Oracle® Warehouse Builder

Sources and Targets Guide

11*g* Release 2 (11.2)

E10582-06

July 2013

ORACLE®

Oracle Warehouse Builder Sources and Targets Guide, 11*g* Release 2 (11.2)

E10582-06

Primary Author: Reema Khosla

Contributors: Antonio Romero, Michelle Bird, Joyce Scapicchio, David Allan, Linda Bittarelli, Ron Gonzalez, Shawn Wu, Veenavani Revanuru, Ramesh Uppala, Alex Wu, Jean-Pierre Dijcks, Brian Maher, Gary Tripp, Xuelin Lu, Bojana Simova, Lyudmila Mogilevich, Ting Liao, Frank Yang, Justin Ho, Robert Paul, Adrian Scott, Robert Velisar, Alex Zhang, Winnie Wan, John Leigh, Thomas Lau, Geoff Watters, Padmaja Potineni, Cathy Shea, Roza Leyderman

Contents

11 Creating New Platforms

12 Using Code Templates to Load and Transfer Data

13 Importing and Publishing Web Services

Index

Preface

This preface includes the following topics:

Audience

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

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Conventions

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

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

The following table lists the conventions used in this guide:

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

Getting Help

Help is readily available throughout Oracle Warehouse Builder:

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

  Some toolbars include a Help icon, which displays the Contents page of the Help system.
- **Lists**: For items presented in lists, a description of the selected item displays beneath the list.
- **Shortcut menus**: Click the arrow icon on the right side of the title bar for a window. Then, choose **Help** from the shortcut menu for context-sensitive information.

Related Publications

In addition to the Sources and Targets Guide, the Oracle Warehouse Builder documentation set includes the following:

- *Oracle Warehouse Builder Installation and Administration Guide*
- *Oracle Warehouse Builder Concepts*
- *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*
- *Oracle Warehouse Builder API and Scripting Reference*
- *Oracle Warehouse Builder OMB*Plus Command Reference*

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

1

Connecting to Sources and Targets in Oracle Warehouse Builder

For different types of data sources and targets, Oracle Warehouse Builder provides different connection methods.

This chapter introduces the data sources and targets and specific connectivity technologies supported by Oracle Warehouse Builder. It includes the following topics:

1.1 Supported Sources and Targets

Table 1–1 lists the data storage systems and applications that Oracle Warehouse Builder 11*g* Release 2 (11.2) can access. The table lists the supported sources and targets for each type of application.

Table 1–1 Sources and Targets Supported in Oracle Warehouse Builder 11g Release 2 (11.2)

Application Type	Supported Sources	Supported Targets
Oracle Database	Oracle Database releases 9.2, 10.1, 10.2, 11.1,11.2	Oracle Database releases 9.2, 10.1, 10.2, 11.1, 11.2 **Note:** Connectivity among database versions over database links may depend upon the version of the database where an ETL mapping is deployed and executing. Refer to the documentation for your database version for details about any limitations on database links between database versions.
Non-Oracle Databases	▪ Any database accessible through "Oracle Database Heterogeneous Services", including but not limited to DB2, Informix, SQL Server, Sybase, and Teradata. ▪ Any data store accessible through ODBC, including but not limited to Excel and MS Access. ▪ Any data store accessible through code templates. See Chapter 12, "Using Code Templates to Load and Transfer Data".	▪ Any database accessible through "Oracle Database Heterogeneous Services", including but not limited to DB2, Informix, SQL Server, Sybase, and Teradata. ▪ Any data source accessible through ODBC, including but not limited to Excel and MS Access. ▪ Any data store accessible through code templates. See Chapter 12, "Using Code Templates to Load and Transfer Data".
Files	Delimited and fixed-format flat files. See "Importing Definitions from Flat Files Using Sampling" on page 3-12.	Delimited, fixed-format, and XML format flat files.
Business Applications	▪ SAP R/3: For details on versions supported, see the Certifications tab on My Oracle Support. See Chapter 7, "Extracting Data from SAP Applications". ▪ Oracle E-Business Suite, see "Importing Metadata from Oracle E-Business Suite Applications" on page 8-1 ▪ PeopleSoft release 8, 9, see "Importing Metadata from PeopleSoft Applications" on page 8-6 ▪ Siebel, see "Importing Metadata from Siebel Applications" on page 8-10	None
Process Flows and Schedules/Oracle Workflow	None	Oracle Workflow releases 2.6.2, 2.6.3, 2.6.4, 11*i*

Table 1–1 (Cont.) Sources and Targets Supported in Oracle Warehouse Builder 11g Release 2 (11.2)

Application Type	Supported Sources	Supported Targets
Process Flows and Schedules/Concurrent Manager	None	Any Oracle Database location, release 10*g* or later. To deploy a schedule in Concurrent Manager, Release 11*i* or 12*i* is required. However, for both releases, you must select 11*i* as the version when you create a location in Oracle Warehouse Builder.
Business Intelligence/Discoverer	None	Oracle BI Discoverer Release 10.1, Oracle Business Intelligence Suite Enterprise Edition
Oracle Designer	Oracle Designer 6*i*, 9*i*, 10*g*	None

1.2 Connecting to an Oracle Database

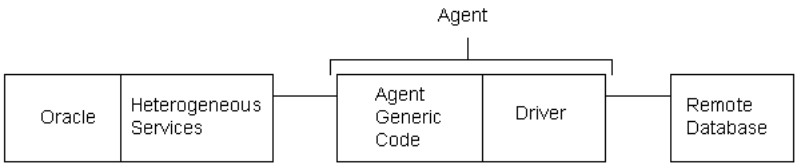
You can directly connect to an Oracle database and import metadata from the available database objects or deploy target objects to the database. To connect to a database on a remote host, you must provide the network credentials to connect to the host. See "Importing Metadata Definitions from Oracle Database" on page 2-9 for more information about connecting to an Oracle database.

1.3 Oracle Database Heterogeneous Services

Oracle Warehouse Builder communicates with non-Oracle systems using Oracle Database Heterogeneous Services and a complementary agent. Heterogeneous Services make a non-Oracle system appear as if it were a remote Oracle Database server. The agent can be an Oracle Database Gateway or the generic Open Database Connectivity (ODBC) agent included with Oracle Database.

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

Figure 1–1 Heterogeneous Services Architecture



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

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

1.3.1 Connecting Through Oracle Database Gateways

Oracle Database gateways provide transparent connections to non-Oracle databases. When using a gateway, you can access these non-Oracle databases just as you would

an Oracle database, including importing object metadata and accessing data as source and target. See Chapter 4, "Connecting to Non-Oracle Data Sources Through Gateways" for details of connecting to different databases through gateways.

1.3.2 Connecting Through JDBC

To connect to data sources that support Java Database Connectivity (JDBC), you must use the appropriate JDBC driver for that data source. The JDBC driver for a non-Oracle database or other data source may be installed with the product, or may require a separate download or purchase. For detailed information about connecting to various data sources using JDBC, see Chapter 6, "Connecting to Data Sources Through JDBC".

1.3.3 Choosing JDBC or Gateways Connectivity

To connect through JDBC, you must install the appropriate JDBC drivers for the database. Most of these drivers are shipped with the database. They are also developed by third-party vendors and are usually available as free downloads. The database objects, such as tables, that you import through a JDBC connection can be used only in Code Template mappings. These database objects cannot be used with other mappings, such as PL/SQL mappings, that you create in Oracle Warehouse Builder.

To connect through a gateway, you must install Oracle Gateway for the specific database. A gateway enables you to access data from non-Oracle databases in the same way that you access data from Oracle Database. The database objects that you import through a gateway connection can therefore be used in any of the mappings that you create in Oracle Warehouse Builder.

1.3.4 Connecting Through ODBC

Oracle Warehouse Builder can leverage support for ODBC provided by the Oracle Database to integrate with any data source that supports ODBC connectivity. ODBC provides a generic connectivity that is intended for low-end data integration solutions and the transfer of data is subject to the rules of specific ODBC or object linking and embedding (OLE) database drivers installed on the client computer. You need not install database-specific agents to connect to different data sources. Instead, you can use the generic connectivity agent included with Oracle Database. You must still create and customize an initialization file for your generic connectivity agent. For detailed information about connecting to data sources using ODBC connectivity, see "Connecting to an ODBC Data Source" on page 4-4.

