystroke commands. For conventions that are supported by most applications designed for Windows 95 and Windows NT, refer to the Microsoft Windows Keyboard Guide.

General Windows 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 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 next object above or below.
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 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 Menu Commands and Access Keys

Keyboard Sequence	Description
Alt	Activates the menu bar of the active window. The leftmost menu name is selected.
Alt + any printing character	Activates the menu with the underlined character (access key) on the main menu bar.

Table A–4 Menu Commands and Access Keys (Cont.)

Keyboard Sequence	Description
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).

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 Auto-Mapping Mnemonic Key Assignments

Keyboard Sequence	Description
Alt + o	OK.
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.

Table A–5 Auto-Mapping Mnemonic Key Assignments (Cont.)

Keyboard Sequence	Description
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 Oracle9*i* Warehouse Builder.

Reserved Words

Table B–1 lists reserved words. Do not use these words as physical object names within a Warehouse Builder Project.

Table B–1 Oracle Warehouse Builder Reserved Words

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

Table B–1 Oracle Warehouse Builder Reserved Words (Cont.)

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	SCHEMA	SELECT
SEPARATE	SESSION	SET	
SIZE	SMALLINT	SPACE	SQL
SQLCODE	SQLERRM	START	STATEMENT
STDDEV	SUBTYPE	SUCCESSFUL	SUM
SYNONYM	SYSDATE	TABAUTH	TABLE
TABLES	TASK	TERMINATE	TIME
THEN	TO	TRIGGER	TRUE
TYPE	UID	UNION	UNIQUE

Table B–1 Oracle Warehouse Builder Reserved Words (Cont.)

UPDATE	USE	USER	VALIDATE
VALUES	VARCHAR	VARCHAR2	VARIANCE
VIEW	VIEWS	WHEN	WHENEVER
WHERE	WHILE	WITH	WORK
WRITE	XOR		

C

Mapping User Interface

Mapping Editor

The following sections provide detailed descriptions of the user interface for the Mapping Editor

Mapping Menu Bar

The menu provides access to all features of the mapping editor, including those commonly done by using the mouse.

Table C–1 lists the menus and menu items that appear on the menu bar of the editor.

Table C–1 Mapping Editor Menu Bar

Menu	Menu Item	Description
Mapping		
	Open…	This item launches the general OWB selection dialog and allows you to open another Mapping Editor.
	Add	Create a new operator.
	Mapping Table, Mapping View, Mapping Materialized View, Mapping Flat File, Mapping Fact, Mapping Dimension, Mapping Sequence	
	Add	
	Mapping Transformation, Expression, Aggregator, Filter, Sorter, Joiner, Splitter, Deduplicator	
	Add	
	Constant, Data Generator, External Process, Mapping Input Parameter, Mapping Output Parameter, Pre-Mapping Process, Post-Mapping Process	
	Validate…	Validates the Mapping and all operator expressions. Launches Validation Results Dialog.
	Generate	Generates the code for the mapping and launches a read-only code-editor to view the results.
	Mapping…, Intermediate Result…	

Table C–1 Mapping Editor Menu Bar (Cont.)

Menu	Menu Item	Description
	Print	Prints the contents of the Mapping Canvas.
	Mapping Properties...	Launches the Mapping Property Sheet.
	Close Window	Closes the Mapping Editor.
Edit		
	Properties...	Launches the Property Inspector for the currently selected operator, Attribute Group, or Attribute. If no operator is selected, the menu item is disabled.
	Display Set...	Launches the Display Set dialog for the currently selected Attribute Group. If no Attribute Group is selected, the menu is disabled.
	Reconcile Inbound...	Launches the inbound reconciliation dialog for the currently selected operator.
	Reconcile Outbound...	Launches the Reconcile Outbound dialog for the currently selected operator (if applicable). This can be used on operators containing attribute groups to copy operators.
	Add/Remove	Launches the Add/Remove Attribute Group dialog if an operator is selected. Launches the Add Attribute dialog if an attribute group is selected. If no operator or attribute group is selected, the menu item is disabled.
	Rename	Launches the Rename Dialog. If no operator, Attribute Group or Attribute is selected, the menu item is disabled.
	Delete	Deletes the selected operator and mapping lines. If no operators or mapping lines are selected, the menu item is disabled.
	Synchronize	Synchronizes mapping operator attributes with the corresponding repository object attributes. Useful in multiple user situations.
View		
	Expand	Expands the currently selected operator. If no operator is selected, the menu item is disabled.
	Expand All	Expands all operators on the canvas.

Table C–1 Mapping Editor Menu Bar (Cont.)

Menu	Menu Item	Description
	Collapse	Collapses the currently selected operator. If no operator is selected, the menu item is disabled.
	Collapse All	Collapses all operators on the canvas.
	Select Display Set All …	Opens a list of available display sets from which to choose. Only enabled when an Attribute Group of an operator is currently selected.
	Validation Messages…	Launches the Validation Messages Screen.
	Reports…	Launches the Warehouse Builder Browser and provides a report on the current Mapping.
	Zoom 400% 200% 100% 75% 50% 25%	Zooms the Canvas to the selected level.
	Lineage	Opens the Warehouse Builder Browser Lineage report.
	Impact Analysis	Opens the Warehouse Builder Impact Analysis report.
Window		
	Arrange All	Arranges all Warehouse Builder editors on your desktop.
	Dynamic Window List	List of currently opened Warehouse Builder windows.
Help		
	Contents…	Launches the online help viewer with the Contents tab selected.
	Index…	Launches the online help viewer with the Index tab selected.
	Search…	Launches the online help viewer with the Search tab selected.
	Topics…	Launches the online help viewer with the Mapping Editor topic showing.
	About…	Launches the Warehouse Builder About dialog.

Mapping Editor Toolbar

You can execute certain commands using the Mapping Toolbar rather than the main menu.

The toolbar on the Mapping Editor contains the buttons listed in Table C–2.

Table C–2 Mapping Editor Toolbar

Button	Help Text	Description
Palette	Tool Palette	Toggles between hiding and showing the Tool Palette.
Print	Print Mapping Diagram	Prints a diagram of the Mapping Canvas.
Validate	Validate Mapping	Validates the Mapping and launches the Validation Results Dialog.
Generate	Generates Code	Generates code for the whole Mapping and launches the Code Editor.
Reconcile Inbound	Reconcile an Operator	Invokes the reconciliation inbound dialog for the current selected operator. Only enabled for operators that can be derived from repository objects.
Reconcile Outbound	Reconcile outbound or Create a new repository object	Launches the Reconcile Outbound dialog for the currently selected operator (if applicable). This can be used on operators containing attribute groups to copy operators.
Synchronize	Synchronize this view with a repository	Synchronizes mapping operator attributes with the corresponding repository object attributes. Useful in a multiple user situation.
Properties	Mapping Properties	Launches the Mapping Properties sheet.
Help	Help	Launches the online help viewer, with the Contents tab selected.

Toolbox

You can add an operator to a mapping by dragging its corresponding icon from the Toolbox and dropping it onto the Mapping Editor canvas.

Table C–3 describes the mapping objects available in the Toolbox. The table groups the objects according to their object type.

Table C–3 Toolbox Icons

Group	Icon/Tool Tip	Description
Relational Objects	Mapping Table, Mapping View, Mapping Materialized View, Mapping Sequence, Mapping Flat File	The operator objects that can be associated with entities in the repository.
Dimensional	Dimension, Fact	The operator objects that can be associated with dimensional objects in the repository.
Transformations	Transformation, Expression, External Process, Pre-Mapping, Post-Mapping	The operator objects that transform data.
Other	Filter, Joiner, Constant, Deduplicator, Splitter, Order, Key Lookup, Aggregator, Set Operation, Name and Address, Mapping Input, Mapping Output	Other operators.

Operator Property Inspector

You edit the properties of a mapping operator, attribute, or attribute group using the Operator Property inspector. The content and organization of properties in the Operator Property inspector is different for each operator, attribute, or attribute group.

Table C–4 describes the Operator Property Inspector.

Table C–4 Operator Property Inspector

Category	Name	Description
Button	Find	Click the flashlight icon to start a search. Enter a search string into the Find field. Use an asterisk (*) as a wild card.
Field	Find	This field appears after clicking Find. Enter a search string for attributes or properties in the name/value grid.
Button	Find Next	Click this icon to locate the next property matching the search string, going down.

Table C–4 Operator Property Inspector (Cont.)

Category	Name	Description
Button	Find Previous	Click this icon to locate the previous property matching the search string, going up.
Grid	Name / Value	Each property consists of a name and value pair. The value cell has a value editor which can include a text field, numeric spin box, or drop-down list. A property can require Expression Builder to define it. A [**...**] button is located to the right of the value cell when you click it. Clicking this button launches a custom editor or Expression Builder, depending upon the property. An attribute or property may have a read-only value. An attribute or property with child objects can be expanded and collapsed by clicking on the + or - next to its name.
	Description	The description area shows relevant information for the currently selected name / value pair.

Using the Mouse in the Mapping Editor

This section describes mouse actions in the Mapping Editor canvas.

Left Mouse Click Operations

The following actions can be performed using the left mouse button:

- Dragging objects from the palette

  Clicking the left mouse button on an icon on the object palette, followed by a mouse-drag operation, starts the creation of an operator.

- Mapping & Moving

  Clicking the left mouse button while the pointer is positioned over a node, followed by a mouse-drag operation, starts a Mapping or Move operation. See "Right Mouse Click Operations" for detailed descriptions of the various mapping types and when they are initiated.

- Single Select

  Clicking the left mouse button while the pointer is located over an operator, an attribute, an attribute group, or a mapping line selects the object.

- Multi-Select

Clicking Control and the left mouse button while the pointer is located over an operator, an attribute, or a mapping line adds the object to the list of currently selected objects.

- Undo selection

  Clicking the left or right mouse button followed by a mouse button release, anywhere on the canvas causes the previous list of selected objects to be unselected again.

- Double-click

  Double-clicking for every object on the canvas causes the object to be selected. In addition, double-clicking operators on the canvas does the following:

  – Double-clicking on a minimized node expands the node.

  – Double-clicking on the header of an expanded node launches the Operator Property Inspector either with the root node selected or the most common property selected, Filter: Filter condition or Join: Join condition for example.

  – Double-clicking on an attribute group launches the Operator Property Inspector with the attribute group node opened and selected.

  – Double-clicking on an attribute launches the Operator Property Inspector with the attribute node opened and selected.

Tool Tips

Rolling the mouse pointer over a node, an attribute, or an attribute group on the canvas shows a tool tip with information. Rolling over a node, an attribute group, or an attribute while working on the Mapping Editor canvas causes the object under the pointer to be highlighted if the object is a valid target.

Right Mouse Click Operations

Clicking the right mouse button anywhere on the canvas and immediately releasing the mouse button without moving the pointer displays a pop-up menu.

Table C–5 lists the pop-up menus that are available from the Mapping Editor canvas.

Table C–5 Mouse Right Click Menus

From Where	Menu Item	Description
Canvas		
	Generate…	Generates the code for the mapping and launches the Generation Results dialog. Where applicable, the system may generate code for multiple strategies. For example, if a step can be implemented in set-based and row-based operating mode, the code viewer includes both modes.
	Mapping Properties…	Launches the Mapping Property Sheet.
Operator Header		
	Operator Properties…	Launches the Property inspector for the currently selected operator.
	Reconcile Inbound...	Launches the Inbound Reconcile dialog for the currently selected operator.
	Reconcile Outbound…	Launches the Outbound Reconcile dialog for the currently selected operator (if applicable).
	Edit Name and Description	Launches the Edit Name and Description dialog.
	Add/Remove Groups	Launches the Add/Remove Groups dialog.
	Delete	Deletes the currently selected operator.
	Report	Opens Warehouse Builder Browser reports.
	Lineage	Opens the Warehouse Builder Browser Lineage report.
	Impact Analysis	Opens the Warehouse Builder Impact Analysis report.
	Expand	Expands the currently selected operator.
	Collapse	Collapses the currently selected operator.
Attribute Group		

Table C–5 Mouse Right Click Menus (Cont.)

From Where	Menu Item	Description
	Group Properties...	Launches the Property Inspector for the currently selected attribute group.
	Display Set...	Launches the Display Set dialog for the currently selected Attribute Group.
	Expand Group	Expands the currently selected operator.
	Collapse Group	Collapses the currently selected operator.
	Use Display Set	Opens a list of available display sets from which to choose.
	Generate Intermediate Result	Generates the code for the mapping up to the current operator and opens a read-only code viewer displaying the results.
	Add/Remove Attributes	Launches the Add or Remove Attribute dialog.
	Rename	Launches the Rename dialog.
Attribute		
	Attribute Properties...	Launches the Property Inspector for the currently selected attribute.
	Rename	Launches the Rename dialog.

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 per 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.

Other Dialogs

When adding or editing operators, Warehouse Builder displays dialogs in which you enter information that defines the operator. This section describes those operators.

Add Operator Dialog

Table C–6 describes the Add Operator dialog.

Table C–6 Add Operator Dialog

Category	Name	Description
Radio Button	Create an unbound operator with no attributes	This option is only enabled for the following operator types: Tables, Views, Materialized Views.
Text Field	Name	Enter a valid name for the operator.
Radio Button	Create new operator in repository and bind	Creates an operator for the mapping using a Wizard appropriate for the operator being added to the mapping that inherits its structure from a newly created repository object.
Choice Box Parent Object	Module	Provides a list of available modules in the same project in which a object can be created. For mapping transformation actions, this gives a list of available transformation categories qualified by its owning module. For other mapping objects such as mapping tables, this gives a list of available modules.
Radio Button	Import an operator into repository and bind	Creates an operator for the mapping using the Import Object Wizard that inherits its structure from an imported repository object.

Table C–6 Add Operator Dialog (Cont.)

Category	Name	Description
Choice Box	Module	Provides a list of available modules in the same project in which an object can be imported. By default, the module that the Mapping belongs to is selected.
Radio Button	Select existing object in repository and bind	Adds an operator to the mapping that matches the type and structure of the object that is currently selected in the tree list.
Field	Find	Enter a search string for objects in the Tree List. The search is not case-sensitive, and you can use an asterisk (*) as a wildcard character.
Button	Find	The search string is used to locate the first object in the tree list that matches the string.
Tree List	Object	Lists all objects that match the type of the operator for all Modules in the same project.

Add/Remove Attributes Dialog

Table C–7 Describes the Add/Remove Attributes dialog.

Table C–7 Add/Remove Attributes Dialog

Category	Name	Mandatory	Description
Text Field	Name	Yes	Enter a name for the new attribute. By default, the old name of the object is displayed.
Button	Add	n/a	Adds an attribute to the current list of attributes using the name as specified in the Name text field. Names are verified for correctness and uniqueness.
Button	Remove	n/a	Removes the selected attribute from the list box of current attributes to be added. Button is disabled if no attributes are selected.
List box	Current list of Attributes	n/a	Lists the current attribute groups to be added.

Inbound and Outbound Reconcile Dialog

Table C–8 describes the Inbound and Outbound Reconcile dialog.

Table C–8 Inbound/Outbound Reconcile Dialog

Category	Name	Description
Drop-down list	Object Type	Provides a list of repository object types to reconcile: Tables, Views, Materialized Views, Files, Sequences, Transformations.
Field	Find	Enter a search string for objects in the tree list. By default, the field contains the bound name of the object if the object is bound.
Button	Find	Locates the first object in the tree that matches the string when the Find button is clicked; use asterisks (*) as wildcards.
Tree List	Objects	Lists all objects that match the object type of the operator for all Modules in the same project. By default, the original module and object that the operator was derived from are selected if available.
Check box	Match by Object Identifier	Matches mapping operators to repository objects by their Object Identifier.
Check box	Match by physical (bound) Name	Matches mapping operators to repository objects by attribute physical (bound) names.
Check box	Match by Position	Matches mapping operators to repository objects by the attribute position; line by line matching.

Physical Properties Inspector

Table C–9 describes the Physical Properties inspector.

Table C–9 Physical Properties Inspector

Configuration group	Description
General (Map Name)	This group includes properties that apply to all steps, for example, the auditing level property.
Runtime Parameters	This group has sub-nodes for each operator that was defined in the mapping. Each operator contains a set of physical properties unique to the type of the operator. For example, a table operator has a database link node and a schema node, and a file operator has a file name node.

Table C–9 Physical Properties Inspector (Cont.)

Configuration group	Description
Sources and Targets	This group has sub-nodes for each Mapping Step that the Analyze operation has created. Each step has a set of physical properties that are determined by the type of the step. For example, a PL/SQL step has different properties than a SQL*Loader step. Steps are automatically created and maintained; you cannot add, delete, or rename them.