1.4 Connecting To Enterprise Applications

Using Oracle Warehouse Builder, you can also connect to ERP and CRM applications, such as SAP, Oracle E-Business Suite, Peoplesoft, and Siebel.

You can also connect to Oracle's Customer Data Hub (CDH), Universal Customer Master (UCM), and Product Information Management (PIM).

The application adapters for Oracle applications provide additional metadata to simplify ETL design from such sources, but connecting to these sources depends upon underlying database connectivity. For example, metadata extraction from E-Business Suite, which is hosted on Oracle database, is done using database links, while metadata extraction from a Peoplesoft application hosted on a DB2 database depends on having the DB2 gateway installed on your target database. You can also connect to Peoplesoft using an alternative ODBC driver.

Connecting to an SAP system and extracting data is accomplished using native SAP R/3 ABAP code and data extraction techniques fully supported by SAP.

When using the application connectors, the details of the underlying connection technologies for different sources are generally hidden from the user, which simplifies managing the connectivity.

For more information about connecting to Oracle E-Business Suite, Peoplesoft, and Siebel, see Chapter 8, "Integrating with Oracle ERP Applications".

For information about extracting data from SAP systems, see Chapter 7, "Extracting Data from SAP Applications".

1.5 JDBC and Code Template-based Connectivity

In some cases, the data movement provided by Oracle-to-Oracle database links, gateways, and ODBC may not be sufficient. Oracle Warehouse Builder 11*g* Release 2 (11.2) also supports a framework that adds more flexible data movement options based on code templates.

When designing an ETL mapping, you can now choose between Oracle-based mappings and code template-based mappings for more flexible connectivity and data movement. Both mappings offer a rich set of data transformation operators.

When using code template-based mappings, you can select code templates to assign alternative data movement methods for individual mappings or specific parts of mappings. Oracle Warehouse Builder generates executable code based on the templates you select, and then deploys that code to a Control Center Agent, where it executes.

The JDBC connectivity for data movement depends on using code template-based mappings, and the data movement code actually executes in the control center agent. You can choose alternative templates for other data movement techniques such as bulk data extraction and loading that fit your use case better. You can also construct new templates that implement data movement methods not supported by Oracle Warehouse Builder out of the box.

For more information about using code templates, see Chapter 12, "Using Code Templates to Load and Transfer Data". Also see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for more details on code templates-based ETL mappings.

1.6 Choosing a Connection Technology

Choosing among the different connection technologies available depends upon your specific data source type and other specifics of your use case:

- Choose Oracle gateways for fully supported end-to-end connectivity with non-Oracle databases, and the most transparent access to those databases from within Oracle Warehouse Builder.
- Choose JDBC connectivity in instances where you need simplified setup of flexible connectivity with good performance across a wide range of non-Oracle data sources.
- Choose ODBC connectivity for sources where data volumes are relatively small and maximum performance is not a concern.
- Use application connectors when working with any supported ERP, CRM, or MDM application source or target.

2

Connecting to Data Sources and Importing Metadata

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

This chapter contains the following topics:

2.1 About Source Data and Metadata

Metadata is data that describes the contents of objects in a data set. For example, the metadata for a table includes such information as the name of the table and the names and data types of columns, relationships between this table and other objects, and so on. When working with objects in data sources not created within Oracle Warehouse Builder, you must first import metadata from those sources.

Oracle Warehouse Builder groups metadata for objects in different data sources in modules, which are visible in the Projects Navigator. The type of module you create depends on the source from which you are importing metadata. For example, to import metadata definitions from an Oracle database, create an Oracle module. To import metadata definitions from flat files, create a flat file module.

2.2 General Steps for Importing Metadata from Sources

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

To import metadata from a source:

1. Review the list of supported sources and targets in Table 1–1, " Sources and Targets Supported in Oracle Warehouse Builder 11g Release 2 (11.2)" to determine if the source from which you want to extract data is supported in Oracle Warehouse Builder.
2. Select an existing location or create a location as described in "Creating Locations" on page 2-3.

3. Create a module for the source metadata as described in "Creating Modules" on page 2-7.
4. Right-click the module and select **Import.**

   You can import metadata from database objects, flat files, COBOL copybook files, and Oracle Warehouse Builder metadata.
5. Follow the prompts in the Import Metadata Wizard.

   The wizard prompts you for information based on the type of source you selected. For more information, see "Using the Import Metadata Wizard" on page 2-8.

Subsequent Steps

For most data sources, after importing the metadata of the data objects, you can view the data stored in these objects.

To move the data using Oracle Warehouse Builder, you can design ETL logic that extracts the data from the source, transforms the data, and loads it into a target schema.

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

See Also:

- "Managing Metadata Dependencies" in the *Warehouse Builder Online Help*

To introduce the changed metadata into Oracle Warehouse Builder, right-click the desired module and select **Import.** As described in "Reimporting Definitions" on page 2-14, Oracle Warehouse Builder recognizes when you are reimporting metadata.

2.2.1 About Locations

Locations enable you to store the connection information to the various files, databases, and applications that Oracle Warehouse Builder accesses for extracting and loading data. Similarly, locations also store connection information to ETL management tools and Business Intelligence tools. For a detailed listing, see Table 1–1 on page 1-1.

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

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

Automatically Created Locations

During installation, Oracle Warehouse Builder creates an Oracle location named `OWB_REPOSITORY_LOCATION`. This location provides the connection details to the Oracle Warehouse Builder workspace.

Types of Locations

You can deploy to several different types of locations. These locations are available on the Locations Navigator of the UI. Each location type has a different use:

- **Databases**: Targets for either relational or dimensional systems, including objects such as tables and views, or dimensions and cubes. See, *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for more details about these objects.
- **Files**: Targets for storing data in delimited, fixed, or XML format.
- **Applications**: Targets for Oracle E-Business Suite, PeopleSoft, Siebel, and SAP systems.
- **Process Flows and Schedules**: Indicates an Oracle Workflow location.
- **Business Intelligence**: Targets for deploying metadata derived from databases or Oracle modules. This is then used by business intelligence tools.
- **URI Locations**: URI locations are used internally as part of the definition of other locations. You must not create any instances of it.
- **Agents**: Specifies the location of a control center agent or an application server.
- **Transportable Modules**: For defining transportable module sources and targets.

2.2.1.1 Creating Locations

You can create your own locations to use as sources or targets.

To create a location:

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

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

2. Right-click the type of location and select **New <Object> Location**.

   The Create <Object> Location dialog box is displayed.

   Or, right-click **Locations** and select **New**. This opens the New Gallery dialog box. Select a location type from the list of items and click **OK**.

   The Create <Object> Location dialog box is displayed.

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

   Provide the required connection details in the Details tab.

Using SQL*Net to Create Locations

When you create Oracle locations of type SQL*Net, you must set up a transparent network substrate (TNS) name entry for these locations. The TNS name must be accessible from the Oracle Warehouse Builder client home. Run the Net Configuration Assistant from the Oracle Warehouse Builder client home. See *Oracle Database Enterprise User Security Administrator's Guide* for information about using the Net Configuration Assistant.

After setting up the TNS name entry, restart the control center service so that it can pick up the changes. Furthermore, if the Warehouse Builder control center home is distinct from the Oracle Database home, then the same TNS name entry must be defined in the Oracle Database home as well. Run the Net Configuration Assistant from the Oracle Database home and specify the same TNS name there as well.

> **Note:** If the TNS names in the Control Center home and the Oracle Database home are different, then connection errors may occur during deployment and execution.

About Locations, Passwords, and Security

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

> **See Also:** *Oracle Warehouse Builder Installation and Administration Guide* for more information about "Managing Passwords".

2.2.1.2 Granting Privileges to a Target Location

Some deployments require the owner of the target location to have more powerful privileges than are granted when creating a new user:

- Upgrade action
- End User Layer (EUL) deployment

A privileged database user can grant the additional privileges.