D

Performance Enhancements

This appendix contains information about partition exchange loading, which is available during the physical configuration of the warehouse. This appendix includes:

- Overview
- When to Use Partition Exchange Loading
- Using Partition Exchange Loading
- Restrictions
- Performance Comparisons
- The Internal Workings
- Function Example 1: GET_PN
- Function Example 2: GET_VC

Overview

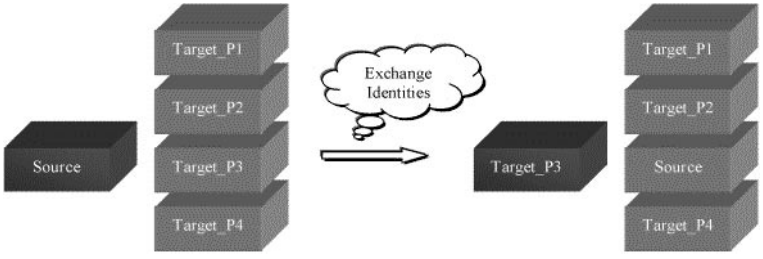
There are many reasons for creating and maintaining data warehouses. One of them is the expected high performance that a data warehouse could bring to business applications, such as DSS, OLAP, and Data Mining. These applications share a salient characteristic: they often need to query a large amount of historical data organized along a time dimension. The horizontal data partitioning technology, available since Oracle8™, can effectively help the queries cut down the amount of data that have to be processed by many levels of magnitude. This is done through an automatic mechanism in the Oracle SQL optimizer known as partition pruning. Data partitioning is also an enables the backup and / or removal of dormant data that has aged out over time.

The way data is partitioned is a key data warehouse design issue, which must be considered after the logical design of a data warehouse. Oracle9*i* Warehouse Builder provides assistance in partitioning various physical objects, including tables, dimensions, facts, materialized views, and indexes.

With Warehouse Builder, design for data partitioning has one added importance: it can also be leveraged within a certain context to achieve significant performance gains in the process of loading a data warehouse. This new Warehouse Builder runtime performance feature is known as Partition Exchange Loading (PEL).

PEL technique performs loading of new data by exchanging them into a target table as a partition. What actually get exchanged are merely identities, in the following manner. The table that holds the new data takes over the identity of one empty partition from the target table; at the same time, this empty partition assumes the identity of the source table. The whole exchange process is purely a DDL operation. No data movement is involved. The general idea of PEL is depicted in Figure D–1.

Figure D–1 Fundamental Concept of Partition Exchange



In Figure D–1, data from the Source table needs to be inserted into a target table consisting of four partitions: Target_P1, Target_P2, Target_P3, and Target_P4.

Suppose that the new data needs to be loaded into Target_P3. The partition exchange operation will swap the names on the data objects. The actual data never moves. After the exchange, the Source table is renamed Target_P3, and the Target_P3 is renamed Source. Most importantly, the target table is still composed of four partitions named Target_P1, Target_P2, Target_P3, and Target_P4. The partition exchange operation provided by Oracle8^{TM} completes the loading process without generating the need for data movement.

Figure D–1 shows a highly simplified case where only one table is presented as a source, and there is an implicit assumption that this source table is readily available for changing its identity. Being a general data warehouse design tool, Warehouse Builder operations are not this simple. Warehouse Builder must be prepared for the more general possibility, that the source may be the results from a join of multiple source tables. Some of these source tables may also be located in other databases.

In general, there may not be a single local table for partition exchange. The design decision in Warehouse Builder is to automatically create a temporary table which materializes results from source processing before the partition exchange actually happens.

Will the materialization of results from source processing hurt performance? Will it render the PEL useless since data movement (i.e., DML) has not been avoided? In the case that there is no single local table for partition exchange, it is quite obvious that avoiding DML entirely is not possible. As such, the PEL's performance potential can only be gauged by an examination of how it does DMLs, and specifically, the INSERTs. The remainder of this section presents a brief explanation of how PEL works. This should answer the above questions. Getting familiar with the technical background will be very important to you if you plan to use the Partition Exchange Loading feature.

Overhead-Free INSERT

When you use PEL, a new temporary or staging table is automatically created to hold results from source processing. Because this staging table is purely internal to the PEL process and invisible to general users, the PEL process does not create any indexes and constraints before the INSERT operation is completed. This measure can drastically reduce the amount of data that need to be processed due to index maintenance and foreign key lookups.

In contrast, using the normal non-PEL style processing requires data to be directly inserted into the target that often has to carry several huge indexes and many constraints. Of course, you could choose to drop all the indexes and disable all the constraints before loading, but when the target has accumulated enough historical

data, recreating indexes and re-enabling constraints after loading may be a very costly decision.

Fast Index Creation

After new data has been loaded into the temporary table, the PEL method creates all the indexes on the temporary table only. The index creations benefit from a parallel index creation mechanism provided by the Oracle database server. It also uses the NOLOGGING option to further speed up the index building process. Using the PARALLEL NOLOGGING options, index creations generate writes for only the index structures, and I/O conflicts are minimized. As the result, the index creation time is minimized.

Fast Constraint Maintenance

After indexes have been created on the temporary table, primary and unique keys are instantly enabled and validated a by using the USING INDEX option of the ALTER TABLE ADD CONSTRAINT command.

Foreign key constraints are maintained by making use of a new parallel validation feature in Oracle8*i*. This is done by first adding a foreign key constraint in ENABLED NOVALIDATE state. Then, a subsequent command does the real foreign key validation in parallel. Adding a foreign key with ENABLED NOVALIDATE status does not hinder performance. The subsequent parallel constraint validation can fully utilize system resources. The overall design is an optimal constraint maintenance mechanism.

In contrast, if the PEL is not used, it can be an almost impossible task to disable and re-enable existing constraints. Suppose, for example, 500,000 rows of new data generated from one-day activities have been inserted into a target table already containing 500 million rows of data; and one primary constraint and one foreign key constraint were disabled before loading and they must now be re-enabled. This whole process amounts to scanning more than 500 million rows of data twice, plus building a unique index. It could be hundreds of times slower than if only the new data was manipulated as in PEL.

Error Handling

Whenever an error is encountered during inserting into the temporary table, building indexes, or enabling constraints, the temporary table is dropped and the error logged. The target is not altered. Data in the target is still in its previous state.

Minimum Target Locking

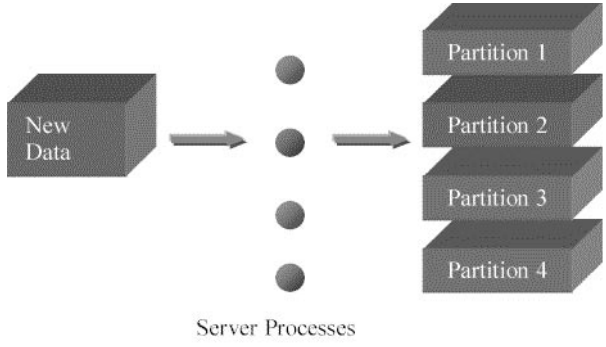
During the entire PEL loading process, except for the final partition exchange step (which normally takes almost no time because it is a pure DDL operation), the target table is always available for any database operations, including any DML operations.

Parallel Direct-Path INSERT

In PEL, insertion into the temporary table is always done using parallel direct-path INSERT. On a multiprocessor machine with a correctly striped tablespace, more than 90% utilization on each CPU can be consistently achieved. The insertion completes in the shortest possible time.

In contrast, if the PEL method is not used, several factors can hinder the loading process from fully utilizing the CPU resource. First, index maintenance can generate more I/O requests and reduce CPU utilization. (The PEL technique would have no index maintenance during insert.) Second, when loading into a partitioned table, a more serious issue results from the fact that Oracle8*i* assigns only one server process to each partition. If all data must go into same partition as is true for incremental loading of daily or monthly data, the loading process runs serially, as shown in Figure D–2.

Figure D–2 One Server Process Assigned to Each Partition



A concrete measurement can help understand the performance impact of incremental loading into only one partition of a partitioned table. On a Sun Ultra 450 server with four 400Mhz CPUs, 6 million rows from one daily collection are inserted into a target table that is partitioned by day. The completion time is 3 minutes 9 seconds. But when the same amount of data is inserted into a

non-partitioned table using parallel direct-path INSERT, it completes in 1 minute 48 seconds. So the serial insert in the partitioned case lost roughly two times of performance on the 4-processor machine. The PEL method solves this problem by always performing parallel direct-path INSERT into a non-partitioned table.

When to Use Partition Exchange Loading

Like all performance features, the PEL technique is advantageous only in a certain context, and can be deemed not useful in some other situations. The foremost condition required by the PEL is that the target table must be partitioned. Another condition under which the PEL could provide greatest performance benefit is if target table has already accumulated a huge amount of historical data. An example is an application where the warehouse collects data from an OLTP database or Web log files daily. Each row of source data is about one mouse click from a Web user. Depending on the popularity of the site, as much as several million rows per day could be generated. These data then need to be transformed and loaded into a data warehouse holding a lot more historical data, e.g., from the past five years. For this type of situations, the PEL will be the key to scalable loading performance that makes the loading time correlated only to the amount of new data.

A third condition under which the PEL can help greatly is when all the new data must be loaded into the same partition in a target table, as shown in Figure D–2. If the data warehouse collection is scheduled on a regular basis, this condition can normally be satisfied.

Again, take the clickstream warehouse as an example. If the target table is partitioned by day, then all the daily data will have to be loaded into one partition. The non-PEL loading will go serially, but the PEL will be able to run parallel direct-path INSERT.

In case the target table is partitioned by month and the collection of new clickstream data is still scheduled daily, new data may have to be inserted into a nonempty partition. In case the target partition is not empty, the PEL process will not actually perform partition exchange. It will directly insert new data into one explicitly specified partition, still using parallel direct-path INSERT. The drawback on performance is that the insertion is accompanied by simultaneous index and constraint maintenance. Therefore, using the PEL method to load data into a nonempty partition will go slower than if the target partition is empty.

Still a third case in our clickstream example: assume the target table is partitioned by hour and new data is still collected daily. New data collected from one day will be able to go into separate partitions in parallel. In this case, the PEL is not useful

anymore. In fact, in Warehouse Builder, the PEL does not work for such a case. Table D–1 summarizes these three possibilities.

Table D–1 Loading Size vs. Partition Granularity Considerations

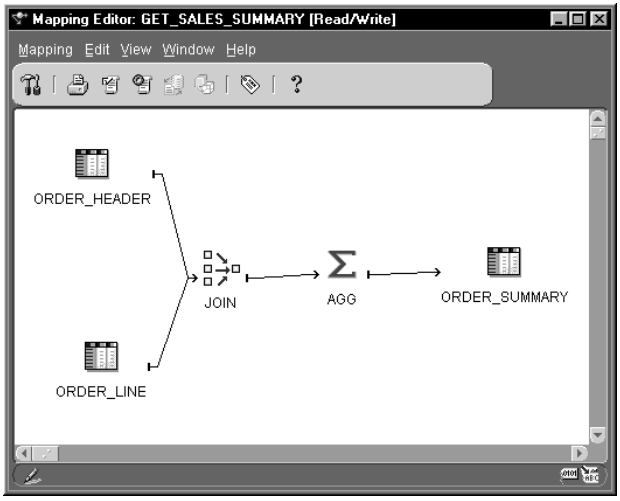
SITUATION	PERFORMANCE
Loading size = Partition Granule (For example, load daily and partition by day.)	Highest performance.
Loading size < Partition Granule (For example, load daily and partition by month.)	High performance but slower than previous case.
Loading size > Partition Granule (For example, load daily and partition by hour.)	Does not work in Warehouse Builder using PEL.

As Table D–1 shows, the PEL method can be used in the case that the loading size is less than or equal to the partition granularity of the target. In Warehouse Builder, you can specify the target partition granularity. But there is currently no validation to check if the loading size is not larger than the partition granularity.

Using Partition Exchange Loading

This section presents a step-by-step example showing how the Partition Exchange Loading feature can be used in Warehouse Builder. Two areas need configuration: the mapping and the target table, dimension, or fact. This section first describes an example mapping, then explains what needs to be configured in the mapping and the target.

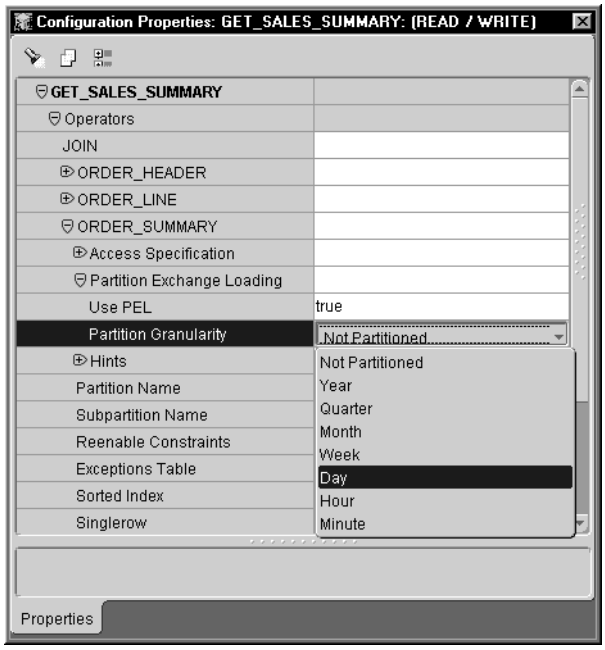
Figure D–3 Example: The GET_SALES_SUMMARY Mapping



Suppose it has been decided that all new data to ORDER_SUMMARY table will always go into same partition, and most times they go into an empty partition. Partition Exchange Loading into the ORDER_SUMMARY table appears to be a good strategy under these conditions.

Configuring the Mapping

Figure D–4 Mapping Property Inspector for Mapping GET_ORDER_SUMMARY



In order to configure the mapping to use PEL, open the Mapping Properties inspector. The Properties inspector for the GET_SALES_SUMMARY is show in Figure D–4. In the property inspector window and within the Operators category, each of the sources and the target starts a property group. Expand the property group for ORDER_SUMMARY as in Figure D–4. You will see a subgroup labeled Partition Exchange Loading. Within this group, two properties must be configured. The first is a Boolean property named set PEL. Set to true for this property. The other property is named Partition Granularity. This property tells Warehouse Builder's PL/SQL code generator how the target table (ORDER_SUMMARY) is partitioned. There are seven levels of partition granularity you can choose from, as shown in Figure D–4.

Suppose the ORDER_SUMMARY table is partitioned by day. We then select Day from the drop-down list as the value for the Partition Granularity property. And that is all we need to do on the mapping in order to enable the Partition Exchange

Loading feature. Once the PEL is enabled, the Warehouse Builder code generator will generate a different kind of batch processing code.

Configuring the Target

After the mapping is configured, there is still one more area of configuration to be done. This second area of configuration is for the target. It turns out that the process for configuring the target is a lot more complicated than the process for configuring the mapping.

To configuring the target, follow these steps:

1. Create All Partitions.
2. Create All Indexes Using the LOCAL option.
3. Specify the USING INDEX Option for Primary / Unique Keys.

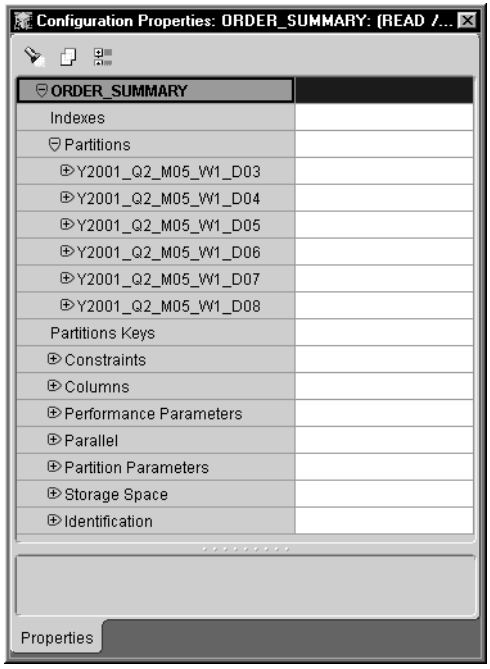
Create All Partitions

All the partitions must be created before the PEL method can be used. Warehouse Builder users can use the table / dimension / fact inspector to add partitions and to modify their properties. After a partitioned object has been deployed into database, the DBA can also add more partitions or to merge several partitions together. In the following explanation about Step 1, only the instructions on how to use Warehouse Builder to create partitioned table are given. On the possible DBA operations, please refer to *Oracle8i Server Administration Guide.*

In our example, since we selected Day as partition granularity for ORDER_SUMMARY table when configuring the mapping, we need to create all the needed daily partitions, one for each day of new data. To create partitions for a table, dimension, or fact, use its property inspector. Figure D–5 below shows the property inspector window for the table ORDER_SUMMARY. This figure shows that six partitions have been added for this table. (The instructions on how these partitions are added are not included.)