For ETL, the owner of the target location must have sufficient privileges to read data from the source location. If the source location is a database table, for example, the owner of the target must have `SELECT` privileges on the table. Similarly, for locations pertaining to Transportable Modules sources and targets, users require additional privileges as described in *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

Upgrade Action

The `GRANT_UPGRADE_PRIVILEGES` PL/SQL script grants the target user the necessary roles and privileges for the upgrade action. Use this syntax:

```
@%OWB_HOME%/owb/rtp/sql/grant_upgrade_privileges username
```

`OWB_HOME` is the home directory for Oracle Warehouse Builder on the target system.

`username` is the owner of the target location.

For example, the following command grants privileges on a Windows system to the `SALES_TARGET` user.

```
@%OWB_HOME%\owb\rtp\sql\grant_upgrade_privileges sales_target
```

EUL Deployment

Oracle BI Discoverer locations require the EUL user to have the `CREATE DATABASE LINK` privilege.

2.2.1.3 Registering and Unregistering Locations

Locations must be created during the design process. All modules, including their source and target objects, must have locations associated with them before they can be deployed. You cannot view source data or deploy target objects unless there is a location defined for the associated module.

Registering a location in a control center establishes a link between the workspace and the locations of source data and deployed objects. You can register a location in multiple control centers. See *Oracle Warehouse Builder Concepts* for more information about control centers.

You can change the definition of a location before it is registered. However, after the location is registered, you can change only the password. To edit other details of a

location or one of its connectors, you must first unregister the location from all the control centers where it is registered. Unregistering a location deletes the deployment history of the location.

Locations are registered automatically by deployment. Alternatively, you can explicitly register a location in the Control Center.

To register a location:

1. Select the required project.
2. Select **Tools**, **Control Center Manager** to open the Control Center Manager.
3. Right-click the location and click **Register.**

   The Location dialog box is displayed.
4. Check the location details carefully.

   Click **Help** for additional information.
5. Click **Test Connection** to verify the connection details.
6. Click **OK**.

To unregister a location:

1. Select **Tools**, **Control Center Manager** to open the Control Center Manager.
2. Right-click the location and click **Unregister**.
3. Click **OK** to confirm the action.

2.2.1.4 Deleting Locations

To delete a location:

1. Right-click the location in Locations Navigator and select **Delete.**

   If the **Delete** option is not available here, this indicates that the location has been registered in a control center and is likely being used. See "Locations Registered in Multiple Control Centers" on page 2-5 for more information.
2. Verify that the location is not in use, unregister the location in the Control Center Manager, and then delete the location from Locations Navigator.

2.2.1.5 Locations Registered in Multiple Control Centers

After you register a location in a control center, the connection details of the location are locked. You can edit the connection details or delete the location only after unregistering it from the control center.

It is also likely that as part of your project requirements, a location is registered in multiple control centers. In such a scenario, you cannot edit the connection details of the location or delete the location unless you unregister the location from all the control centers where it is registered.

To see the details of all the control centers where a location is registered:

1. On the Locations Navigator, right-click the location and click **Open**.

   The Edit Location dialog box is displayed.
2. Click **Registration** to view the list of control centers where the location is registered.

2.2.2 About DB Connectors and Directories

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

To create a database directory, a user requires the `CREATE DIRECTORY` and `DROP DIRECTORY` privileges. When you create an Oracle Warehouse Builder user, these privileges are not automatically granted to the user. Therefore, the database administrator must explicitly grant these privileges to the Oracle Warehouse Builder user. For example:

```
GRANT CREATE ANY DIRECTORY TO OWB_USER;
GRANT DROP ANY DIRECTORY TO OWB_USER;
```

where `OWB_USER` is an Oracle Warehouse Builder user performing the deployment.

Or else, a user with these privileges must create the link connectors and grant you access to use them. You can then create the connectors manually and select the database link or directory from a list.

Note: To create a Database Link, a user requires the `CREATE DATABASE LINK` privilege. This privilege is granted automatically when an Oracle Warehouse Builder user is created.

See Also:

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

To manually create a database connector:

1. In the Locations Navigator, expand the **Locations** folder and then expand the subfolder for the target location.
2. Right-click **DB Connectors** and select **New DB Connector**.

   Follow the steps in the Create Connector Wizard.

To manually create a directory connector:

1. In the Locations Navigator, expand the **Locations** folder and then expand the subfolder for the target location.
2. Right-click **Directories** and select **New Directory**.

   The Create Connector Wizard opens.
3. Follow the steps and create a directory connector.

2.2.3 About Modules

Modules are grouping mechanisms in the Projects Navigator that correspond to locations in the Locations Navigator. A single location can correspond to one or more modules. However, a given module can correspond to only one metadata location and data location at a time.

The association of a module to a location enables you to perform certain actions more easily in Oracle Warehouse Builder. For example, group actions such as creating snapshots, copying, validating, generating, deploying, and so on, can be performed on all the objects in a module by choosing an action on the context menu when the module is selected.

2.2.3.1 Creating Modules

You can create a module from the Projects Navigator.

To create a module:

1. Expand the **Projects Navigator** until you find the node for the appropriate metadata type.

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

2. Right-click the desired node and select **New<Module Type>**.

   The Create Module Wizard opens.

   You can also right-click the desired node and select **New**. This opens the New Gallery dialog box. You can now select the item you want to create (module) and click **OK**.

   The Create Module Wizard opens.

3. On the Name and Description page, provide a name and an optional description for the module. For non-Oracle sources and applications, you must not select the access method. Select from **Native Database Connection** and **Oracle Gateway**.

4. Click **Next**.

   The Connection Information page is displayed.

5. Provide details about the location that is associated with this module.

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

6. Click **Next** to display the Summary page.

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

7. Click **Finish**.

While using Oracle Warehouse Builder, you must not associate a module with a new location. For example, assuming your production environment uses different locations than those used by your development environment, then you must change the module associations when moving code from the production to the development environment.

To change the location associated with a module, you must edit the configuration properties of the module. Configuration properties define the physical information pertaining to the metadata.

To change the location associated with a module:

1. In the Projects Navigator, right-click the module and select **Configure**.

   The Configuration Properties editor is displayed.

2. In the **Identification** node, select a new value for the Locations property. If the desired location is not in the list, then edit the module and add the location.

See *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for more information about "Configuring Data Objects".

2.2.3.2 Creating User Folders

After you create a module for a particular object such as an Oracle database, you can view all the associated objects under the database such as Mappings, Transformation, Tables, and so on.

You can also create your own folder, called User Folder, under a module. You can then define the required objects within the folder. For example, related tables and views that must be generated or deployed can be placed under a common folder. User Folders provide more flexibility in organizing objects within a module.

To create a User Folder:

1. Right-click a module, and click **New**.

   The New Gallery dialog box is displayed.

2. Select **User Folder** from the list of items and click **OK**.

   The Create User Folder dialog box is displayed.

3. Provide a name and description (optional), and click **OK**.

The newly created user folder is now available under the module. The folder is empty with no nodes under it. All the objects that can be added under a module can also be added under the newly created user folder. For example,

To add a table:

1. Right-click the folder, and click **New**.

   The New Gallery dialog box is displayed.

2. Select **Table** from the list of items and click **OK**.

   The Create Table dialog box is displayed.

3. Provide a name and description (optional) and click **OK**.

The newly created table is now available under the Tables node within the folder.

Within a user folder, you can also:

- Create other user folders.
- Copy valid objects from other user folders.
- Copy valid objects from other compatible module types.

2.3 Using the Import Metadata Wizard

Use the Import Metadata Wizard to import metadata definitions into modules.

The Import Metadata Wizard supports importing of tables, views, materialized views, dimensions, cubes, external tables, sequences, user-defined types, and PL/SQL transformations directly or through object lookups using synonyms.

Importing a table includes importing its columns, primary keys, unique keys, and foreign keys, which enable the import of secondary tables. When you import an external table, Oracle Warehouse Builder also imports the associated location and directory information for the associated flat file.

You can import metadata definitions either from the Oracle Database catalog or Designer/2000 (Oracle Designer).