Notice the naming of all the partitions in Figure D–5. Although Warehouse Builder users must manually key in those names when they use Warehouse Builder to add partitions, all partition names must follow a strict naming convention. For example, for the partition that will hold data for May 3, 2001, its name must be Y2001_Q2_M05_W1_D03.

Figure D–5 Property Inspector for Table ORDER_SUMMARY



The partition naming convention is in fact very straightforward. For example, the partition name Y2001_Q2_M05_W1_D03 corresponds to Year 2001, Quarter 2, Month 5, Week 1, and Day 3, which is exactly May 3, 2001. But can the same partition be simply named Y2001_M05_D03 without the quarter and week information? The answer is no. The partition name must contain information of all the time levels leading to the intended granularity.

The detailed naming scheme is designed for ease of partition administration later. If the DBA decides to merge all the daily partitions into monthly partitions, the DBA can easily find which partitions to merge together based on their names. For example, all the daily partitions for May 2001 are named like Y2001_Q2_M05*. (Here the asterisk means a wildcard matching a string of characters.) They can be merged to form a renamed partition Y2001_Q2_M05. Furthermore, this naming scheme helps create a consistency between the partitions' name ordering and value ordering. That is, if the partitions are sorted based on the partition names, the partitions holding smaller DATE values will appear earlier.

However, to many people, figuring out to which quarter and week a date belongs, as is required by the naming convention, is not as immediate as figuring out its day,

month, and year components. This may lead to errors during partition creations. If partitions are created with errors, it will be extremely difficult to correct them later. To help eliminate errors during partition creations, you may find the two PL/SQL functions toward the end of this appendix very useful.

Function Example 1: GET_PN on page D-19 contains source code for a stand-alone PL/SQL function named get_pn, which stands for get partition name. It can help Warehouse Builder users calculate partition name from a given date value. For example, if we want to find the correct partition name for the partition containing value May 3, 2001, and we want to partition target table by day, the correct partition name can be obtained by issuing a SQL statement that calls the get_pn function as follows:

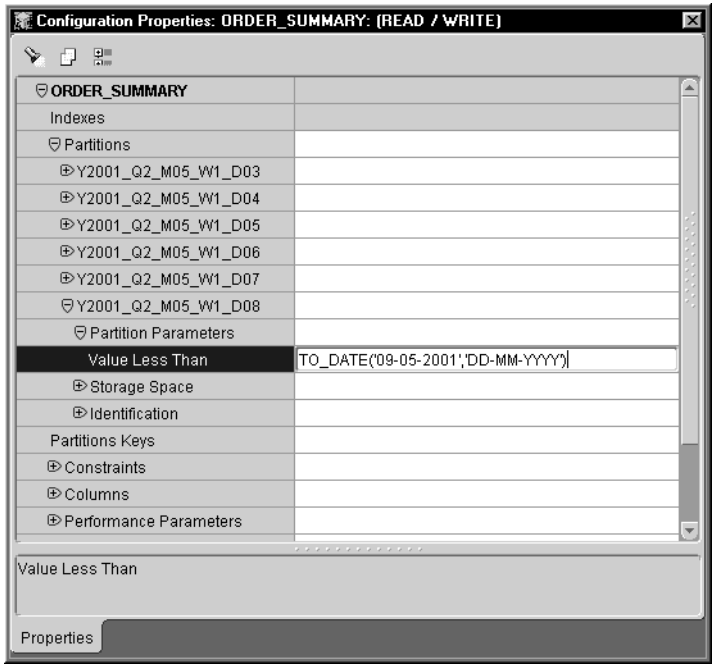
```
SQL> SELECT get_pn(TO_DATE('03-MAY-2001','DD-MON-YYYY'),
2               'DAY') partition_name
3  FROM DUAL;

PARTITION_NAME
--------------------
Y2001_Q2_M05_W1_D03
```

The first argument is a date value, and the second argument is the partition granularity. The function returns the correct partition name under the intended partition granularity for the partition containing the date value. The above example shows that the correct partition name for the partition containing May 3, 2001 is Y2001_Q2_M05_W1_D03. If, however, the second argument to the get_pn function was Month instead of Day, the partition name result would be Y2001_Q2_M05.

After all the partitions are added with correct names, the partitions must be further configured for their VALUE LESS THAN properties as shown in Figure D–6 below. Figure D–6 shows that the VALUE LESS THAN property for partition Y2001_Q2_M05_W1_D08 is being configured. The property value is TO_DATE('09-05-2001','DD-MM-YYYY'). Using the name and the VALUE LESS THAN property, Warehouse Builder will generate a DDL script for creating the partitioned table. Part of the DDL script related to the partition shown in Figure D–6 will read as follows.

```
. . .
PARTITION Y2001_Q2_M05_W1_D08
      VALUES LESS THAN (TO_DATE('09-05-2001','DD-MM-YYYY')),
. . .
```

Figure D–6 Specifying Value Less Than Property for a Partition



The get_vc function is used in a similar manner as get_pn. First, find any value that would fall within the partition you are trying to calculate the VALUE LESS THAN property value. Then call the get_vc function as follows.

```
SQL> SELECT get_vc(TO_DATE('08-MAY-2001','DD-MON-YYYY'),
2                 'DAY') value_less_than
3  FROM DUAL;

VALUE_LESS_THAN
-----------------------------------
TO_DATE('09-05-2001','DD-MM-YYYY')
```

Create All Indexes Using the LOCAL option

Figure D–7 Configure an Index as a Local Index

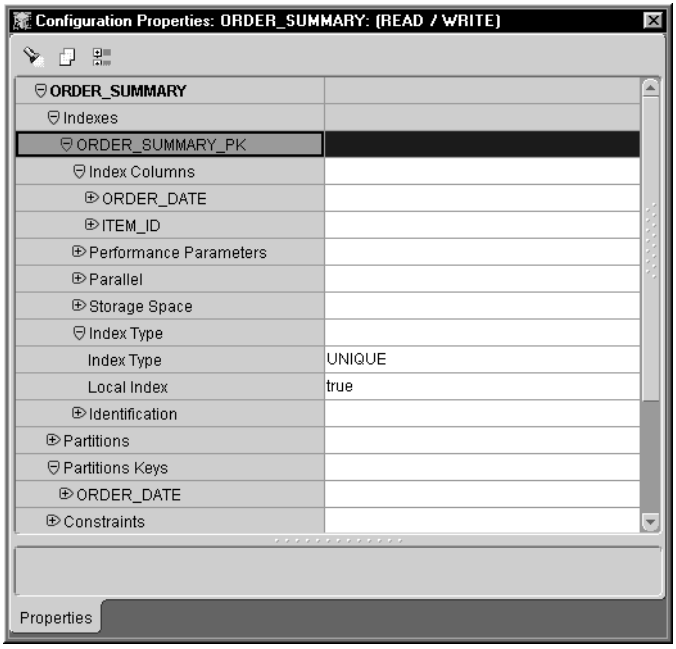


Figure D–7 above shows an index (ORDER_SUMMARY_PK) is added to ORDER_SUMMARY table. This index has two columns, ORDER_DATE and ITEM_ID. The Index Type parameter is set to UNIQUE and the Local Index parameter is set to true. If other indexes existed on the ORDER_SUMMARY table, they should all be configured in this way, i.e., with the Local Index parameter set to true.

Finally, remember that if an index is created as a local index, the partition key column must be the leading column of the index, or the Oracle server does not create the index. In Figure D–7 above, the partition key is ORDER_DATE. It is the leading column in index ORDER_SUMMARY_PK.

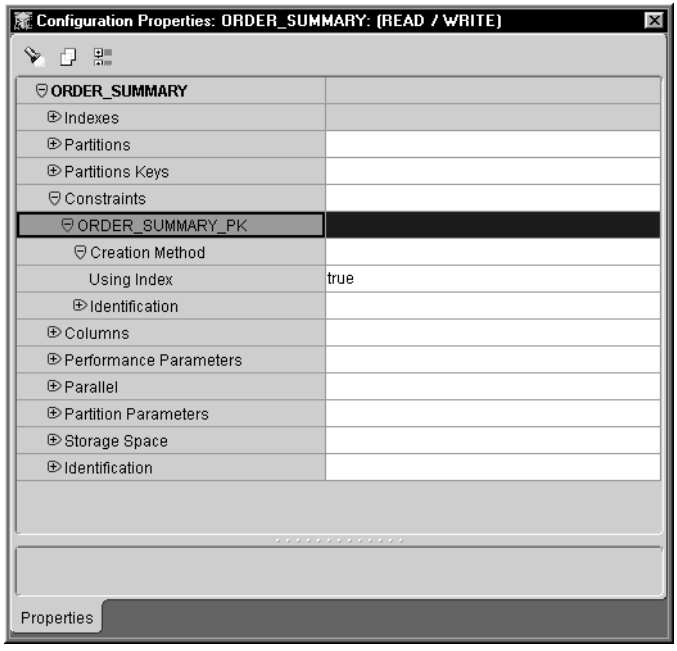
Specify the USING INDEX Option for Primary/Unique Keys

The very last step in configuring the target is for specifying that all primary and unique key constraints are created with the USING INDEX option. Using the USING INDEX option, a constraint will not trigger automatic index creation when it is being added to the table. Instead, the server will look among existing indexes for

one with same column list as that of the constraint. This naturally implies that each primary or unique key constraint must be backed by a user-defined unique local index.

Figure D–8 below shows an example in which the primary key constraint on ORDER_SUMMARY table is specified with the USING INDEX option. The index that supports this option was created in Step 2 earlier.

Figure D–8 Specify a Constraint with USING INDEX option



Indexes and constraints can share the same name without problem, and it is customary to create a constraint and its underlying index with the same name as shown in Figure D–7 and Figure D–8.

Restrictions

There are several restrictions that Warehouse Builder users need to be aware of. These restrictions are necessary to minimize the complexity of implementation of the PEL feature. Some of the restrictions might be lifted in future releases.

Restriction 1: Allow One Date Partition Key

Currently, only one partition key column is allowed, and this column must be of DATE data type. Numeric partition key is not supported in Warehouse Builder. This will most likely be enhanced in near future releases.

Restriction 2: Allow Natural Calendar System Only

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. But this support will most likely be added in near future releases.

Restriction 3: All Data Partitions Must be in the Same Tablespace

The current release requires that all partitions of a target (table, dimension, or fact) must be created in the same tablespace. This requirement is needed for eliminating user intervention during the PEL process. Otherwise, users must pass a runtime parameter to the PEL mapping to specify the target tablespace.

Restriction 4: All Index Partitions Must be in the Same Tablespace

This requirement is established for the same purpose as the restriction 3. Users are thus not required to pass a runtime parameter specifying the index tablespace. However, the index tablespace can be different from the data tablespace.

Performance Comparisons

There can be three possible methods for doing incremental loading into a target, which cumulates historical data. This type of target is usually a fact table. The three loading methods are

- Method 1: Use the PEL method and load into a partitioned target table.
- Method 2: Do not use the PEL method and load into a partitioned table.
- Method 3: Do not use the PEL method and load into a non-partitioned table.

The qualitative performance advantages of PEL can clearly be seen in the section on General Feature Description. This section demonstrates the quantitative performance advantage of the PEL over the other two incremental loading methods.

An experiment is performed on a four-processor Sun Ultra 450 server. Each CPU has a speed of 400MHz. The memory size is 2GB. There are twenty 9.1GB internal disk drives. The database is created using Oracle8*i* Release 3.

The experiment uses each of the three loading methods to do exactly the same loading operations. The loading operations consist of a series of inserts into a target table. Each insert is for loading one-day worth of new data, with a volume of 500,000 rows. In this case the target is partitioned (by day), and each batch goes into one partition. The most important aspect about the target is that there is an index created on it. The index is built on four numeric columns and one DATE column. If the target is partitioned (as is true in Methods 1 and 2), the index is also local.

In this setting, Method 1 (the PEL) will have advantage over other two methods in index maintenance. Method 1 will also have advantage over Method 2 in doing parallel direct-path insert.

Method 2, which employs partitions for index maintenance, will be able to avoid having to traverse in the global index structure that can grow larger as more data are loaded.

The results confirmed the qualitative analysis discussed so far. Method 3 (no PEL on non-partitioned table) is clearly not a good choice. (Imagine what the loading time would be after one year!) Method 2 does maintain independence of load time from already loaded data, but it is about three times slower than the PEL. The lack of parallelism is believed to be the main culprit for the loss of performance in Method 2.

The Internal Workings

This section is presented only for the purpose of helping Warehouse Builder users better understand how the Partition Exchange Loading works so that they can better plan and use this feature. The explanation below should not be taken as a recipe for custom warehouse construction because the method reported herein might be subject to patent protection, and Oracle only supports the PEL-style PL/SQL code that is generated from Warehouse Builder without modifications.

The PEL technique is a set-based processing method (or generally called batch processing method in the data warehouse field). Therefore, to actually run a mapping with the PEL technique, the processing mode must be set to set-based or set-based with fail-over.

When a mapping using the PEL technique is run, the following things happen, in this order.

Step 1: Find target's tablespace name. If the target has more than one tablespace, the loading process is aborted with an error message like the following.

> Fatal error: all partitions of target table RT_OWNER.ORDER_SUMMARY must be in one tablespace

Step 2: Find the tablespace name for all indexes. If there are more than one tablespace name found, the loading process is aborted with an error message like the following.

> Fatal error: all partitions of local indexes on table RT_OWNER.ORDER_SUMMARY must be in one tablespace

Step 3. Create a temporary table that has same schema definition as the target. Should this step fail, the loading process is aborted with the following error message.

> Fatal error: failed the attempt to create a temporary table like table RT_OWNER.ORDER_SUMMARY

Step 4: Parallel direct-path load into the temporary table. If there is anything wrong in this step, the batch processing terminates with an Oracle server error, but the entire loading process is not aborted. The loading process may drop down to row-by-row auditing mode to find error details if the set-based with fail-over mode is selected.

Step 5: Find partition name from the first row of freshly loaded data. If this step fails, the following error message is logged in Warehouse Builder audit trail.

> Fatal error: calculating partition name failed

Step 6: If the target partition is *not* empty, then parallel direct-path insert into the target partition. Oracle server errors may occur and be logged in Warehouse Builder audit trail. But the loading process is not aborted. It may drop down to row-by-row processing if user so configured the mapping. No matter if errors occur, the mapping terminates.

Step 7: If the target partition is empty, all indexes are created on the temporary table. If errors occur in this step, the loading process is not aborted. The row-based process may be invoked if user selected set-based with fail-over mode.

Step 8: Add all constraints, including primary / unique key constraints and foreign key constraints. If errors occur in this step, the loading process is not aborted. The row-based process may be invoked if user selected set-based with fail-over mode.

Step 9: Exchange partition. If error occurs, the loading process is aborted with the following error message.

Fatal error: failed exchanging partition Y2001_Q2_M05_W1_D03 in target RT_OWNER.ORDER_SUMMARY

Step 10. Drop the temporary table. If any error occurs, a warning is logged in Warehouse Builder audit trail, but the mapping is still considered being successful. The warning looks like the following.