2.4 Importing Metadata Definitions from Oracle Database

Create an Oracle module to store the imported metadata.

To create an Oracle module:

1. From the Projects Navigator, right-click **Oracle** under the Databases node and select **New Oracle Module**.

   The Create Module Wizard is displayed.

2. On the Name and Description page, provide a name and an optional description for the module. Click **Next** to open the Connection Information page.

3. Use the Connection Information page to select a database location for the module. To define a new location, click **Edit**. The Edit Oracle Database Location dialog box is displayed. See "Edit Oracle Database Location Dialog Box" on page 2-9 for the connection parameters to be entered in this dialog box.

   Click **OK** to open the Summary page.

4. Verify the specified details on the Summary page and click **Finish**.

   The newly created Oracle module is now available under the Oracle node in the Projects Navigator.

2.4.1 Edit Oracle Database Location Dialog Box

Use this dialog box to specify the location details of an Oracle location.

Name

A name for the location.

Description

An optional description.

Connection Type

Lists the connections available for access to a database location. You cannot change the type after creating the location. There are four connection types available: `HOST:PORT:SERVICE`, `Database Link`, `SQL*NET Connection`, and `OCI`.

Note: When the source and target locations are on the same host, be sure to identify them in the same way. Otherwise, the product treats them as separate locations and creates a database link between them. The database link may slow down the loading of data.

Common mistakes of this type include the following:

- Specifying `localhost` for one location and the actual computer name (such as `mycomputer-pc`) for another location.
- Specifying the domain inconsistently, for example, `mycomputer-pc` for one location and `mycomputer-pc.us.oracle.com` for another.

- **HOST:PORT:SERVICE**: Makes a connection using the Easy Connect Naming method, which requires no prior setup:
  - **User Name**: The database user credential that has permission to access the schema location.

    When connecting to a database that does not have user names, enter any text as a mock user name.
  - **Password**: The password associated with user name.

    When connecting to a database that does not have passwords, enter any text as a mock password.
  - **Host**: The name of the system where the database is installed.

    If Oracle Warehouse Builder is installed on the same system as the Oracle database, you can enter `localhost` instead of the computer name. If the database resides on a remote system, then provide the internet protocol (IP) address of the remote system.
  - **Port**: The SQL port number for the database.
  - **Service Name**: The service name of the database.
  - **Use Global Name**: The unique name of the database, which is composed of the database name and the domain in the form *database_name.database_domain*. For example, `orcl.us.example.com` identifies the `orcl` database in the `us.example.com` domain. Select this option when connecting to a database on a different network.
- **Database Link**: Makes a connection to another database using an existing database link. Select this method only when you do not have privileges that enable you to make a direct connection. You cannot deploy to a location that uses a database link. Not available for Oracle Business Intelligence or Discoverer locations.

  A database link is a schema object that contains information for connecting to a remote database. Database links are used in distributed database environments and enable a client to access two physical databases as one logical database.
  - **From Location**: An existing location where the database link is defined.
  - **Database Link**: The object name of the database link.
- **SQL*NET Connection**: Makes a connection using a net service name previously defined using a tool such as Oracle Net Configuration Assistant. The net service name provides a convenient alias for the connection information. This method of connecting is the best for Oracle RAC installations.
  - **User Name**: The database user credential that has permission to access the schema location.

    When connecting to a database that does not have user names, enter any text as a mock user name.
  - **Password**: The password associated with user name.

    When connecting to a database that does not have passwords, enter any text as a mock password.
  - **Net Service Name**: The name of the predefined connection.
  - **Use Global Name**: The unique name of the database, which is composed of the database name and the domain in the form *database_name.database_domain*.

For example, `orcl.us.example.com` identifies the `orcl` database in the `us.example.com` domain. Select this option when connecting to a database in a different network.

- **OCI**: Makes a connection using Oracle Call Interface (OCI) to interact with an Oracle database. This connection type uses the JDBC OCI driver that is specific to your version of Oracle database and your platform. Use this method when you need a high degree of compatibility with an earlier release of Oracle. The JDBC OCI drivers enable you to call the OCI directly from Java.
  - **User Name**: The database user credential that has permission to access the schema location.

    When connecting to a database that does not have user names, enter any text as a mock user name.
  - **Password**: The password associated with user name.

    When connecting to a database that does not have passwords, enter any text as a mock password.
  - **Net Service Name**: The name of the predefined OCI connection.
  - **Use Global Name**: The unique name of the database, which is composed of the database name and the domain in the form *database_name.database_domain*. For example, `orcl.us.example.com` identifies the `orcl` database in the `us.example.com` domain. Select this option when connecting to a database in a different network.

Schema

The schema where the source data is stored or the target objects are deployed. The schema must be registered. By default, it is the user name schema.

When connecting to a type of database that does not have schemas, do not enter any value in this field.

Version

The version number of Oracle Database. Not available for non-Oracle database locations.

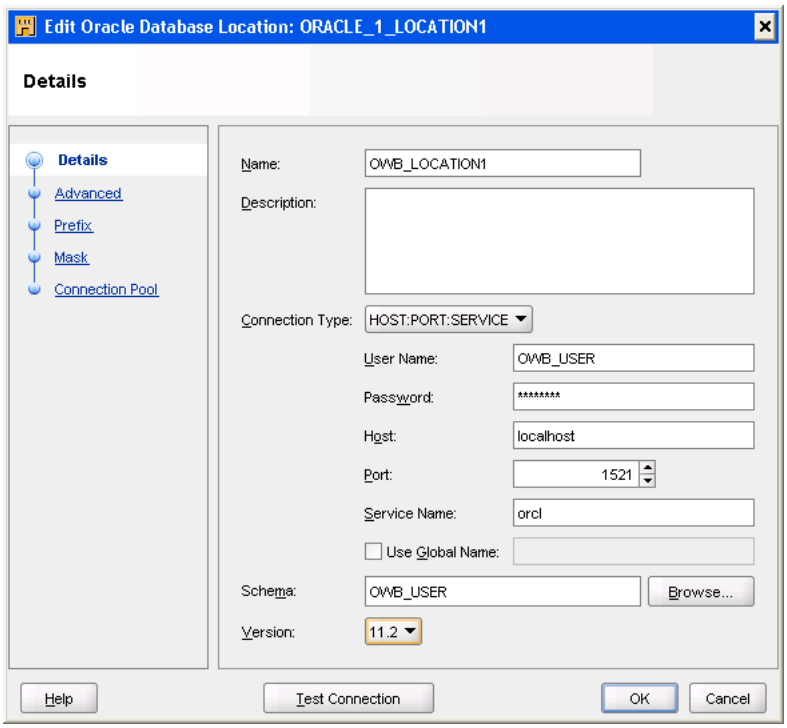
Test Connection

Attempts to make a connection using the values provided on this page.

Test Results

A message that reports whether a test connection succeeded or failed.

Figure 2–1 Edit Oracle Location Dialog Box



2.4.2 Importing Metadata Definitions

Use the Import Metadata Wizard to import metadata from an Oracle database into the module.

To import metadata definitions from an Oracle database:

1. Right-click the newly created Oracle module and select **Import**, then **Database Objects**.

   The Welcome page of the Import Metadata Wizard is displayed. This page lists the steps to import object metadata. Click **Next** to proceed with the import.

2. Complete the following:

   - "Filter Information Page"
   - "Object Selection Page"
   - "Summary and Import Page"
   - "Import Results Page"

2.4.3 Filter Information Page

Use the Filter Information page to limit the search using one of the following methods:

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

Search Based on the Object Name Use the **Only select objects that match the pattern** option to specify a search pattern. Oracle Warehouse Builder searches for objects whose names match the pattern specified. Use the percent sign (%) as a wildcard match for multiple characters and an underscore (_) as a wildcard match for a single character. For example, you can enter a warehouse project name followed by a percent sign (%) to import objects that begin with that project name.

Click **Next** to retrieve names that meet the filter conditions from the data dictionary. Oracle Warehouse Builder then displays the Object Selection page.

2.4.4 Object Selection Page

Select items to import from the Available list and click the right arrow to move them to the Selected list. If you are reimporting definitions, previously imported objects appear in bold.