Warning: cannot drop table RT_OWNER.T53

Function Example 1: GET_PN

Deploy this function in any Oracle database (version 8 or later).

```
CREATE OR REPLACE FUNCTION get_pn (p_date         IN DATE,
                                   p_granularity  IN VARCHAR2)
                                   RETURN VARCHAR2 IS
   x_pn VARCHAR2(30);
   x_granularity VARCHAR2(30) := UPPER(p_granularity);
BEGIN
   x_pn := 'Y'||TO_CHAR(p_date, 'YYYY');
   IF x_granularity != 'YEAR' THEN
      x_pn := x_pn||'_Q'||TO_CHAR(p_date,'Q');
IF x_granularity != 'QUARTER' THEN
   x_pn := x_pn||'_M'||TO_CHAR(p_date,'MM');
   IF p_granularity != 'MONTH' THEN
      x_pn := x_pn||'_W'||TO_CHAR(p_date,'W');
      IF p_granularity != 'WEEK' THEN
         x_pn := x_pn||'_D'||TO_CHAR(p_date,'DD');
         IF p_granularity != 'DAY' THEN
            x_pn := x_pn||'_H'||TO_CHAR(p_date,'HH24');
            IF p_granularity != 'HOUR' THEN
               x_pn := x_pn||'_M'||TO_CHAR(p_date,'MI');
            END IF;
         END IF;
      END IF;
   END IF;
END IF;
   END IF;
   RETURN x_pn;
END get_pn;
/
```

Function Example 2: GET_VC

Deploy this function in any Oracle database (version 8 or later).

```
CREATE OR REPLACE FUNCTION get_vc (p_date        IN DATE,
                                   p_granularity IN VARCHAR2 DEFAULT 'DAY')
                                                    RETURN VARCHAR2 IS
    x_vc VARCHAR2(80);
    x_granularity VARCHAR2(30) := UPPER(p_granularity);
    x_year    NUMBER;
    x_quarter NUMBER;
    x_month   NUMBER;
    x_day     NUMBER;
    x_hour    NUMBER;
    x_minute  NUMBER;
  BEGIN
    IF x_granularity NOT IN ('YEAR',
                             'QUARTER',
                             'MONTH',
                             'WEEK',
                             'DAY',
                             'HOUR',
                             'MINUTE') THEN
      x_granularity := 'DAY';
    END IF;

    x_year := TO_NUMBER(TO_CHAR(p_date, 'YYYY'));
    IF x_granularity = 'YEAR' THEN
      x_year := x_year + 1;
      x_vc := 'TO_DATE(''01-01-'||x_year||''',''DD-MM-YYYY'')';
    ELSE
      x_quarter := TO_NUMBER(TO_CHAR(p_date, 'Q'));
      IF x_granularity = 'QUARTER' THEN
        x_quarter := x_quarter + 1;
        IF x_quarter = 5 THEN
          x_year := x_year + 1;
          x_quarter := 1;
        END IF;
        IF x_quarter = 1 THEN
          x_vc := 'TO_DATE(''01-01-'||x_year||''',''DD-MM-YYYY'')';
        ELSIF x_quarter = 2 THEN
          x_vc := 'TO_DATE(''01-04-'||x_year||''',''DD-MM-YYYY'')';
        ELSIF x_quarter = 3 THEN
          x_vc := 'TO_DATE(''01-07-'||x_year||''',''DD-MM-YYYY'')';
        ELSE
```

```
    x_vc := 'TO_DATE(''01-10-'||x_year||''','DD-MM-YYYY'')';
  END IF;
ELSE
  x_month := TO_NUMBER(TO_CHAR(p_date, 'MM'));
  IF x_granularity = 'MONTH' THEN
    x_month := x_month + 1;
    IF x_month = 13 THEN
      x_year := x_year + 1;
      x_month := 1;
    END IF;
    x_vc := 'TO_DATE(''01-'||TO_CHAR(x_month,'09')||'-'||x_year||
                                          ''','DD- MM-YYYY'')';
  ELSE
    IF x_granularity = 'WEEK' THEN
      IF p_date+7 > LAST_DAY(p_date) THEN
        x_day := 1;
        x_month := x_month + 1;
        IF x_month = 13 THEN
          x_year := x_year + 1;
          x_month := 1;
        END IF;
      ELSE
        x_day := (TRUNC(TO_NUMBER(TO_CHAR(p_date,'DD'))/7)+1)*7+1;
      END IF;
      x_vc := 'TO_DATE('''||TO_CHAR(x_day,'09')||'-'||
                           TO_CHAR(x_month,'09')||'-'||x_year||
                                        ''','DD- MM-YYYY'')';
    ELSE
      IF x_granularity = 'DAY' THEN
        x_vc := 'TO_DATE('''||TO_CHAR(p_date+1,'DD-MM-YYYY')||
                                          ''','DD-MM-YYYY'')';
      ELSE
        x_day := TO_NUMBER(TO_CHAR(p_date,'DD'));
        IF x_granularity = 'HOUR' THEN
          x_hour := TO_NUMBER(TO_CHAR(p_date,'HH24'));
          IF x_hour = 23 THEN
            x_vc := 'TO_DATE('''||TO_CHAR(p_date+1,'DD-MM-YYYY')||
                                          ''','DD-MM-YYYY'')';
          ELSE
            x_hour := x_hour + 1;
            x_vc := 'TO_DATE('''||TO_CHAR(p_date,'DD-MM-YYYY')||
                                  TO_CHAR(x_hour,'09')||
                                          ''','DD-MM-YYYY HH24'')';
          END IF;
        ELSE
```

```
                x_minute := TO_NUMBER(TO_CHAR(p_date,'MI'));
                IF x_minute = 59 THEN
                  IF x_hour = 23 THEN
                    x_vc := 'TO_DATE('''||TO_CHAR(p_date+1,'DD-MM-YYYY')||
                                                ''',''DD-MM-YYYY'')';
                  ELSE
                    x_hour := x_hour + 1;
                    x_vc := 'TO_DATE('''||TO_CHAR(p_date,'DD-MM-YYYY')||
                                  TO_CHAR(x_hour,'09')||
                                                ''',''DD-MM-YYYY HH24'')';
                  END IF;
                ELSE
                  x_minute := x_minute + 1;
                  x_vc := 'TO_DATE('''||TO_CHAR(p_date,'DD-MM-YYYY HH24:')||
                                  TO_CHAR(x_minute,'09')||
                                                ''',''DD-MM-YYYY HH24: MI'')';
                END IF;
              END IF;
            END IF;
          END IF;
        END IF;
      END IF;
    END IF;
    RETURN x_vc;
END get_vc;
   /
```

E

Batch Services API

The Warehouse Builder Batch Services interface is called `WBBatchInvokeAPI.java`. All third parties should use this Batch Service API to integrate with and invoke essential services that are provided by Warehouse Builder. The services supported through Batch Services are metadata import, validation, generation, and deployment.

`WBBatchInvokeAPI` can be accessed by importing in a Java class by using the following:

1. `oracle/wh/service/sdk/batchservice/WBBatchInvokeAPI.java`
2. `oracle/wh/service/impl/batchservice/WBBatchInvoke.java`

The Batch Service interface provides the following types of methods to enable third party developers to access Oracle9*i* Warehouse Builder through batch interaction:

- Repository
- Validation
- Generation
- Deployment
- Metadata Import
- Metadata Export

Repository

Repository methods include:

- Open Repository Connection
- Close Repository Connection
- Projects
- Modules

Open Repository Connection

To establish a connection with the metadata repository, use the following method. You must enter five parameters: username, password, host name, port number, and SID. This method will cause a WBConnectionFailureException and WBException if the method fails to establish a connection with the repository.

```
public void openConnection(
    String username, String password,
    String host, String port, String sid)
    throws WBConnectionFailureException, WBException;
```

Close Repository Connection

The following methods are available for closing repository connections:

- To close the connection without committing the repository data, use the following method:

  ```
  public void closeConnection() throws WBException,
  SQLException;
  ```

- To close the connection after committing the repository data, use the following method. This allows the user to have control over committing the data before closing the connection.

  ```
  public void commitCloseConnection() throws WBException,
  SQLException;
  ```

Projects

The following methods are available for manipulating projects:

- To retrieve an array of projects that are currently available, use the following method:

```
public String[] getProjects() throws WBException;
```

- To make a project the current project, insert the project name into the following method:

```
public void setCurrentProject(String projectName) throws
WBException;
```

Modules

The following methods are available for manipulating modules:

- To get all the applications that are present within a project, use the getModules() method. It returns an array of application names that are present in the given project.

```
public String[] getModules() throws WBException;
```

- Assign an application to a current application by using setCurrentModule method. Provide the application name as the parameter. The return type is void. WBException is thrown when Warehouse Builder is not able to set the current module.

```
public void setCurrentModule(String appName) throws
WBException;
```

Validation

An element such as a dimension or a table must be validated. This can be done using the validate method. Provide an application name, the element name, and type of the element. The `typename` can be any of the following: TABLE, VIEW, MATERIALIZEDVIEW, DIMENSION, FACT, SEQUENCE, MAPPING.

```
public boolean validate(
    String appName, String elementname, String typename)
    throws WBException;
```

After the validation has been performed, you can get the validation results in an array of strings.

```
public String[] getValidationResults(
    String appName, String element, String typeName)
    throws WBException;
```

Display the validation results in the console by using the following method:

```
public void displayValidationResults(
    String appName, String element, String type)
    throws WBException;
```

Generation

To generate an element, use the following generate method. The `typename` can be any of the following: TABLE, VIEW, MATERIALIZEDVIEW, DIMENSION, FACT, SEQUENCE, MAPPING.

```
public void generate(
    String appName, String elementname, String typename)
    throws WBException;
```

Deployment

The following methods allow you to deploy your previously generated scripts to either a file system or database. The `typename` can be any of the following: TABLE, VIEW, MATERIALIZEDVIEW, DIMENSION, FACT, SEQUENCE, MAPPING.

Deployment into a File System

```
public void deployInFileSystem(
    String appName, String elementname, String typeName,
    String deployDir)
    throws WBException;
```

Deployment in a Database

```
public void deployInDatabase(
    String appName, String elementname, String typename,
    String user, String passwd,
    String host, String port, String sid)
    throws WBException, Exception;
```

Deployment Results can extracted by using the following:

```
public String[] getDeploymentResults() throws WBException;
```

> **Note:** Batch Services does not generate objects when you run the deployment methods. You must generate the scripts first by using the generation method.

Metadata Import

The following method allows importing of MDL exported data as an input stream. The log file parameter must include the full path name. The Mode denotes the mode of importing (Create, Update). The usePhyiscalName parameter ensures that the user uses a physical name while importing.

```
public void importMetaData(
    InputStream importFileStream,  String logFile,
    String mode, boolean usePhysicalNames)
    throws WBException, IOException;
```

The following method allows the import of Metadata using an MDL exported file without using the character set. The default character set is used.

```
public void importMetaData(
    String m_import_file, String m_log_file,
    String m_mode, boolean m_usePhysicalNames)
    throws WBException;
```

The following method allows the import of metadata using an MDL exported file. A valid character set string would look like WE8MSWIN1252.

```
public void importMetaData(
    String m_import_file, String m_log_file,
    String m_mode, boolean m_usePhysicalNames,
    String characterSet)
    throws WBException;
```

Metadata Export

The following method exports metadata using mandatory parameters.

```
    public boolean exportMetaData(
```

```
        String projectName,
        String appName,
        String elementName,
        String typeName,
        String exportFileName,
        String logFileName,
        String characterSet,
        String fieldSeparator
        ) throws WBException, Exception
```

The following method exports metadata using mandatory parameters. It also allows the user to specify the use of the physicalName. The physicalName argument can be true or false. The default value is true.

```
public boolean exportMetaData(
        String projectName,
        String appName,
        String elementName,
        String typeName,
        String exportFileName,
        String logFileName,
        String characterSet,
        String fieldSeparator,
        String physicalName
        ) throws WBException, Exception
```

The following method exports metadata using mandatory parameters and optional parameters.

```
public boolean exportMetaData(
        String projectName,
        String appName,
        String elementName,
        String typeName,
        String exportFileName,
        String logFileName,
        String characterSet,
        String fieldSeparator,
        String physicalName,
        String configParameter
        ) throws WBException, Exception
```

F

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 Oracle8*i*/9*i* tables, CLOB database columns, Advanced Queues, and other Oracle8*i*/9*i* database objects.

An XML document conforms to the Extensible Markup Language (XML) specification. XML allows developers to design their own customized markup languages 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 Oracle8*i*/9*i* 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.

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 Oracle8*i*/9*i* database objects. The Toolkit can store data in the following objects:

- Table or multiple tables in a target Oracle8*i*/9*i* 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.

How to Call 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 trigger 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 trigger 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 trigger.

The following example illustrates a script that can be used to implement an Warehouse Builder transformation which extracts data from an XML document stored in the file products.xml and loads it into the target table books.

```
begin
   wb_xml_load(
                  '<OWBXMLRuntime>'                                                    ||
                      '<XMLSource>'                                                    ||
                      '    <file>\ora817\GCCAPPS\products.xml</file>'                  ||
                      '</XMLSource>'                                                   ||
                      '<targets>'                                                      ||
                      '    <target XSLFile="\ora817\XMLstyle\GCC.xsl">books</target>'  ||
                      '</targets>'                                                     ||
                '</OWBXMLRuntime>');
end;
```

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 Oracle8*i* / 9*i* 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 F-20.

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 below (ORDERS) resides in a file named \ora817\examples\ex1.xml. This file name is relative to the Oracle8*i*/9*i* 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 below match the element names of the XML document described above.

```
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 below 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 Oracle8*i*/9*i* 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 trigger, the following actions occur:

1. The XML Toolkit reads the document ORDERS into a buffer and parses its data.
2. The Toolkit calls Oracle8*i*/9*i* 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 below (purchaseORDER) resides in the file \ora817\examples\ex2.xml. This file name is relative to the Oracle8*i*/9*i* 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 "The Target Table" on page F-6.

The Control File

The control file below 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 trigger, 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 Oracle8*i*/9*i* 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 F-7 describes the source document (purchaseORDER).

The Target Tables The column names for Purchase_Orders are described on page F-6; the column names for items are not required to understand the control file script.

The Control File The control file below 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 below, then the XML Toolkit divides the source document into as many parts as there are categories.

The Control File The control file below 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.

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 F-5.

The Control File The control file below 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 Oracle8*i*/9*i* 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 *Oracle8i/9i 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 F-5.

The Control File The control file below 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>'||
        '</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 Oracle8*i*/9*i* 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 *Oracle8i/9i 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 F-5.

The Control File The control file below 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>newOrdersAQ</AQName>'||
            '<ObjectType>xmlMessages</ObjectType>'||
            '<CLOBColumn>xmlEntry</CLOBColumn>'||
        '</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

This table describes all the elements used to define sources of XML documents and their respective attributes.

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.

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 *Oracle8i / 9i 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.

Name	Description
CLOB.table	Specifies the table or view that contains a CLOB column. Attributes: None.
CLOBColumn	Specifies the name of a CLOB column. Attributes: None.
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

This table 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.

Name	Description
targets	Specifies all the database targets. You specify individual targets using target elements. Attributes: None.

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 Oracle8*i* / 9*i* XML Reference. ignoreCase commitBatch rowTag (to set, use VARCHAR 'NULL') dateFormat batchSize keyColumnList updateColumnList

Runtime Control

This table describes elements that determine the runtime environment.

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 below 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 Oracle8*i* / 9*i* XML SDK software components. For more information, refer to the following:

- *Application Developer's Guide - Advanced Queuing*
- *Oracle8i* / 9*i Application Developer's Guide - XML*
- *Oracle8i* / 9*i XML Reference*

Web sites that contain information on the XML specification are:

http://www.w3.org/XML/

http://www.XML.com

G

OWB Public View Tables Management

Table G–1 ALL_IV_INSTALLATIONS

Column Name	Data Type	Description
installation_id	Number(9)	object ID
installation_name	Varchar2(255)	name of the project
business_name	Varchar2(4000)	display name for the project
description	Varchar2(4000)	
installed_version	Varchar2(2000)	
release	Varchar2(40)	
repository_model_version	Number(9)	
public_view_version	Char(6)	Version ID of the public SQL views
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–2 ALL_IV_PROJECTS

Column Name	Data Type	Description
project_id	Number(9)	object ID
project_name	Varchar2(255)	name of the project
business_name	Varchar2(4000)	display name for the project
description	Varchar2(4000)	description
version_label	Varchar2(255)	label representing the project version

Table G–2 ALL_IV_PROJECTS (Cont.)

Column Name	Data Type	Description
is_valid	Varchar2(7)	
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–3 ALL_IV_INFORMATION_SYSTEMS

Column Name	Data Type	Description
project_id	Number(9)	object ID
project_name	Varchar2(255)	name of the project that
information_system_id	Number(9)	Object ID of the information system
information_system_name	Varchar2(255)	Name of the information system
business_name	Varchar2(4000)	display name for the information system
description	Varchar2(4000)	description for the information system
product_type	Varchar2(255)	This is the application product type
system_type	Varchar2(255)	System type
version_label	Number(9)	
vendor	Varchar2(2000)	This is optional. This would return ‘Oracle’, ‘SAP’.
database_link	Varchar2(40)	this helps connect to the physical external data source. For data warehouse information systems this is optional. Typically, if a warehouse has been created based on the design of an external database schema, this would be populated for warehouses as well.
integrator_name	Varchar2(255)	Name of the integrator. For each Information System there is one and only one Integrator specified
is_valid	Varchar2(7)	
status	Varchar2(40)	The status of the information system. Warehouse Builder supports Production, Development and Quality Assurance
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Data Model

Table G–4 ALL_IV_SCHEMAS

Column Name	Data Type	Description
project_id	Number(9)	Project ID
project_name	Varchar2(255)	name of the project that the schema is in
schema_id	Number(9)	Identifier for the schema
schema_name	Varchar2(255)	name of the schema
business_name	Varchar2(4000)	display name for the schema
description	Varchar2(4000)	description
is_valid	Varchar2(7)	
information_system_id	Number(9)	The information system which this schema belongs to
information_system_name	Varchar2(255)	The name of the information system
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–5 ALL_IV_DIMENSIONS

Column Name	Data Type	Description
schema_id	Number(9)	Owner (schema) of the dimension
schema_name	Varchar2(255)	Name of the schema
dimension_id	Number(9)	Object ID of the dimension
dimension_name	Varchar2(255)	Name of the dimension
business_name	Varchar2(4000)	display name
plural_name	Varchar2(40)	plural name for the dimension used for reporting purposes
description	Varchar2(4000)	Description for dimension
is_valid	Varchar2(7)	
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–6 ALL_IV_DIM_LEVELS

Column Name	Data Type	Description
dimension_id	Number(9)	ID of the dimension
dimension_name	Varchar2(255)	Name of the dimension
level_id	Number(9)	object ID
level_name	Varchar2(255)	Level name. Unique within a dimension
business_name	Varchar2(4000)	display name
description	Varchar2(4000)	Description for level
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–7 ALL_IV_DIM_LEVEL_ATTRIBUTES

Column Name	Data Type	Description
level_id	Number(9)	ID of the dimension level
level_name	Varchar2(255)	Name of the dimension level
attribute_id	Number(9)	Object ID
attribute_name	Varchar2(255)	Unique within a dimension level
business_name	Varchar2(4000)	display name
description	Varchar2(4000)	Description for dimension attribute
position	Number(9)	position of the level attribute within a level
data_type	Varchar2(255)	Data type of the level attribute
length	Number(9)	length of the attribute
precision	Number(9)	precision of the attribute
scale	Number(9)	scale of the attribute
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–8 ALL_IV_DIM_HIERARCHIES

Column Name	Data Type	Description
dimension_id	Number(9)	ID of the dimension
dimension_name	Varchar2(255)	Name of the dimension
hierarchy_id	Number(9)	Object ID
hierarchy_name	Varchar2(255)	Hierarchy name

Table G–8 ALL_IV_DIM_HIERARCHIES(Cont.)

Column Name	Data Type	Description
business_name	Varchar2(4000)	display name
description	Varchar2(4000)	Description for the hierarchy
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–9 ALL_IV_DIM_HIERARCHY_LEVELS

Column Name	Data Type	Description
level_use_id	Number(9)	Object ID that represents a particular use of a level within a hierarchy
hierarchy_id	Number(9)	ID of the hierarchy
hierarchy_name	Varchar2(255)	Hierarchy name
level_id	Number(9)	ID of the level
level_name	Varchar2(255)	Name of the level
position	Number	Hierarchical position within this hierarchy, position 1 being the most detailed

Table G–10 ALL_IV_CUBES

Column Name	Data Type	Description
schema_id	Number(9)	schema of the cube
schema_name	Varchar2(255)	name of the schema
cube_id	Number(9)	object ID of the cube
cube_name	Varchar2(255)	Name of the cube
business_name	Varchar2(4000)	display name
description	Varchar2(4000)	Description for cube
is_valid	Varchar2(7)	
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–11 ALL_IV_CUBE_MEASURES

Column Name	Data Type	Description
cube_id	Number(9)	ID of the cube
cube_name	Varchar2(255)	Name of the cube
measure_id	Number(9)	object ID of the measure
measure_name	Varchar2(255)	Name of the measure
business_name	Varchar2(4000)	display name
description	Varchar2(4000)	Description for measure
position	Number(9)	position of the measure within the cube
data_type	Varchar2(255)	data type of the measure
length	Number(9)	length of the attribute
precision	Number(9)	precision of the attribute
scale	Number(9)	scale of the attribute
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–12 ALL_IV_CUBE_MEASURE_DIM_USES

Column Name	Data Type	Description
cube_id	Number(9)	Object ID
cube_name	Varchar2(255)	Name of the cube
measure_id	Number	ID of the measure
measure_name	Varchar2(255)	Name of the measure
dimension_id	Number	ID of the dimension
dimension_name	Varchar2(255)	name of dimension
dimension_alias	Varchar2(255)	alias of dimension

Table G–13 ALL_IV_TABLES

Column Name	Data Type	Description
schema_id	Number(9)	Owner of the table
schema_name	Varchar2(255)	Name of the schema
table_id	Number(9)	object ID of the table
table_name	Varchar2(255)	Name of the table
business_name	Varchar2(4000)	Logical - business name for this object

Table G–13 ALL_IV_TABLES(Cont.)

Column Name	Data Type	Description
description	Varchar2(4000)	description
is_valid	Varchar2(7)	
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–14 ALL_IV_VIEWS

Column Name	Data Type	Description
schema_id	Number(9)	Owner of the view
schema_name	Varchar2(255)	Name of the schema
view_id	Number(9)	object ID of the view
view_name	Varchar2(255)	name of the view
business_name	Varchar2(4000)	Logical - business name for this object
description	Varchar2(4000)	description
is_valid	Varchar2(7)	
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–15 ALL_IV_MATERIALIZED_VIEWS

Column Name	Data Type	Description
schema_id	Number(9)	Owner of the view
schema_name	Varchar2(255)	Name of the schema
view_id	Number(9)	object ID of the view
view_name	Varchar2(255)	name of the view
business_name	Varchar2(4000)	Logical - business name for this object
description	Varchar2(4000)	description
is_valid	Varchar2(7)	
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–16 ALL_IV_SEQUENCES

Column Name	Data Type	Description
schema_id	Number(9)	Owner of the sequence table
schema_name	Varchar2(255)	Name of the schema
sequence_id	Number(9)	Object ID of the key
sequence_name	Varchar2(255)	Name of the sequence
business_name	Varchar2(4000)	Display name **Note:** This will more then likely be equal to the name of the sequence.
description	Varchar2(4000)	description
is_valid	Varchar2(7)	
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–17 ALL_IV_KEYS

Column Name	Data Type	Description
entity_id	Number(9)	ID of the entity that this key is attached to
entity_type	Varchar2(4000)	
entity_name	Varchar2(255)	Name of the entity that the key is attached to
key_id	Number(9)	
key_name	Varchar2(255)	Name of the foreign key
business_name	Varchar2(4000)	Display name of the key
description	Varchar2(4000)	description
schema_id	Number(9)	Owner of the constraint
schema_name	Varchar2(255)	Name of the schema
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–18 ALL_IV_FOREIGN_KEYS

Column Name	Data Type	Description
entity_id	Number(9)	ID of the entity that this key is attached to
entity_type	Varchar2(4000)	
entity_name	Varchar2(255)	Name of the entity that the key is attached to

Table G–18 ALL_IV_FOREIGN_KEYS(Cont.)

Column Name	Data Type	Description
foreign_key_name	Varchar2(255)	Name of the foreign key
foreign_key_id	Number(9)	Object ID
business_name	Varchar2(4000)	Display name of the key
description	Varchar2(4000)	description
key_id	Number(9)	Id of the key that this foreign key is pointing to
key_name	Varchar2(255)	Name of the key that this foreign key is pointing to
schema_id	Number(9)	Owner of the constraint
schema_name	Varchar2(255)	Name of the schema
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–19 ALL_IV_KEY_COLUMN_USES

Column Name	Data Type	Description
key_id	Number(9)	Object ID of the key
key_type	Varchar2(11)	
key_name	Varchar2(255)	Name of the foreign key
column_id	Number(9)	ID of the column
column_name	Varchar2(255)	Name of the key column
position	Number(9)	Ordinal position of the key column within the key.

Table G–20 ALL_IV_COLUMNS

Column Name	Data Type	Description
entity_id	Number(9)	ID of the entity that this key is attached to
entity_type	Varchar2(4000)	
entity_name	Varchar2(255)	Name of the entity that the key is attached to
column_id	Number(9)	Object ID of the column
column_name	Varchar2(255)	Name of the column
business_name	Varchar2(4000)	Logical - business name for this object
description	Varchar2(4000)	description
position	Number(9)	position of the column within the table

Table G–20 ALL_IV_COLUMNS(Cont.)

Column Name	Data Type	Description
data_type	Varchar2(255)	Data type of the column
length	Number(9)	length of the column
precision	Number(9)	precision of the column
scale	Number(9)	scale of the column
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Implementation Model

Table G–21 ALL_IV_DIM_IMPLS

Column Name	Data Type	Description
implementation_id	Number(9)	Object ID of the dimension implementation
item_id	Number(9)	Object ID of the dimensional item
item_type	Char(15)	Type of item
item_name	Varchar2(255)	Name of the dimensional item
dimension_id	Number(9)	ID of the dimension
dimension_name	Varchar2(255)	Name of the dimension
column_id	Number(9)	ID of the column
column_name	Varchar2(255)	object ID
position	Number(9)	e.g. Position of the level attribute column. Note that Oracle8 attribute is single column.
table_id	Number(9)	Object ID of the table
table_name	Varchar2(255)	Name of the table that owns the column

Table G–22 ALL_IV_CUBE_IMPLS

Column Name	Type	Description
implementation_id	Number(9)	Object ID of the cube map Note: Exception to the naming convention rules: No map_name column exists.
item_id	Number(9)	Object ID of the cube item
item_type	Varchar2(18)	Type of item

Table G–22 ALL_IV_CUBE_IMPLS(Cont.)

Column Name	Type	Description
item_name	Varchar2(255)	Name of the cube item
cube_id	Number(9)	ID of the cube
cube_name	Varchar2(255)	Name of the cube
dimension_id	Number	ID of the dimension that CUBE_DIM_USE is pointing to
dimension_name	Varchar2(255)	Name of the dimension
dimension_alias	Varchar2(255)	Alias of the dimension
column_id	Number	ID of the column
column_name	Varchar2(255)	object ID
position	Number	e.g. Position of the level attribute column. Note that Oracle8 attribute is single column.
table_id	Number(9)	Object ID of the table
table_name	Varchar2(255)	Name of the table that owns the column or foreign key
foreign_key_id	Number	ID of the foreign key
foreign_key _name	Varchar2(255)	object ID
dim_implementation_id	Number	Object ID of the dimension map that this foreign key relies on (map dependency from the CWM model)

Flat File/Record Model

Table G–23 ALL_IV_FILES

Column Name	Type	Description
information_system_ID	Number(9)	Object ID of the information system
information_system_name	Varchar2(255)	Name of the information system
file_id	Number(9)	Object ID of the file
file_name	Varchar2(255)	Name of the file
business_name	Varchar2(4000)	display name
description	Varchar2(4000)	description
is_valid	Varchar2(7)	

Table G–23 ALL_IV_FILES(Cont.)

Column Name	Type	Description
record_classifier_position	Number(9)	When a file consists of multiple record types, each row of information read from the file contains a field that encodes the record type of the data row. This points to the position of that field.
record_classifier_length	Number(9)	This returns the length of the field that acts as record classifier.
record_size	Number(9)	This indicates the maximum size of a record for non delimited files.
n_physical_records_in_logical	Number(9)	The number of physical records that make up a logical record. This is only used where a fixed number of physical records make up a logical record.
Continuation_at_end	Char(1)	If the continuation string is given, this indicates when the continuation string is found at the end of the physical record. If false the continuation string will be at the start.
Continuation_delimiter	Varchar2(40)	When the number of physical records per logical record is variable, this indicates how a continuation record is identified. Note that this is more like a physical record delimiter when compared to recordDelimiter which is logical.
is_self_describing	Char(1))	True if fields in the first record contain column names applicable to subsequent records. Source from hasFieldNamesInFirstRow (will be added to Warehouse Builder).
is_fixed_width	Char(1)	Indicates that all records are of the same fixed length.
record_delimiter	Varchar2(40)	The string that delimits a record. If this is not set then the RecordSize should be used to determine the end of a record.
text_start_delimiter	Varchar2(40)	The delimiter of a text string in the record such as a quote.
text_end_delimiter	Varchar2(40)	
updated_on	Date	The time the object was last modified.
created_on	Date	The time this object was created.

Table G–24 ALL_IV_RECORDS

Column Name	Type	Description
file_id	Number(9)	Object ID of the file
file_name	Varchar2(255)	Name of the file
record_id	Number(9)	Object ID of the record
record_name	Varchar2(255)	Name of the record
business_name	Varchar2(4000)	display name
description	Varchar2(4000)	description
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–25 ALL_IV_FIELDS

Column Name	Type	Description
record_id	Number(9)	Object ID of the record
record_name	Varchar2(255)	Name of the record
field_id	Number(9)	ID for the field
field_name	Varchar2(255)	Name of the field