To search for specific items by name, click the **Find Objects** icon that looks like a flashlight.

To move all items to the Selected Objects list, click **Move All**.

Importing Dependent Objects The Import Metadata Wizard enables you to import the dependent objects of the object being imported.

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

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

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

Importing Dimensions When you import a dimension that uses a relational implementation, the implementation table that stores the dimension data is not imported. You must explicitly import this table by moving the table from the Available list to the Selected list on the Object Selection page. Also, after importing the table, you must bind the dimension to its implementation table. For more information about how to perform binding, refer to *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

2.4.5 Summary and Import Page

This page summarizes your selections in a spreadsheet that lists the name, type of object, and whether the object are reimported or created. Verify the contents of this page and add descriptions, if required, for each of the objects.

You can specify additional properties for objects that have been imported into the module. Click **Advanced Import Options** to specify these properties. The Advanced Import Options dialog box is displayed. For more information about the contents of this dialog box, see "Advanced Import Options" on page 2-15. For more information about reimporting definitions, see "Reimporting Definitions" on page 2-14.

Click **Finish** to import the selected objects. The Importing Progress dialog box shows the progress of the import activity. After the import completes, the Import Results page is displayed.

2.4.6 Import Results Page

This page summarizes the import result, and lists the objects and details about whether the object was created or synchronized.

Click **OK** to accept the results. To save a metadata loader (MDL) file associated with this import, click **Save.** Oracle Warehouse Builder stores the definitions in the database module from which you performed the import.

2.5 Reimporting Definitions

Reimporting your source database definitions enables you to import changes made to your source metadata after your previous import. You do not have to remove the original definitions from the workspace. Oracle Warehouse Builder provides you with options that also enable you to preserve any changes you may have made to the definitions since the previous import activity. This includes any new objects, foreign keys, relationships, and descriptions you may have created in Oracle Warehouse Builder.

To reimport definitions:

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

   The Welcome page for the Import Metadata Wizard is displayed.

2. Click **Next.**

   The Filter Information page is displayed.

3. Complete the "Filter Information Page" and "Object Selection Page", selecting the same settings used in the original import to ensure that the same objects are reimported.

4. The Summary and Import page is displayed.

   If the source contains new objects related to the object you are reimporting, the wizard requires that you import the new objects at the same time.

5. Click **Advanced Import Options** and make selections. (Optional)

6. Click **Finish.**

   Oracle Warehouse Builder reconciles and creates objects. When this is complete, the Import Results dialog box displays.

   The report lists the actions performed by Oracle Warehouse Builder for each object. For objects that have been reimported, the Actions column displays Synchronized. You can also expand an object to see which elements have changed. For example, you can see which columns of a table have changed since the last import.

   Click **Save** to save the report. You should use a naming convention that is specific to the reimport.

7. Click **OK** to proceed or click **Undo** to undo all changes to your workspace.

2.5.1 Advanced Import Options

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

By default, all options on this dialog box are selected. Deselect these options to have these objects replaced and not preserved.

For example, after importing tables or views for the first time, you manually add descriptions to the table or view definitions. To ensure that these descriptions are not overwritten while reimporting the table or view definitions, you must select the Preserve Existing Definitions option.

The contents of this dialog box depend on the type of objects being imported. For more information about the advanced import options for each type of objects, see the following sections:

- "Advanced Import Options for Views and External Tables"
- "Advanced Import Options for Tables"
- "Advanced Import Options for Object Types"

2.5.1.1 Advanced Import Options for Views and External Tables

Select these options to reconcile views or external tables:

- **Import descriptions:** The descriptions of the view or external table are imported. Existing descriptions are not preserved.
- **Preserve workspace added columns:** The columns you added to the object in the workspace are preserved.

> **See Also:** *Oracle Warehouse Builder Concepts* for more information about "Workspaces".

2.5.1.2 Advanced Import Options for Tables

Select these options to reconcile tables:

- **Preserve workspace added columns:** To retain any columns added to the table in the workspace.
- **Preserve workspace added constraints:** To preserve the constraints you added to the table in Oracle Warehouse Builder.
- **Import indexes:** To specify additional details about how indexes should be imported. Importing indexes consists of the following options:
  - **Preserve workspace added indexes:** To retain any indexes added to the workspace table.
  - **Import physical properties of indexes:** To indicate how indexes should be imported. Select the **Preserve workspace added physical properties of indexes** option below this option to specify that any physical properties added to the indexes should be preserved.
  - **Import index partitioning:** To indicate how index partitions should be imported. Select the **Preserve repository added index partitioning** option to specify that any index partitions added to the workspace table must be preserved.

- **Import partitioning:** To specify additional details about how partitions should be imported. Importing partitions contains the following options:
  - **Preserve workspace added partitioning:** To retain all partitions added to the workspace table.
  - **Import physical properties of partitioning:** Use this option to indicate how the physical properties of partitions should be imported. Select **Preserve workspace added physical properties of partitioning** to indicate that all physical properties of the partitions in the workspace table should be retained.
- **Import physical properties:** To indicate how the physical properties of the table should be imported. Select the **Preserve workspace added physical properties** option to specify that all physical properties added to the workspace table must be preserved.
- **Import descriptions:** To import the descriptions of the table.

2.5.1.3 Advanced Import Options for Object Types

Select these options to reconcile object types:

- **Import descriptions:** To import the descriptions of the object type.
- **Preserve workspace added attributes:** To retain the attributes added to the object type in the workspace.

2.5.2 Updating Source Module Definitions

The Edit Module dialog box enables you to edit the name, metadata location, and the data locations for a source module.

To update the database definitions of an Oracle module:

1. Right-click the module, and select **Open**.

   The Edit Module dialog box displays.

2. To edit the metadata location, click the **Metadata Location** tab and specify the following:
   - **Source Type**: Identifies the location of the metadata. It can be either Oracle Data Dictionary or Oracle Designer Repository. Select Oracle Data Dictionary if the metadata is stored in the default workspace of the Oracle Database. Select Oracle Designer Repository if the metadata is stored in an Oracle Designer repository.
   - **Location**: Identifies the metadata location for the module. You can select a location from the list.
3. To edit the data location, click the **Data Locations** tab. By default, the location name specified while creating the module is selected. To change the location, you can either select from other existing locations or create a new location. To create a new location, click **New**. The Edit Oracle Database Location dialog box displays. Specify the details of the data location in the dialog box.

This chapter provides a generic description of importing metadata, which is identical for Oracle databases and other data source types. Follow these procedures when working with Oracle databases.

For information about importing metadata definitions from non-Oracle databases and other applications, see:

- Chapter 3, "Using Flat Files as Sources or Targets" for importing metadata definitions from flat files and COBOL copybooks.
- Chapter 4, "Connecting to Non-Oracle Data Sources Through Gateways" for connecting to and importing metadata from non-Oracle databases such as DB2 and SQL Server using Oracle Gateways.
- Chapter 6, "Connecting to Data Sources Through JDBC" for connecting to non-Oracle databases such as DB2 and SQL Server using JDBC connectivity.
- Chapter 8, "Integrating with Oracle ERP Applications" for details of importing metadata from Oracle applications including Oracle E-Business Suite, PeopleSoft, and Siebel applications.
- Chapter 7, "Extracting Data from SAP Applications" for details of connecting to a SAP application and extracting data.
- Chapter 9, "Importing Oracle Warehouse Builder Data into Business Intelligence Applications" for information about deriving business intelligence objects and deploying these objects into Oracle Business Intelligence Suite Enterprise Edition and Oracle BI Discoverer.
- Chapter 10, "Importing Design Definitions from Oracle Designer" for information about importing design definitions from Oracle Designer applications.

3

Using Flat Files as Sources or Targets

You can use flat files as either source files or target files within mappings in Oracle Warehouse Builder.

This chapter describes the use of flat files as sources and targets in Oracle Warehouse Builder. It contains the following topics:

3.1 About Flat Files

When using flat files as sources:

- You can read from character data set files or binary flat files.
- You can read from delimited files, fixed length files, or XML files.
- You can use flat file operators in SQL*Loader mappings to read from flat files directly, or you can add an external table and access the flat file data in a mapping using SQL and PL/SQL.
- You can also add flat file operators in code template based mappings and leverage code templates that are specifically constructed for files or the generic SQL code templates which leverages a built-in JDBC driver for files.