business_name	Varchar2(4000)	display name
description	Varchar2(4000)	description
position	Number(9)	position of the field within the record
data_type	Varchar2(255)	Data type of the field
length	Number(9)	length of the field
precision	Number(9)	precision of the field
scale	Number(9)	scale of the field
picture	Varchar2(40)	Picture clause
sign_type	Number(9)	The type of sign
usage	Varchar2(40)	How data is stored internally
occurs	Number(9)	
structure _id	Number(9)	This is a foreign key to the parent field (structure)
structure_name	Varchar2(255)	Name of the parent field (structure)
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Impact Analysis/Data Lineage Model

Table G–26 ALL_IV_ALL_OBJECTS

Column Name	Type	Description
object_id	Number(9)	ID of the object
object_type	Varchar2(4000)	Type of the object
object_name	Varchar2(255)	Name of the object
business_name	Varcahr2(4000)	Display name for the object
context_name	Varcahr2(4000)	The fully qualified name for the object. For example a table ORDERS in schema OP in project MyProject would be MyProject.OP.ORDERS
description	Varchar2(4000)	description
parent_object_id	Number	Object ID of the parent
parent_object_type	Varchar2(4000)	Type of the parent object
parent_object_name	Varchar2(4000)	Name of the parent object. For example, Table is considered to be parent of Column, Dimension is considered to be parent of Level, and Level Attribute, Cube is considered to be parent of Measures, EntityMap is considered to be parent of ItemMaps, etc.
is_valid	Varchar2(7)	
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–27 ALL_IV_IMPACT_DEPENDENTS

Column Name	Type	Description
object_id	Number	Object ID of the object whose dependents are listed
object_type	Varchar2(4000)	Type of the object
object_name	Varchar2(255)	Name of the object.
business_name	Varchar2(4000)	display name of the object
context_name	Varcahr2(4000)	The fully qualified name for the object. For example a table ORDERS in schema OP in project MyProject would be MyProject.OP.ORDERS
description	Varchar2(4000)	description
parent_object_id	Number	Object ID of the parent

Table G–27 ALL_IV_IMPACT_DEPENDENTS(Cont.)

Column Name	Type	Description
parent_object_type	Varchar2(4000)	Type of the parent object
parent_object_name	Varchar2(4000)	Name of the parent object. For example, Table is considered to be parent of Column, Dimension is considered to be parent of Level, and Level Attribute, Cube is considered to be parent of Measures, EntityMap is considered to be parent of ItemMaps.
dep_object_id	Number	ID of the dependent object
dep_object_type	Varchar2(4000)	type of the dependent object
dep_object_name	Varchar2(255)	Name of the dependent object
dep_object_business_name	Varchar2(4000)	Display name for the dependent object
dep_object_context_name	Varchar2(4000)	Display name for the dependent object
dep_expression	Varchar2(4000)	If a dependent object has an expression, it will be returned here. For example, if a column x is the result of a function 'h=g+1' and x is wired to h, when you get the dependents of x, you will get h together with the 'h=g+1' expression In addition, if the dependent object is an item (e.g.'column'), where an entity level expression (such as a filter or join condition on the stage component) applies, this expression will also be returned with each item (e.g. stage component parameter) that is aggregated by that entity (e.g. stage component) **Suggestion**: that the expressions coming from the entities are prefixed by the type of expression. for example: "FILTER:d>1'. This way, we can differentiate what immediately applies to the item and what comes from the parent entity. **Strong Recommendation**: Replace %1 %2… with the actual argument names.
dep_object_parent_id	Number	ID of the dependent object
dep_object_parent_type	Varchar2(4000)	type of the dependent object
dep_object_parent_name	Varchar2(4000)	Name of the dependent object
dep_object_operator_type	Varchar2(4000)	Name of the dependent object

Table G–28 ALL_IV_LINEAGE_DEPENDENTS

Column Name	Type	Description
object_id	Number	Object ID of the object whose dependents are listed
object_type	Varchar2(4000)	Type of the object
object_name	Varchar2(255)	Name of the object
business_name	Varchar2(4000)	display name of the object
context_name	Varcahr2(4000)	The fully qualified name for the object. For example a table ORDERS in schema OP in project MyProject would be MyProject.OP.ORDERS
description	Varchar2(4000)	description
parent_object_id	Number	Object ID of the parent
parent_object_type	Varchar2(4000)	Type of the parent object
parent_object_name	Varchar2(4000)	Name of the parent object. For example, Table is considered to be parent of Column, Dimension is considered to be parent of Level, and Level Attribute, Cube is considered to be parent of Measures, EntityMap is considered to be parent of ItemMaps, etc.
dep_object_id	Number	ID of the dependent object
dep_object_type	Varchar2(4000)	type of the dependent object
dep_object_name	Varchar2(255)	Name of the dependent object
dep_object_business_name	Varchar2(4000)	Display name for the dependent object
dep_object_context_name	Varchar2(4000)	
dep_expression	Varchar2(4000)	If a dependent object has an expression, it will be returned here. For example, if a column x is the result of a function 'h=g+1' and x is wired to h, when you get the dependents of x, you will get h together with the 'h=g+1' expression. In addition, if the dependent object is an item (e.g.'column'), where an entity level expression (such as a filter or join condition on the stage component) applies, this expression will also be returned with each item (e.g. stage component parameter) that is aggregated by that entity (e.g. stage component) **Suggestion:** that the expressions coming from the entities are prefixed by the type of expression. for example: "FILTER:d>1'. This way, we can differentiate what immediately applies to the item and what comes from the parent entity. **Strong Recommendation:** Replace %1 %2… with the actual argument names.
dep_object_parent_id	Number	ID of the dependent object

Table G–28 ALL_IV_LINEAGE_DEPENDENTS(Cont.)

dep_object_parent_type	Varchar2(4000)	type of the dependent object
dep_object_parent_name	Varchar2(4000)	Name of the dependent object
dep_object_operator_type	Varchar2(4000)	

Classification Model

Table G–29 ALL_IV_CLASSIFICATIONS

Column Name	Type	Description
information_system_id	Number(9)	Object ID of the information system
information_system_name	Varchar2(255)	Name of the information system
classification_id	Number(9)	Object ID of the catalog
classification_name	Varchar2(255)	Name of the catalog. May be null if the root of the catalog tree.
business_name	Varchar2(4000)	display name
description	Varchar2(4000)	description
classification_type	Varchar2(14)	Type of the classification
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–30 ALL_IV_CLASSIFICATION_ITEMS

Column Name	Type	Description
classification_id	Number	Object ID of the classification.
classification_name	Varchar2(255)	name of the classification
classified_object_id	Number(9)	ID of the object that is classified
classified_object_type	Varchar2(4000)	Type of the object
classified_object_name	Varchar2(255)	Name of the object classified

Expression & Function Model

Table G–31 ALL_IV_FUNCTION_LIBRARIES

Column Name	Type	Description
information_system_id	Number(9)	ID of the information system
information_system_name	Varchar2(255)	Name of the information system
function_library_id	Number(9)	Unique ID of the function library
function_library_name	Varchar2(255)	Name of the function library
business_name	Varchar2(4000)	Display name
description	Varchar2(4000)	description
is_valid	Varchar2(7)	
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–32 ALL_IV_FUNCTIONS

Column Name	Type	Description
function_library_id	Number(9)	ID of the function library
function_library_name	Varchar2(255)	Name of the function library.
function_id	Number(9)	Unique ID of the function
function_name	Varchar2(255)	Name of the function. Note that this name by itself may not be unique across the project. Use signature to display a unique function signature.
business_name	Varchar2(4000)	Display name. Note that this name by itself may not be unique across the project. Use signature to display a unique function signature.
description	Varchar2(4000)	description
signature	Varchar2(4000)	signature string.
function_type	Varchar2(40)	Type of the function
is_valid	Varchar2(7)	
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–33 ALL_IV_FUNCTION_PARAMETERS

Column Name	Type	Description
function_id	Number(9)	Unique ID of the function
function_name	Varchar2(255)	name of the function
parameter_id	Number(9)	ID of the parameter
parameter_name	Varchar2(255)	Name of the parameter
business_name	Varchar2(4000)	display name
description	Varchar2(4000)	Description
position	Number(9)	Position of the parameter within function
data_type	Varchar2(255)	Data type of the parameter
length	Number(9)	length of the parameter
precision	Number(9)	Precision
scale	Number(9)	scale of the parameter
parameter_type	varchar2(2000)	Whether the parameter is input, output or input_output
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–34 ALL_IV_FUNCTION_IMPLS

Column Name	Type	Description
function_id	Number(9)	Unique ID of the function
function_name	Varchar2(255)	name of the function
function_implementation_id	Number(9)	ID of the parameter
unction_implementation_name	Varchar2(255)	Name of the parameter
business_name	Varchar2(4000)	display name
description	Varchar2(4000)	Description
language	Varchar2(255)	Implementation language
script	Varchar2(4000)	The text of the script
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Configuration & Deployment Model

Table G–35 ALL_IV_OBJECT_CONFIGURATIONS

Column Name	Type	Description
configured_object_id	Number(9)	ID of the configured object
configured_object_type	Varchar2(4000)	Type of the object
configured_object_name	Varchar2(255)	Name of the configured object
physical_name	Varchar2(255)	Name of the physical object
configuration_parameter_ID	Number(9)	Unique identifier for the configuration structure
configuration_parameter_name	Varchar2(4000)	Name of the configuration structure
configuration_parameter_type	Varchar2(23)	Name of the configuration structure
business_name	Varchar2(4000)	Display name for the configuration parameter
description	Varchar2(4000)	description for the configuration parameter
position	Number(9)	The position of the parameter within the parameter group
argument	Varchar2(2000)	The actual value assigned for a configuration item (parameter). This is of type string.
complex_argument	Varchar2(4000)	
group_name	Varchar2(64)	Parent configuration structure identifier. This may be null.
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Transformation/Mapping Model

Table G–36 ALL_IV_XFORM_MAPS

Column Name	Type	Description
information_system_id	Number(9)	ID of the parent information system
information_system _name	Varchar2(255)	name of the parent information system
map_id	Number(9)	ID of the map
map_name	Varchar2(255)	name of the map
business_name	Varchar2(4000)	display name
description	Varchar2(4000)	description

Table G–36 ALL_IV_XFORM_MAPS(Cont.)

Column Name	Type	Description
composite_map_component_id	Number(9)	ID of the root map component
composite_map_component_name	Varchar2(255)	Name of the root map component
is_valid	Varchar2(7)	
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–37 ALL_IV_XFORM_MAP_COMPONENTS

Column Name	Type	Description
map_id	Number(9)	ID of the map
map_name	Varchar2(255)	Name of the map
map_component_id	Number(9)	ID of the map component
map_component_name	Varchar2(255)	Name of the map component
business_name	Varchar2(4000)	Display name
description	Varchar2(4000)	Description
operator_type	Varchar2(4000)	Type of the operator
composite_component_id	Number(9)	Parent transformation map component ID. This is a foreign key to self.
composite_component_name	Varchar2(255)	Name of the parent transformation map component
data_entity_id	Number(9)	Unique ID of the object that is passed as argument
data_entity_type	Varchar2(4000)	Type of the object
data_entity_name	Varchar2(255)	Name of the object
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–38 ALL_IV_XFORM_MAP_PROPERTIES

Column Name	Type	Description
map_component_id	Number(9)	ID of the map component
map_component_name	Varchar2(255)	Name of the map component
property_id	Number(9)	ID of the property

Table G–38 ALL_IV_XFORM_MAP_PROPERTIES(Cont.)

Column Name	Type	Description
property_name	Varchar2(255)	Name of the property
business_name	Varchar2(4000)	Display name
description	Varchar2(4000)	Description
property_group_name	Varchar2(255)	Group name
property_value	Varchar2(255)	Value of the property

Table G–39 ALL_IV_XFORM_MAP_PARAMETERS

Column Name	Type	Description
map_component_id	Number(9)	ID of the map component
map_component_name	Varchar2(255)	Name of the map component
parameter_id	Number(9)	ID of the map parameter
parameter_name	Varchar2(255)	Name of the parameter
business_name	Varchar2(4000)	Display name
description	Varchar2(4000)	Description
map_id	Number(9)	ID of the transformation map
map_name	Varchar2(255)	Name of the map
parameter_group_name	Varchar2(255)	Name of the parameter group
parameter_group_id	Number(9)	
parameter_type	varchar2(4000)	Whether the parameter is input, output or input_output
Position	Number(9)	Position of the parameter within the mapping
data_type	Varchar2(2000)	Data type of the parameter
transformation_expression	Varchar2(4000)	Expression that defines the transformation **NOTE:** The expression string should be parsed, and the tokens has to be replaced with real object names before presenting here.
data_item_id	Number(9)	unIque ID of the object that is passed as argument
data_item_type	Varchar2(2000)	type of the data object
data_item_name	Varchar2(255)	Name of the data object
source_parameter_id	Number(9)	unique ID of the object that is passed as argument

Table G–39 ALL_IV_XFORM_MAP_PARAMETERS(Cont.)

Column Name	Type	Description
Source_parameter_name	Varchar2(255)	Name of the source parameter object
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Runtime Public Views

Table G–40 ALL_IV_PROCESS

Column Name	Type	Description
process_id	Number	
process_name	Varchar2(4000)	
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–41 ALL_IV_PROCESS_RUN

Column Name	Type	Description
process_id	Number	FK to ALL_IV_PROCESS.process_id
process_run_id	Number	ID of the process run
start_time	Date	The time the process was started
end_time	Date	The time the process was completed
status	Varchar2(4000)	The status of the process. Could be 'RUNNING', 'COMPLETE', 'FAILURE'
number_records_selected	Number	Number of records selected
number_records_inserted	Number	Number of records inserted
number_records_updated	Number	Number of records updated
number_records_deleted	Number	Number of records deleted
number_records_discarded	Number	Number of records discarded
number_errors	Number	Number of errors in the process run
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–42 ALL_IV_MAP_RUN

Column Name	Type	Description
Process_run_id	Number	FK to ALL_IV_PROCESS_RUN.process_run_id
map_id	Number	not currently used
map_name	Varchar2(4000)	name of the map.
map_run_id	Number	ID of the map run
Start_time	Date	The time the map run was started
end_time	Date	The time the map run was completed
Status	Varchar2(4000)	The status of the map run. Could be 'RUNNING', 'COMPLETE', or 'FAILURE'.
number_records_selected	Number	Number of records selected. Sum of the number of records selected for all component steps.
number_records_inserted	Number	Number of records inserted. Sum of the number of records inserted for all component steps.
number_records_updated	Number	Number of records updated. Sum of the number of records updated for all component steps.
number_records_deleted	Number	Number of records deleted. Sum of the number of records deleted for all component steps.
number_records_discarded	Number	Number of records discarded. Sum of the number of records discarded for all component steps.
number_errors	Number	Number of errors in the map run. Sum of the errors for all component steps.
updated_on	Date	The time the object was last modified. The latest update time of the component steps.
Created_on	Date	The time this object was created. The earliest creation time of the component steps.

Table G–43 ALL_IV_RUN_TARGET

Column Name	Type	Description
map_run_id	Number	FK to ALL_IV_MAP_RUN.map_run_id
run_target_name	Varchar2(4000)	Name of the target
run_target_id	Number	ID of the run target entry
map_name	Varchar2(4000)	Name of the mapping step
number_records_inserted	Number	Number of records inserted
number_records_updated	Number	Number of records updated
number_records_deleted	Number	Number of records deleted

Table G–43 ALL_IV_RUN_TARGET(Cont.)

Column Name	Type	Description
number_errors	Number	Number of errors in this mapping step
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

Table G–44 ALL_IV_RUN_ERROR

Column Name	Type	Description
map_run_id	Number	FK to ALL_IV_MAP_RUN.map_run_id
run_target_name	Varchar(4000)	Name of the target
run_target_id	Number	ID of the run target entry. FK to ALL_IV_RUN_TARGET.run_target_id
run_error_id	Number	ID of the map run
error_number	Number	Number of the error
error_message	Varchar2(4000)	Text of the error message
updated_on	Date	The time the object was last modified
created_on	Date	The time this object was created

H

Warehouse Builder Bridges: Transfer Parameters and Considerations

Transfer Parameters

The Warehouse Builder Transfer Wizard allows you to export metadata 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 and import the metadata into Oracle warehousing tools, such as Oracle Discoverer, Oracle Express, and Oracle9*i* OLAP Server.

When you use the Warehouse Builder Transfer Wizard, you are required to enter transfer parameter information based upon the metadata source or target you have selected. The following tables list details about the transfer parameters you need to provide for each data source or target type.

Table H–1 Transfer Parameters for OMG CWM FIle Import

Transfer Parameter Name	Description
OMG CWM Input File	Name of the file where you want to store the imported metadata in the Warehouse Builder repository. Click "..." to browse for the file location.
Warehouse Builder Project	Warehouse Builder name for the imported project—this project will be located under the Projects node within Warehouse Builder.
Name Matching Mode	Name matching mode you want to utilize during the import; physical name mode is the default.
Import Mode	Import mode: create, replace, update, incremental update.

Table H–1 Transfer Parameters for OMG CWM FIle Import (Cont.)

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 H–2 Transfer Parameters for Importing from CA ERwin

Transfer Parameter Name	Description
Input Model File	Location of the ERwin Input file. Click "**...**" to browse for the file location.
Warehouse Builder Project	Warehouse Builder name for the imported project—this project will be located under the Projects node within Warehouse Builder.
Warehouse Builder Model Name	Name of the .erx file you are importing from ERwin.
Name Matching Mode	Name matching mode you want to utilize during the import. Physical name mode is the default.
Import Mode	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 H–3 Transfer Parameters for importing from Powersoft PowerDesigner

Transfer Parameter Name	Description
Input Model File	Location of the PowerDesigner Input file. Click "**...**" to browse for the file location.
Warehouse Builder Project	Warehouse Builder name for the newly imported Project. This Project will be located under the Projects node within Warehouse Builder.
Warehouse Builder Model Name	Name of the model you are importing from ERwin into Warehouse Builder.
CDM Name Mapping	Choose whether you want to use the logical (NAME) name or physical (CODE) name for the Conceptual Data Model (CDM). Default is NAME.
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
Name Matching Mode	Name matching mode you want to utilize during the import. Physical name mode is the default.
Import Mode	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 H–4 Transfer Parameters when OMG CWM is selected as the target

Transfer Parameter Name	Description
Warehouse Builder Exported Business Areas	Select the Business Areas you want to export from Warehouse Builder. The default is All Business Areas; you can select individual Business Areas from the drop-down list.
OMG CWM Output File	The path and filename for the OMG CWM output 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. .

Table H–5 Transfer Parameters when Discoverer is selected as target

Transfer Parameter Name	Parameter Description
Warehouse Builder Exported Business Areas	Select the Business Areas you want to export out of Builder from the drop-down menu; the default is All Business Areas.
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 fact 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 H–5 Transfer Parameters when Discoverer is selected as target (Cont.)

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.

Table H–6 Transfer Parameters when Express is selected as target

Transfer Parameter Name	Parameter Description
Warehouse Builder Exported Business Areas	From the drop-down list, select the business areas you want to export out of Warehouse Builder. The default is All Business Areas.
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.

Table H–6 Transfer Parameters when Express is selected as target (Cont.)

Transfer Parameter Name	Parameter Description
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 H–7 Transfer Parameters for Importing from Oracle 9i OLAP server

Transfer Parameter Name	Parameter Description
User Name	User name for connecting to the Oracle9*i* OLAP server.
Password	Password for connecting to 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.
Measure Folder	The measure folder to import the OLAP definitions from. The measure folder will be used to drive the information imported. So for a measure, the associated fact and dimensions will be imported.
Project	Name for the imported project.
Default module type for relational schemas.	Select Warehouse module, since you will import OLAP definitions.
Process OLAP Physical Representation	True or False. Select whether to read the physical mapping of the fact to table, and dimension to table. There are certain structures which Warehouse Builder does not currently support, such as snowflake dimension schemas, or facts based on views, so utilizing the physical mapping is not always possible. If it is not possible or False is selected, the import will generate a default physical representation.

Table H–7 Transfer Parameters for Importing from Oracle 9i OLAP server (Cont.)

Transfer Parameter Name	Parameter Description
Name matching mode	<your standard description goes here>
Import mode	<your standard description goes here>
Log Level	<your standard description goes here>

Table H–8 Transfer Parameters when Oracle 9i OLAP server is selected as target

Transfer Parameter Name	Parameter Description
Warehouse Builder Exported Business Areas	<your standard description goes here>
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	<your standard description goes here>

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 appendix provides instructions for preparing to transfer 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 business areas. Before you export metadata, you must create business areas within the Warehouse Builder repository. See Chapter 4, "Defining Dimensional Targets" for instructions.

This section includes the following topics:

- Importing Metadata from an Object Management Group Common Warehouse Metamodel 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 3.1 and 4i
- 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 Common Warehouse Metamodel Standard System

Table H–9 lists the object conversion fom an OMG CWM file system to the Warehouse Builder repository.

Table H–9 Object Conversion for Import from OMG CWM into Warehouse Builder

Object in OMG CWM	Imported into Warehouse Builder
Package	Project
Schema	Module
Table	Table
Column	Columns
Foreign Key	Foreign Keys
Unique Constraint / Primary Key	Unique Keys
View	View

Table H–9 Object Conversion for Import from OMG CWM into Warehouse Builder

Object in OMG CWM	Imported into Warehouse Builder
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 H–10 lists the object conversion from ERwin to the Warehouse Builder repository.

Table H–10 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 H–11 lists the object conversion fom PowerDesigner to the Warehouse Builder repository.

Table H–11 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 H–11 Object Conversion for Import from PowerDesigner into Warehouse Builder (Cont.)

Object in PowerDesigner	Imported into Warehouse Builder
View	View
Column	Columns

Importing Metadata from Oracle9*i* OLAP server

Table H–12 lists the object conversion fom Oracle9*i* OLAP server to the Warehouse Builder repository.

Table H–12 Object Conversion for Import from Oracle9i OLAP server into Warehouse Builder

OLAP Object in Oracle9*i*	Imported into Warehouse Builder
Schema	Module
Cube	Fact
Measure	Measure
Dimension	Dimension
Level	Level
Hierarchy	Hierarchy
Attribute	LevelAttribute
MeasureFolder	BusinessArea

There are certain structures that are not supported in Warehouse Builder that are possible in the Oracle9*i* OLAP server. For example, snowflake modelling and facts based upon views are not supported. If you have OLAP definitions which use these structures, the logical definition of the objects can be imported into Warehouse Builder, but not the physical representation.

Exporting Metadata to Oracle Discoverer 3.1 and 4*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 H–13 lists the object conversion fom Warehouse Builder repository to Discoverer.

Table H–13 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
Fact	Folder
Fact Attributes	Item
Table	Folder
Columns	Item
Unique Keys	Key
View	Folder (View)
Columns	Item
Attributes flagged as Item Classes	Item Classes
Business Tree	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:

- Creating business areas 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 fact 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. Dimension roles are required in Discoverer 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 then 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 actual dummy table to real dimension association relies on any one of the contained columns. The dummy column to fact association is determined by making the column's physical name the same as the physical name of the foreign key constraint in the fact.

The following example illustrates how dimensional roles can be set up for exporting to Discoverer:

Defining Dimensions and Facts in Warehouse Builder

1. Create dimensions–DAY, PRODUCT and OPERATOR–in the normal way.
2. Create facts–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.**

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 fact 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 3.1 and 4*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. (Refer to the section "Configuring Warehouse Builder for Dimensional Reuse").

Dimension roles are required to make query building easier in Discoverer. When two tables have multiple joins between them, Discoverer asks which join to use the first time the two folders representing the tables are referred to in a query, and then always uses that join in further parts of the query. If the query requires different joins at different times, the only way to support this 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 fact, 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 fact, the original role is reused. For example, if a FLIGHTS fact was also using the ORDER_DATE role, it would reference the same folder as the one referenced by the SALES fact.

Dimensional reuse (two or more foreign key constraints pointing to the same dimension in Warehouse Builder) is denoted in the Discoverer navigation tree 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 default role above)

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 H–14 lists the object conversion from the Warehouse Builder repository to the OMG CWM standard.

Table H–14 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
Fact	Cube
Fact Attributes	Measure
Table	Base 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**. 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).
10. 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 (give it a name which reflects the project that you transferred). After you enter a name, 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 H–15 lists the object conversion from the Warehouse Builder repository objects to OLAP objects in Oracle9*i*.

Table H–15 Object Conversion for Export from Warehouse Builder into Oracle9i OLAP server

Object into Warehouse Builder	Exported OLAP Object
Fact	Cube
Measure	Measure
Dimension	Dimension
Level	Level
Hierarchy	Hierarchy
LevelAttribute	Attribute
BusinessArea	MeasureFolder

When the PL / SQL file is generated after a metadata export to the Oracle9*i* OLAP Server, the file can be deployed on your server by 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.

To create the OLAP short and long description attributes for a dimension, you must perform a best practice. The best practice is when creating a level attribute to edit the column name that implements the attribute. Add the suffix _SHORT_NAME to the column name to represent a short description attribute or add _LONG_NAME for a long description.

Glossary

additive

Describes a fact (or 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*.)

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.*

aggregate

Data that is aggregated by sum, average, or another aggregation function. For example, unit sales of a particular product could be summarized by day, month, quarter, and yearly sales.

ancestor

A value at any level above 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. (See also *descendant*, *hierarchy*, *level*.)

attribute

A descriptive characteristic of one or more levels. Attributes represent logical groupings that enable end users to select data based on like characteristics. In relational modeling, an attribute is defined as a characteristic of an entity. In Oracle9*i*, an attribute is a column in a dimension that characterizes elements of a single level.

child

A value at the level below 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. (See also *hierarchy*, *level*, *parent*.)

cleansing

The process of resolving inconsistencies and fixing the anomalies in source data, typically as part of the ETL process. (See also *ETL*.)

Common Warehouse Metamodel

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.

console

The main window of the Oracle Warehouse Builder application. It contains a menu bar, launcher, and navigator.

data source

A database, application, data definition source, or file that contributes data.

data mart

A data warehouse that is designed for a particular line of business, such as sales, marketing, or finance. In a dependent data mart, the data can be derived from an enterprise-wide data warehouse (as a dependent data mart). In an independent data mart, data can be collected directly from sources. (See also *data warehouse*.)

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 *ETL, OLAP.*)

ddl

Data Definition Language.

denormalize

The process of allowing redundancy in a table so that it can remain flat. (Contrast with *normalize*.)

derived fact (or measure)

A fact (or measure) that is generated from existing data using a mathematical operation or a data transformation. Examples include averages, totals, percentages, and differences.

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.

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.

dml statement

There are three types of data manipulation language, or dml, statements: insert, update, or delete.

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 *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*.)

drill up

To collapse the list of descendant values that are associated with a parent value in the hierarchy.

element

An object or process. For example, a dimension is an object, a mapping is a process, and both are elements.

ETL

Extraction, transformation, and load. ETL refers to the methods involved in accessing and manipulating source data and loading it into a data warehouse. The order in which these processes are performed varies. (See also *data warehouse, extraction, transformation*.)

editor

A window in Oracle Warehouse Builder used to define or edit objects and their relationships to each other.

extraction

The process of taking data out of a source as part of an initial phase of ETL. (See also *ETL*.)

fact table

A table in a star schema that contains facts. A fact table typically has two types of columns: those that contain facts and those that are foreign keys 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 *summary tables*). A fact table usually contains facts with the same level of aggregation.

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.

file-to-table mapping

Maps data from flat files to tables in the warehouse.

hierarchy

A logical structure that uses 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. (See also *dimension, level*.)

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.

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. (See also *hierarchy*.)

level value table

A database table that stores the values or data for the levels you created as part of your dimensions and hierarchies.

mapping

The definition of the relationship and data flow between source and target objects.

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 multidimensional 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.

model

An object that represents something to be made. A representative style, plan, or design. Metadata that defines the structure of the data warehouse.

module (source and target)

The metadata containers for source data, process data, and warehouse data.

navigator

A panel in the Oracle Warehouse Builder console that displays the objects for the active environment in a tree structure.

nonadditive

Describes a fact (or 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

Online analytical processing. OLAP functionality is characterized by dynamic, multidimensional 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.

parent

A value at the level above 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, hierarchy, level*.)

physical instance

A collection of related database objects that, when deployed, becomes a schema. Objects include tables, views, and other objects.

project

A project contains all design definitions and information needed by Warehouse Builder to build a warehouse. The project structure helps users organize their work.

row

A row is the basic unit for the processing of data in any mapping. A row has a structure and is defined by attributes, where each attribute is given a name and data type, and length, scale, and precision.

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.

schema

A collection of related database objects. Relational schemas are grouped by database user ID and include tables, views, and other objects. (See also *snowflake schema, star schema*.)

semi-additive

Describes a fact (or 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.

short name

The short name is a unique identifier that is used as the root name for all related objects created in Warehouse Builder.

snowflake schema

A type of star schema in which the dimension tables are partly or fully normalized. (See also *schema, star schema.*)

Software Library

Contains the integrators currently installed for Warehouse Builder. Accessed through the Administration environment.

source

A database, application, file, or other storage facility from which the data in a data warehouse is derived.

star schema

A relational schema whose design represents a multidimensional data model. The star schema consists of one or more fact tables and one or more dimension tables that are related through foreign keys. (See also *schema, snowflake schema.*)

subject area

A classification system that represents or 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. (See also *data mart.*)

table

A layout of data in columns.

target

Holds the intermediate or final results of any part of the ETL process. The target of the entire ETL process is the data warehouse. (See also *data warehouse, ETL.*)

transformation

The process of manipulating data. Any manipulation beyond copying is a transformation. Examples include cleansing, aggregating, and integrating data from multiple sources.

Transformation Library

Stores the reusable formulas for the transformation of data as it moves between source and target objects.

validation

The process of verifying metadata definitions and configuration parameters.

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.

Index

A

ABAP scripts, 8-32
access level, Oracle Workflow Builder client, 10-10
add objects with Metadata Import Utility, 11-7
administration environment, 2-7
administration tree, 2-7
agents
  transparent gateways, 5-4
aggregation, mixed levels, 4-6
aggregator operator, 7-25
analyze statistics and targets, 8-21
application
  generic file based, 5-14
Archive/Restore
  archiving a project, 11-47
  restoring a project, 11-50
  setting up preferences, 11-45
  using, 11-44
attribute sets
  creating, 3-16
  default, 6-4
Auto-Mapping dialog
  copy, 6-20
  Match by Name, 6-20
  Match by Position, 6-20

B

binding operators, 6-4
bound name, 8-11
bridges
  business areas, 4-39
  versions, 11-36
bridge-type attribute set, 3-16
bulk processing, 8-21
business area, 4-38
  bridges, 4-39
  Business Tree tab, 4-39
  creating a, 4-39
business tree
  business area, 4-38

C

character set
  flat file, 5-31
check constraint, 3-14
CHECK/INSERT, 8-11
cleansing data, 7-34
clipboard, 2-10
code viewer, 8-37
column
  attribute sets, 3-16
  default order, 3-11
  generated, 4-9
  reorder, 3-11, 3-12
commit, 2-25
commit frequency, 8-21
configuration
  flat file physical properties, 8-26