When using flat files as targets:

- You can use only character data set files. Binary flat files are not supported as targets.
- You can write to delimited files and fixed length files.
- You can use flat file operators to write data to flat files.

Note: The same flat file can act as a source and a target file.

3.1.1 About Metadata for Flat Files

File metadata describes the structure of data records in the file, including column names and data types. Before you use a flat file as source, it is desirable to define the metadata of that flat file.

You can import flat file metadata from several sources:

- For character files with displayable data, you can use the Flat File Sample Wizard to view and analyze the flat file contents and deduce metadata from them.
- For COBOL copybooks, you can import metadata definitions directly from the copybook file.
- For binary files, files that are too complex for the Flat File Sample Wizard, and for target files for which no sample is available yet, you can explicitly define metadata for your flat file using the Create Flat File Wizard.

3.1.2 About Flat File Modules and Locations

You can create flat file modules in your project that store metadata for source and target files. Each flat file module must be associated with a metadata and data location. When you specify a location for a module, both metadata location and data location point to the same location. You can specify different locations for the metadata and the data by editing the module.

3.1.3 Working with Flat Files as Sources or Targets

The usual cycle, when working with a flat file as a source or target, is to:

1. Ensure that the location where the flat file stored is accessible from the host on which the mappings accessing the flat file is deployed.
2. Create a flat file module and associate it with the flat file module location. See "Creating Flat File Modules" on page 3-4.
3. Define the flat file and specify its structure, based on whether its a character file, binary file, or a COBOL copybook. See "Using the Flat File Sample Wizard" on page 3-13 for details of Flat File wizard. See "Using the Create Flat File Wizard" on page 3-6 for details of creating a flat file for a binary file. See "Importing Metadata Definitions from COBOL Copybooks" on page 3-20 for details of importing metadata from COBOL copybooks.
4. Select the type of mapping to use to extract data from the file. Consider whether you want to use flat file operators or external tables. See "Choosing Between External Table and Flat File Operators" on page 3-3. If using a PL/SQL mapping, create external tables to represent the file contents as database tables.
5. Design your PL/SQL or SQL*Loader ETL mapping using the flat files as a source or target. See *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for details of creating mappings.

3.1.4 Flat Files as Sources

To use a flat file as a source, first define the metadata structure of the flat file and then import metadata into it.

3.1.4.1 Defining Character Data Files

You can import metadata from various types of files including character data set files and COBOL copybooks.

To define flat file metadata, complete the following steps:

1. Create a flat file module.

   Create a module for each unique directory or path in your file system from which you want to import file metadata. See "Creating Flat File Modules" on page 3-4.

2. Define the structure of the file.

   The Flat File Sample Wizard enables you to view a sample of the flat file and to define record organization and file properties. The wizard enables you to sample and define common flat file formats such as string and ASCII. See "Using the Flat File Sample Wizard" on page 3-13.

   For files with complex record structures, the Flat File Sample Wizard may not be suitable for sampling the data. In such cases, you must create a flat file and define its structure accordingly. See "Using the Create Flat File Wizard" on page 3-6 for creating and defining the structure of a flat file.

   For COBOL copybooks, use the COBOL import dialog box to import metadata from copybooks. You can also set the import options depending on the copybook. See "Importing Metadata Definitions from COBOL Copybooks" on page 3-20 for more details.

3.1.4.2 About External Tables

An external table is a read-only table that is associated with a single record type in a flat file. External tables represent data from a non-relational source in a relational table format. When you use an external table in a mapping, column properties are based on the SQL properties defined when importing the flat file. For more information about SQL properties for flat files, see "SQL Properties" on page 3-11.

When you use an external table as a source table in a mapping, you can use it as a regular source table. Oracle Warehouse Builder generates PL/SQL code to select rows from the external table. You can also get parallel access to the file through the table. You also have access to additional relational function operators.

Note: You can use external tables only for source tables.

You can either import an existing external table from another database as described in"Importing an External Table" on page 3-37 or define a new external table as described in "Creating a New External Table Definition" on page 3-36.

3.1.4.3 Choosing Between External Table and Flat File Operators

You can introduce source data from a flat file into a mapping either through an external table or a flat file operator. In general, external tables are the preferred method of loading large volumes of data from flat files.

Note the following details when comparing external tables and flat files:

- External table operators and PL/SQL mappings provide for maximum performance, including exploitation of database parallelism during load. The full range of transformation operators is available, because the mappings are PL/SQL mappings. Over time, ETL mappings built with external tables takes advantage of further performance improvements at the database level.
- Flat file operators and SQL*Loader mappings are a fully supported method of loading flat files. Oracle Warehouse Builder generates native SQL*Loader code for a SQL*Loader mapping. A more limited range of operators is supported in

SQL*Loader mappings. You must stage data in an intermediate table and then use a PL/SQL mapping to transform it further before loading into a final target.

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

For more information about different types of mappings, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

3.1.5 Flat Files as Targets

When you use a flat file as a target, it is desirable to define the metadata but is not necessary. For example, while using unbound flat file operators in mappings, the metadata for the flat file might not be defined before using it in the mapping. However, in practice, it might be more convenient to define the metadata definitions of a target file rather than to use an existing file as a target.

Creating a New Flat File as a Target

Create a flat file and define its structure before loading data into it.

To design a new flat file, complete the following steps:

1. Create a module for the flat file. See "Creating Flat File Modules" on page 3-4.
2. Use the Create Flat File Wizard to design the metadata structure of the flat file. See "Using the Create Flat File Wizard" on page 3-6.
3. Use the newly created flat file as a target.

Note: You can also create a target flat file in a mapping. Add an unbound flat file operator to a mapping, then map from the source table or operator to the flat file. Finally, do a Create and Bind for the flat file operator. For more information about mappings, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide.*

3.1.6 Creating Flat File Modules

Flat files are stored within modules that enable you to group multiple flat files.

To create a flat file module:

1. Right-click the **Files** node in the Projects Navigator and select **New Flat File Module**.

   Oracle Warehouse Builder displays the Welcome page for the Create Module Wizard.

2. Define the module in the following steps:

   1. "Describing the Flat File Module"
   2. "Defining Locations for Flat File Modules"

3. The Finish page summarizes the information you provided on each of the wizard pages. When you click **Finish,** the wizard creates the flat file module and inserts it under Files in the Projects Navigator.

   After creating a flat file module, you can either define a new flat file, as described in "Using the Create Flat File Wizard" on page 3-6, import existing flat files into this module as described in "Using the Flat File Sample Wizard" on page 3-13, or import a COBOL file as described in "Importing a Copybook" on page 3-28.

3.1.6.1 Describing the Flat File Module

Enter a name and an optional description for the flat file module on the Name and Description page.

3.1.6.2 Defining Locations for Flat File Modules

Locations for flat file modules identify the paths in the file system from which you sample existing files or to which you create new files. You can define a new location or select an existing location on the Connection Information page.

Flat file modules have metadata and data locations. When you specify a location for a module, both metadata location and data location point to the same location. You can specify different locations for the metadata and the data by editing the module.

To import metadata from flat files located in different directories or paths in the system then for ease of use, create separate Oracle Warehouse Builder modules for each path. For example, suppose the files are located at the following paths, `c:\folder1` and `c:\folder1\subfolder`. You can create two file modules *C_FOLDER1* and *C_FOLDER1_SUBFOLDER* and associate them with the corresponding paths. However, associating a module with a path does not restrict you from importing metadata definitions of files residing in a different path. You can define a path as the default, and later import files from a different path.

A data location identifies only a folder in the file system and does not include subfolders.

3.1.6.3 Connection Information Page

The Connection page is displayed with a default location name that is based on the module name that you entered in the Name and Description page. If you do not want to create a location, then select from the list of existing locations.