  flat file sources, 8-26
  flat file targets, 8-26
  loading properties, 8-15
  mapping, 8-9
  mapping operator attributes, 8-14
  mapping physical properties, 8-16
  mapping sources and targets, 8-21

mapping table operators, 8-9
Oracle Reports, 12-2
runtime parameters, 8-19
runtime parameters, SAP files, 8-32
SAP file physical properties, 8-32
table operator, 8-10
configuration parameters
dimension, 9-6
materialized views, 9-8
rules for export, 11-4
sequences, 9-10
warehouse module, 9-3
Workflow Queue Listener, 9-4
configuration properties
analyze statistics and targets, 8-21
attributes, 8-15
bound name, 8-11
bulk processing, 8-21
commit frequency, 8-21
conditional loading, 8-12
constraint management, 8-23
default audit level, 8-20
default operating mode, 8-19
default purge group, 8-21
delete (use for matching), 8-16
deployable, 8-19
Field Names in the First Row, 8-13
flat file operators, 8-13
hints, 8-22
insert (use for loading), 8-15
keys, 8-12
loading type, 8-11, 8-13
maximum number of errors, 8-21
partition exchange loading, 8-22
SQL*Loader parameters, 8-24
step type, 8-19
transformation operators, 8-14
update (operation), 8-16
update (use for loading), 8-16
update (use for matching), 8-16
configuring
data warehouse, 9-2
database links, 9-12
dimensions, 9-6
hash partitions, 9-21
materialized views, 9-8
new physical objects, 9-3
partitions, 9-17
physical instance, 9-2
range partitions, 9-21
range with hash subpartitions, 9-22
sequences, 9-10
views, 9-11
warehouse module, 9-3
connection information
non-Oracle database system, 5-3
Oracle Designer Repository, 3-3
remote function call (RFC), 5-16
SAP application, 5-16
source modules, 5-2
updating, 5-28
warehouse module, 3-3
connection type, 5-16
console
administration tree, 2-7
menu, 2-3
project tree, 2-5
toolbar, 2-3, 2-4
transformation tree, 2-6
tree view buttons, 2-3
utilities button, 2-3, 2-7
views, 2-5
wizard buttons, 2-3
constant operator, 7-9
constraint
change, 3-12
change UK to PK, 4-25
check, 3-14
DEFAULTIF and NULLIF, 5-32
fact table foreign key references, 4-3, 4-20
generated unique key constraints, 3-13
unique key constraint on levels, 4-5
view, 3-25
constraint management, 8-23
container
database sources, 5-5, 5-9
flat file sources, 5-13
flat files, 5-13
Oracle Designer Repository, 5-8
SAP application, 5-15

source module, 5-2
warehouse modules, 3-2
create definition
delimited flat file, 5-43
dimension, 3-7, 4-4
fact table, 4-19
flat file, 5-30
flat file sources, 5-30
hierarchy, 4-12
materialized view, 3-19
Oracle Workflow process, 10-10
transformation, 4-33
view, 3-23
creating
database links, 5-4, 12-48
diagram of source definitions, 5-29
dimensions, 3-7, 4-7
indexes, partitions, database links, 9-3
physical objects, 9-3
source modules, 5-5, 5-9
time dimension, 4-15
warehouse module, 3-2

D

data generator operator, 7-6
data source, 5-1
data transformation, 4-27, 7-1
data type
flat file fields, 5-32
portable, 5-32
data warehouse
configuration parameters, 9-2
database links
configuring, 9-12
creating, 5-4, 9-3
New Database Link dialog, 5-3
Oracle database system, 5-4
SAP, 5-16
source module, 5-2
warehouse module, 3-4
deduplicator operator, 7-24
default audit level, 8-20
default operating mode, 8-19
row based, 8-20
row based (target only), 8-20
set based, 8-19
set based fail over to row based, 8-20
set based fail over to row based (target only), 8-20
default purge group, 8-21
definition
attribute, 4-10
business area, 4-38
delimited file, 5-42
delimited flat file, 5-43
fixed-length file, 5-33
flat file, 5-30
materialized view, 3-19
transformation, 4-33
valid and invalid, 9-25
validate a set of definitions, 9-23
warehouse key column, 4-12
DELETE, 8-11
delete (use for matching), 8-16
DELETE / INSERT, 8-11
delimited file
definition, 5-42
deployable, 8-19
deploying
PL/SQL mappings, 9-29
scripts, 9-33
tables, 9-30
Designer Repository
import definitions into Warehouse Builder repository, 3-18, 4-26
desktop view, 3-23
diagram source definitions, 5-29
dimension
attribute definition, 4-10
basic definition, 4-3
create definition, 3-7, 4-4
define a level of aggregation, 4-9
denormalized dimension table, 4-6
dimension object, 4-3
generated scripts, 9-28
hierarchies, 4-11, 4-12
levels, 4-5
mapping, 6-5
New Dimension Wizard, 3-7, 4-4

products
create, 3-7, 4-7
property sheet, 3-10, 4-14
table property sheet, 3-10, 4-15
time dimension, 3-10, 4-15
update, 3-9, 4-12
validate example, 9-23, 9-24
dimension table
default column order, 3-11
physical aspects, 4-3
reordering the columns, 3-11
dimensions
creating, 3-7, 4-7
directives
Metadata Export Utility, 11-16
Metadata Import Utility, 11-18
Discoverer bridge
business areas, 4-39
display sets
default attribute sets, 6-4
display welcome page, 2-20
DISTINCT, 7-24
drawer, utilities, 2-7

E

editing
dimensions, 3-9, 4-12
invalid objects, 9-25
editor
dimension, 3-9, 4-12
materialized view, 3-23
object, 2-23
source module
generate diagram, 5-29
warehouse module, 3-6
environments
administration, 2-7
project, 2-5
export
log file, 11-39, 11-43
metadata export file, 11-3
metadata with Metadata Export Utility, 11-3
subset of objects, 11-3
exporting metadata, 11-39
Express bridge
business areas, 4-39
expression operator, 7-57
external process operator, 7-52
extract transformation, 8-2

F

fact table
create definition, 4-19
default column order, 3-11
define foreign key reference, 4-20
physical aspects, 4-3
property sheet, 4-23
reordering the columns, 3-12
update definition, 4-22
filter data with a transform, 4-33
Filter operator, 7-28
Find, 2-24
fixed-length file
definition, 5-33
flat file
character set, 5-31
configuration, 8-26
create definition, 5-30
data sample, 5-34
definition, 5-30
delimited
create format, 5-43
update format, 5-47
delimited format, 5-42
field constraint, 5-32
field mask, 5-32
field type, 5-32
fixed-length, 5-33
create format, 5-34
format, 5-38
generic file based application, 5-14
mapping sources, 8-26
mapping target, 8-26
multiple physical records per logical record, 5-31, 5-42
physical record length, 5-31
source module, 5-13
update format definition, 5-47

Flat File Sample Wizard, 5-30, 5-33
folder-in-use lock, 2-17

G

gateways
  transparent, 5-4
generated
  column name, 4-9
  unique key constraints, 3-13
generated scripts
  dimension, 9-28
  general description, 9-23
  mapping, 9-29
  materialized view, 9-29
  sequences, 9-30
  tables, 9-30
  types, 9-40
generating upgrade scripts, 9-40
generic file based application, 5-14
GROUP BY, 7-25
group by operation, 7-25

H

hash partitions, 9-17, 9-21
HAVING, 7-25
heterogeneous source, 5-2
hierarchies
  create definition, 4-12
  dimension, 4-11
hints, 8-22

I

ID column
  rename as warehouse key column, 4-12
import
  definition
    database source, 5-19
    database systems, 5-19
    Import Metadata Wizard, 5-23
    re-import, 5-23
  definitions into a warehouse module, 3-18, 4-26
  Metadata Import Utility, 11-7
  selected modules, 11-19
  validation rules, 11-20
Import Metadata Wizard, 5-33
  invoking from navigation tree, 5-50
  invoking from New Module Wizard, 5-18
importing metadata, 11-7, 11-9, 11-35
inbound reconcile, 8-2
indexes
  creating, 9-3
INI file for SAP Integrator, 5-17
init.ora, 5-5
input roles, 7-36
INSERT, 8-11
insert (use for loading), 8-15
INSERT/UPDATE, 8-11
integrators, 5-2
intermediate results, 8-37
intersect, 7-14

J

job dependencies, 10-12
job scripts, 10-15
joiner operator, 7-16

K

key lookup operator, 7-10
keys, 8-12

L

library
  transformation, 4-28
loading properties, 8-15
loading type, 8-11, 8-13
  check/insert, 8-11
  delete, 8-11
  delete/insert, 8-11
  insert, 8-11
  insert/update, 8-11
  merge, 8-12
  none, 8-11
  truncate/insert, 8-11
  update, 8-11

update/insert, 8-11
log file
export to OMG CWM, 11-39, 11-43
logical constraints
view, 3-25
logical name
creating, 2-20
maximum length, 2-21
requirements, 2-21
logon to a repository, 2-2

M

mapping, 6-1
adding or removing operator groups, 6-22
Auto-Mapping dialog, 6-19
binding operators, 6-4
code viewer, 8-37
connecting attribute groups, 6-19
create, 6-7
data flow operators, 6-16
definitions, 6-2
dimension, 6-5
edit DML Type, 8-10
extract transformation, 8-2
flat file sources, 8-26
flat file targets, 8-26
generate, 8-35
generated scripts, 9-29
intermediate results, 8-37
operator attribute groups, 6-3
operator attributes, 6-3
operator display set, 6-3
operator properties, 6-3
operator property inspector, 6-6
SAP source, 6-14
selecting flat file operators, 6-13
selecting operators, 6-10
transformations, 7-1
triggers, 8-2
validation, 8-33
Mapping Editor
about, 6-5
keyboard operations, C-10
menu bar, C-2
mouse operations, C-7
toolbar, C-5
Toolbox, C-5
mapping flat file operators
sources, 8-26
mapping input parameter operator, 7-46
mapping output parameter operator, 7-49
mapping sequence operator, 7-5
materialized view, 4-4
configuring, 9-8
create definition, 3-19
default column order, 3-11
generated scripts, 9-29
performance, 4-4
refreshing, 9-10
reordering the columns, 3-12
update definition, 3-23
materialized view editor, 3-23
maximum number of errors, 8-21
menu, 2-3
MERGE, 8-12
metadata
detailed reports, 12-17
implementation reports, 12-19
lineage and impact analysis diagrams, 12-21
lineage and impact analysis reports, 12-20
summary reports, 12-16
Metadata Export Utility
directives, 11-16
Metadata Import Utility
directives, 11-18
Metadata Loader
Export Utility, 11-3
Import Utility, 11-7
Loader Utilities in MS-DOS window, 11-15
metadata loader (MDL), 11-2
command line utility, 11-15
exporting metadata, 11-5
import options, 11-10
import searching, 11-7
importing, 11-7, 11-9
large files, 11-2
multiple user considerations, 11-2
migrating a repository, 11-2
minus, 7-14

mixed levels of aggregation, 4-6
module
  New Module Wizard, 5-16
  source module, 2-17, 5-2
    database definitions, 5-5, 5-9
    flat files, 5-13
  unique names in project, 2-20
  warehouse module, 3-2
    creating a business area, 4-39
module editor
  source module
    generate diagram, 5-29
  warehouse, 3-6
multi-user access, 2-17
  read-only mode, 2-18
  read / write mode, 2-18
  synchronization, 2-18

N

name
  creating, 2-20, 2-21
  default naming mode, 2-22
  policies, 2-20
  renaming an object, 2-21
name and address operator, 7-34
named attribute sets, 3-16
naming mode
  default naming mode, 2-22
  preferences, 2-22
naming policies, 2-20
navigation
  search tree, 2-24
  warehouse module navigation tree, 3-6
  wizards, 2-19
navigation tree, 2-3
New Database Link dialog, 5-3
New Fact Table Wizard, 4-19
New Materialized View Wizard, 3-19
New Module Wizard, 5-14, 5-16
New Time Dimension Wizard, 4-15
New Transform Wizard, 4-33
New View Wizard, 3-23
NONE, 8-11
non-Oracle database systems, 5-3

O

object
  children of first-class objects, 11-9
  import selected modules, 11-19
  import with Metadata Import Utility, 11-7
open_links, 5-5
Operator Property Inspector, 6-6
operators
  aggregator, 7-25
  constant, 7-9
  data generator, 7-6
  deduplicator, 7-24
  definition of, 6-2
  expression, 7-57
  external process, 7-52
  Filter, 7-28
  joiner, 7-16
  joiner, full outer join, 7-20
  key lookup, 7-10
  mapping flat files, 8-26
  mapping input parameter, 7-46
  mapping output parameter, 7-49
  mapping sequence, 7-5
  name and address, 7-34
  post-mapping process, 7-44
  pre-mapping, 7-42
  set operation, 7-14
  sorter, 7-31
  splitter, 7-21
  transformation, 7-54
optimizer hints as a transformation, 8-9
Oracle Designer, 9-12
  import definitions into Warehouse Builder repository, 3-18, 4-26
  source module, 5-8
Oracle Discoverer
  business areas, 4-39
Oracle Express
  business areas, 4-39
Oracle Portal, 12-15
Oracle Reports
  configuration, 12-2
Oracle Transparent Gateway, 5-3
Oracle Workflow

client access level, 10-10
define process, 10-10
Deployment Wizard, 10-5
server, 10-5
ORDER BY, 7-31
outbound reconcile, 8-6
output components, 7-37

P

parameter
schedule job with parameters, 10-15
transformation, 4-27
partition exchange loading, 8-22, 9-22
partitions
configuring, 9-17
creating, 9-3
hash, 9-17, 9-21
range, 9-17, 9-21
range with hash subpartitions, 9-22
performance
materialized views, 4-4
warehouse keys, 4-3
physical aspects
dimension table, 4-3
fact table, 4-3
physical name
creating, 2-20
syntax, 2-20, 2-21
uniqueness requirements, 2-20
post-mapping process operator, 7-44
predefined time dimension elements, 4-15
Preferences
Clipboard/Recycle, 2-10
display welcome page on all wizards, 2-20
naming mode, 2-22
utility, 2-8
add a utility, 2-9
remove a utility, 2-10
update a utility, 2-10
Warehouse Builder Browser, 12-2
pre-mapping process operator, 7-42
print report, 12-4
Project, 2-15
project
unique module names, 2-20
project environment, 2-5
project tree, 2-5
property sheet, 2-23
default column order, 3-11
dimension, 4-14
object, 3-10, 4-14
table, 3-10, 4-15
fact table, 4-23
materialized view, 3-23
table, 3-10

Q

query rewrite, 4-4

R

range partitions, 9-17, 9-21
with hash subpartitions, 9-22
RECNUM, 7-6
reconciliation
inbound, 8-2
outbound, 8-6
recycle bin, 2-10
reference materials
distributed database systems, 5-4
non-Oracle database systems, 5-4
Oracle8i Data Warehousing Guide, 4-2, 4-4
Ralph Kimball books, 4-2, 4-4
refreshing materialized view, 9-10
REMAINING_ROWS output group, 7-21
remote function call (RFC)
RFC connection, 5-16
SAP RFC connection, 5-17
rename warehouse object
view, 3-27
renaming, 2-21
reorder columns, 3-11
warehouse objects, 3-12
Reports
creating custom, 12-38
detailed, 12-17
implementation, 12-19
lineage and impact analysis, 12-20

lineage and impact analysis diagrams, 12-21
printing, 12-4
summary, 12-16
repository
logging on, 2-2
reserved words, B-2
ROW BASED, 8-20
ROW BASED (TARGET ONLY), 8-20
row sets
definition of, 1-7
rows
definition of, 1-7
rules
dimension object, 4-5
running scripts, 9-32
Runtime Audit Viewer, 10-22
purging runtime entries, 10-27
refreshing the view, 10-27
viewing by date range, 10-24
viewing the job audit, 10-17
viewing the job instance audit, 10-18
viewing the task audit, 10-19
viewing the task details audit, 10-21
runtime parameters, 8-19

S

sample
data sample for flat file, 5-34
SAP application
source module, 5-15
SAP file physical properties, 8-32
SAP Integrator
create definitions, 5-15
objects supported, 5-50, 5-51
required files, 5-17
SAPRFC.INI file, 5-17
search navigation tree, 2-24
secondary source
view, 3-25
semi-colon, 3-21
SEQUENCE, 7-7
sequences
configuring, 9-10
generated scripts, 9-30
services
heterogeneous services, 5-3
SET BASED, 8-19
SET BASED FAIL OVER TO ROW BASED, 8-20
SET BASED FAIL OVER TO ROW BASED (TARGET ONLY), 8-20
set operation operator, 7-14
intersect, 7-14
minus, 7-14
union, 7-14
union all, 7-14
short name
defining, 1-8
reserved words, B-2
software integrators, 5-2
SORTED INDEXES, 8-25
sorter operator, 7-31
source
data source, 5-1
source definition
update, 5-26
source module, 5-2
connection information, 5-2
SAP source, 5-16
create definition, 5-14
creating, 5-5, 5-9
database definitions, 5-5, 5-9
database links, 5-2
flat files, 5-13
import definitions, 5-19
Oracle Designer Repository, 5-8
SAP application, 5-15
Source Module Editor
generate diagram, 5-29
splitter operator, 7-21
SQL optimizer hints as a transformation, 8-9
SQL*Loader parameters, 8-24
star schema
basic example, 4-2
step type, 8-19
summarize data with a transformation, 7-25
synchronization, 2-18
syntax errors
extra semi-colon, 3-21
SYSDATE, 7-7

T

table
  attribute sets, 3-16
  default column order, 3-11
  generate and deploy, 9-30
  generated scripts, 9-30
  reordering the columns, 3-12
table properties
  dimension table, 3-10, 4-14
  fact table, 4-24
tablespace, 9-8
testing deployed scripts, 9-32
time dimension
  create, 3-10, 4-15
toolbar, 2-3, 2-4
  commit, 2-25
  property icon, 2-23
Transfer Wizard
  log file
    export, 11-39, 11-43
  version, 11-36
transformation, 7-1
  create definition, 4-33
  custom example, 4-33
  extract, 8-2
  filter data, 4-33
  group by operation, 7-25
  library, 4-28
  overview, 4-27
  parameters, 4-27
  unique names, 2-20
transformation operator, 7-54
transformation tree, 2-6
transparent
  agents, 5-4
transportable tablespace, 9-17
tree view buttons, 2-3
trigger
  general discussion, 8-2
TRUNCATE/INSERT, 8-11

U

union, 7-14
union all, 7-14
unique key
  constraint on levels, 4-5
Universal Identifiers (Universal IDs), 11-7
UPDATE, 8-11
update
  connection
    data source, 5-28
  flat file
    format, 5-47
  operations on specific columns, 8-16
  source definitions, 5-26
    fixed format file, 5-47
update (operation), 8-16
update (use for loading), 8-16
update (use for matching), 8-16
update definition
  view, 3-26
UPDATE/INSERT, 8-11
user-defined attribute set, 3-16
using
  Properties inspector, 9-3
  Workflow, 10-5
utilities button, 2-3, 2-7
utilities drawer, 2-7
  add a utility, 2-9
  configuration, 2-8
  remove a utility, 2-10

V

validating, 9-23
  editing invalid objects, 9-25
  validation codes, 9-25
  validation messages, 9-25
validation
  mappings, 8-33
  rationale and general discussion, 9-23
  rules for imported definitions, 11-20
versions
  bridges, 11-36
  Transfer Wizard, 11-36
view
  configuring, 9-11
  conventional, 4-4

create definition, 3-23
default column order, 3-11
logical constraints, 3-25
reordering the columns, 3-12
viewing
validation details, 9-25

W

Warehouse Builder Browser
administration, 12-45
browsing the favorites, 12-33
contents tab, 12-26
creating custom reports, 12-38
creating database links, 12-48
customizing the favorites pages, 12-36
detailed reports, 12-17
dropping database links, 12-51
editing database links, 12-50
implementation reports, 12-19
lineage and impact analysis diagrams, 12-21
lineage and impact analysis reports, 12-20
properties tab, 12-25
purging stale user information, 12-53
QA user role, 12-57
registering a custom report, 12-46, 12-52
registering a repository, 12-46
related tab, 12-27
reports tab, 12-28
roles, 12-55, 12-56
setting up preferences, 12-2
summary reports, 12-16
unregistering a repository, 12-52
viewing database links, 12-49
warehouse developer role, 12-57
warehouse user role, 12-57
Warehouse Builder MDL File Upgrade Utility, 11-1
Warehouse Builder Repository Assistant, 11-1
Warehouse Builder Runtime Assistant, 11-1
Warehouse Builder Transfer Wizard, 11-34
warehouse keys
column, 4-12
performance, 4-3
warehouse module, 3-2
connection information, 3-3
contents, 2-17
creating, 3-2
creating a business area, 4-39
database links, 3-4
displaying, 3-6
import definitions, 3-18, 4-26
navigation tree, 3-6
rename objects
materialized view, 3-23, 3-27
unique object names, 2-20
warehouse module editor, 3-6
warehouse object, 3-2
rename objects, 3-23
materialized view, 3-23
view, 3-27
welcome page, display, 2-20
WHERE, 7-28
wizard
display welcome page, 2-20
Flat File Sample Wizard, 5-30
Import Metadata Wizard
invoking from navigation tree, 5-50
invoking from New Module Wizard, 5-18
New Dimension Wizard, 3-7, 4-4
New Fact Table Wizard, 4-19
New Materialized View, 3-19
New Module Wizard, 3-2, 5-14, 5-16
New Time Dimension Wizard, 4-15
New Transform Wizard, 4-33
New View Wizard, 3-23
Oracle Workflow Deployment Wizard, 10-5
wizard buttons, 2-3
wizard navigation, 2-19
Workflow
defining the process, 10-10
deploying mappings, 10-5
process example, 10-12
Workflow Queue Listener
configuration parameters, 9-4
write lock, 2-17