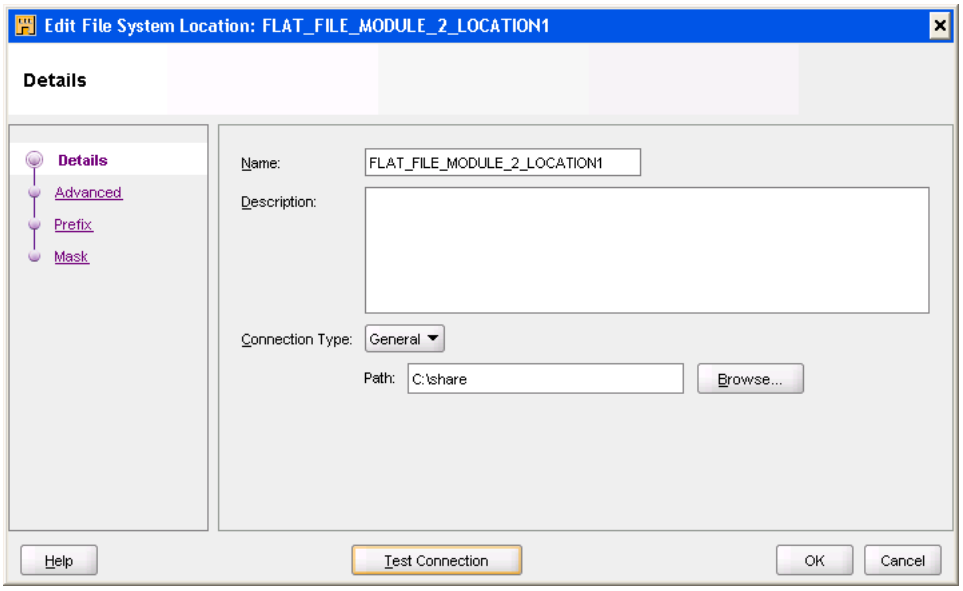
On the Connection Information page, click **Edit** to open the "Edit File System Location Dialog Box" and specify the location details. This location becomes the metadata and data location.

3.1.6.4 Edit File System Location Dialog Box

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

Figure 3–1 shows the Edit Location dialog box.

Figure 3–1 Edit File System Location Dialog Box



The figure displays the Details tab of the Edit File System Dialog Box. The fields from top to bottom are Name, Description, Connection Type, and Path. The Browse button next to the Path field enables you to browse and select a path. On the bottom of the screen, the following buttons are available from left to right, Help, Test Connection, OK, and Cancel. The following tabs are available on the top left corner and are listed from top to bottom: Details, Advanced, Prefix, and Mask.

****

3.2 Using the Create Flat File Wizard

Use the Create Flat File Wizard to design the structure of a new flat file in Oracle Warehouse Builder. This could be the case when you must define binary files and using the Flat File Sample wizard is not a viable solution. You can also use this wizard to create a flat file for use as a target in a mapping.

To use the Create Flat File wizard, right-click the flat file module and select **New Flat File**.

The Create Flat File Wizard guides you in completing the following steps:

- "Describing a Flat File"
- "Defining File Properties for a Flat File"
- "Defining the Record Type for a Flat File"
- "Defining Field Properties for a Flat File"

3.2.1 Describing a Flat File

Use the Name and Description page to provide a name for the flat file and to specify general properties associated with it.

- **Name**: This name uniquely identifies the file within the module. Enter a name that does not include a space or any punctuation. You can include an underscore. You can use uppercase and lowercase letters. Do not start the name with a digit. Do not start a name with the reserved prefix `OWB$`.

- **Default Physical File Name**: A physical file name may be specified. This name can be altered at any time using configuration properties. If you are creating a new file, you can leave this name blank. If you are defining an existing binary file, enter the name of the file. Do not include the file path.
- **Character set**: Select a character set or accept the default character set defined for the system on which Oracle Warehouse Builder resides. For complete information about NLS character sets, see *Oracle Database Globalization Support Guide*.
- **Description**: You can enter an optional description for the file.

3.2.2 Defining File Properties for a Flat File

Use the File Properties page to specify Record Organization, Logical Record Definition, Number of Rows to Skip, and the Field Format for the flat file as shown in Figure 3–2.

Use the file properties page to specify Record Organization, Logical Record Definition, Number of Rows to Skip, and the Field Format for the flat file.

3.2.2.1 Record Organization

Indicate how to organize the records in the file. Select between the two options to indicate how the length of each record in the file is determined:

- **Records delimited by:** Select this option to designate the end of each record by a delimiter. Then specify that record delimiter. You can accept the default record delimiter, new line (\n), or you can enter a new value. You can provide multiple characters and hexadecimal characters as a record delimiter. The hexadecimal character format is `x'<hexadecimal string>'` or `X'<hexadecimal string>'`. Using hexadecimal characters is useful if the delimiter character is not a new line character (\n) or carriage return<CR>. For example, to specify the pipe symbol (|) as the delimiter, use its hexadecimal value `x'7C'`.
- **Record length (in characters):** Select this option to create a file with all records having the same length. Then specify the number of characters in each record. For files with multibyte characters, count a multibyte character as one character.

3.2.2.2 Logical Record Definition

By default, the wizard creates a file in which each physical record corresponds to one logical record. You can override the default to create a file composed of logical records that correspond to multiple physical records.

- **Number of physical records for each logical record**: The data file contains a fixed number of physical records for each logical record.

```
PHYSICAL_RECORD1
PHYSICAL_RECORD2
PHYSICAL_RECORD3
PHYSICAL_RECORD4
```

  In the preceding example, if the number of physical records for each logical record is 2, then `PHYSICAL_RECORD1` and `PHYSICAL_RECORD2` form one logical record and `PHYSICAL_RECORD3` and `PHYSICAL_RECORD4` form the second logical record.

- **End character of the current physical record**: The data file contains a variable number of physical records with a continuation character at the end that signifies that the record is continued in the next physical record.

  In the following example, the continuation character is a percentage sign (%) after the record.

```
PHYSICAL_RECORD1%
PHYSICAL_RECORD2       end log rec 1
PHYSICAL_RECORD3%
PHYSICAL_RECORD4       end log rec 2
```

- **Start character of the next physical record**: The data file contains a variable number of physical records with a continuation character at the beginning of each physical record that signifies that the record continues from the previous physical record.

  The following example shows two logical records with a continuation character at beginning of the record.

  ```
  PHYSICAL_RECORD1
  %PHYSICAL_RECORD2      end log rec 1
  PHYSICAL_RECORD3
  %PHYSICAL_RECORD4      end log rec 2
  ```

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

  ```
  PHYSICAL_RECORD1
  %PHYSICAL_RECORD2
  %PHYSICAL_RECORD25
  %PHYSICAL_RECORD26   end log record 1
  PHYSICAL_RECORD3
  %PHYSICAL_RECORD4
  %PHYSICAL_RECORD45
  %PHYSICAL_RECORD46 end log record 2
  ```

3.2.2.3 Number of Rows to Skip

When defining an existing file, indicate the number of records to skip at execution time in **Skip rows.** This is useful for skipping over header and field name records.

When creating a new target file, you can leave this value blank.

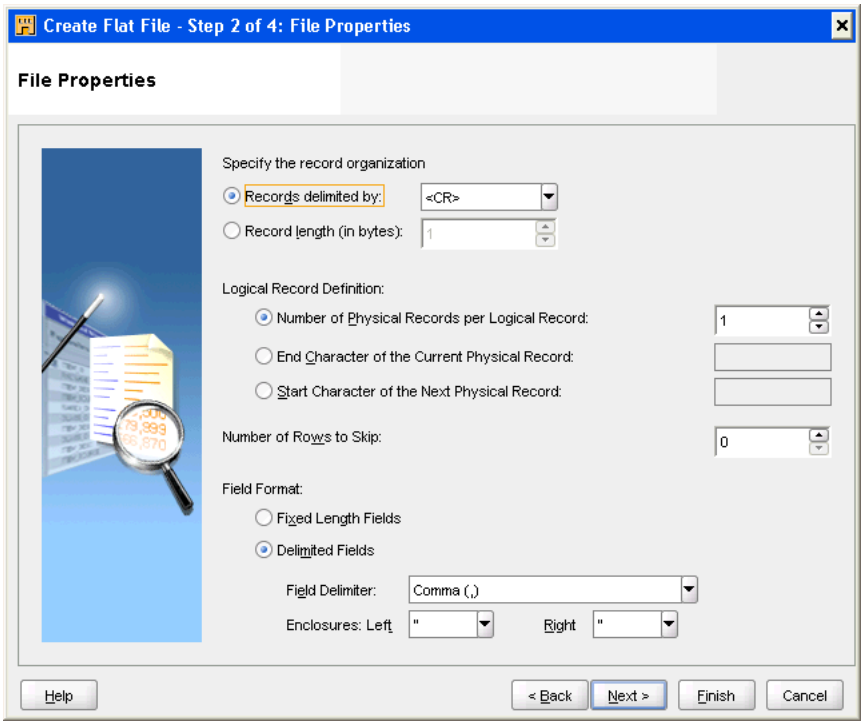
3.2.2.4 Field Format

Select between **Fixed Length** and **Delimited** formats for the file.

To create a delimited file, specify the following properties:

- **Field delimiter**: Field delimiters designate where one field ends and another begins. You can enter a field delimiter or select one from the list. The list displays common field delimiters. However, you may enter any character as a delimiter except the ones used for enclosures. The default is the comma (,). You can provide multiple characters and hexadecimal characters as a delimiter. The hexadecimal character format is `x'<hexadecimal string>'` or `X'<hexadecimal string>'`. For example, to specify the pipe symbol (`|`) as the delimiter, use its hexadecimal value `x'7C'`.
- **Enclosures (Left and Right)**: Some delimited files contain enclosures that denote text strings within a field. If the file contains enclosures, enter an enclosure character in the text box or select one from the list. The list displays common enclosures. However, you may enter any character. The default for both the left and right enclosure is the double quotation mark ("). You can specify multiple characters and hexadecimal characters as field enclosures.

Figure 3–2 File Properties Page



The figure shows the File Properties page of the Create Flat File Wizard. The following fields are listed from top to bottom: Records delimited by, Record length (in bytes), Number of Physical Records per Logical Record, End Character of the Current Physical Record, Start Character of the Next Physical Record, Number of Rows to Skip, Fixed Length Fields, Delimited Fields, Field Delimiter, Enclosures Left, Right. At the bottom of the page, the following buttons are available from left to right: Help, Back, Next, Finish, Cancel.

****

3.2.3 Defining the Record Type for a Flat File

Indicate whether the file you create is to contain a single record type or multiple record types. The default is set to **Single Record.**

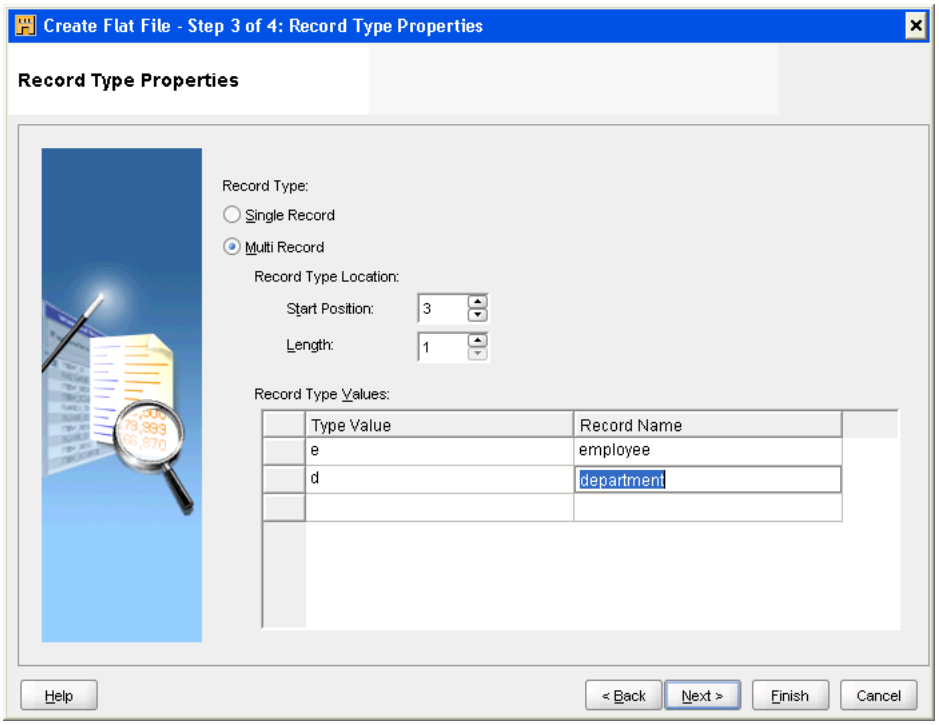
If the file contains multiple record types, select **Multi Record**. For each record type you want to create, specify values under **Record Type Location** and then its type value and record name.

Valid entries for Record Type Location depend on the field format you selected on the File Properties page, fixed length or delimited fields.

For example, if you specify the fields as delimited, then indicate the field position as shown in Figure 3–3.

For fixed-length files, the page displays two fields, Start Position and Length, under Record Type Location. Indicate the start position and the length of the field.

Figure 3–3 Record Type Properties Page



The figure shows the Record Type Properties page of the Create Flat File Wizard. The following fields are listed from top to bottom: Single Record, Multi Record, Field Position. Below this is the Record Type Values spreadtable, which consists of two fields: Type Value and Record Name. At the bottom of the page, the following buttons are available from left to right: Help, Back, Next, Finish, Cancel.

****

3.2.4 Defining Field Properties for a Flat File

Use the Field Properties page to define properties for each field.

Since you can use a flat file in a mapping either directly as a source or a target, or indirectly through an external table, the Field Properties page shows both "SQL*Loader Properties" and "SQL Properties". Use the scroll bar to scroll to the right and view all the properties.

3.2.4.1 SQL*Loader Properties

The first set of properties the wizard displays are for the SQL*Loader utility. When you use the flat file directly as a source in a mapping, SQL*Loader and the properties you set here are used. SQL*Loader properties include details of how the following are mapped to a relational table: "Type", "Length", "Precision", "Scale", "Mask", "NULLIF", and "DEFAULTIF". See *Oracle Database Concepts* for more details.

Type

Describes the data type of the field for SQL*Loader. You can use the wizard to import many data types such as `CHAR`, `DATE`, `DECIMAL EXTERNAL`, `FLOAT EXTERNAL`, `INTEGER EXTERNAL`, `ZONED`, and `ZONED EXTERNAL`. For complete information about SQL*Loader field and data types, see *Oracle Database Utilities*.

Length

For delimited files, specifies the maximum field length to be used by SQL* Loader.

Precision

Specifies the number of digits for certain data types such as Zoned and Float. See *Oracle Database Utilities* for more details.

Scale

Specifies the number of decimal digits for certain data types such as Zoned and Float. See *Oracle Database Utilities* for more details.

Mask

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 SQL*Loader default, you can enter the true format as a mask.

NULLIF

You can override the default action of SQL*Loader by placing a NULLIF condition on a field. For example, when a character field contains all blanks, you can direct SQL*Loader to mark the field as null rather than storing the blanks. Valid syntax for this field includes *=BLANKS, ='quoted string', =X'ff'* to indicate hexadecimal values, and *!=* for 'not equal to' logic.

DEFAULTIF

You can override the default action of SQL*Loader by placing a DEFAULTIF condition on a field. For example, when a numeric or DATE field contains all blanks, SQL*Loader rejects the entire record. To override this action, type *=BLANKS* in the DEFAULTIF property. When SQL*Loader evaluates this condition, it sets the numeric field to zeros and loads the record. Valid syntax for this field includes *=BLANKS, ='quoted string', =X'ff'* to indicate hexadecimal values, and *!=* for 'not equal to' logic.

3.2.4.2 SQL Properties

These properties specify how the fields in a flat file translate to the columns in a relational table. They are used to define the characteristics of an external table. They are also used to automatically generate fields for mapping to relational operators. Similarly, if the flat file is used as a target, then these properties are used to generate the PL/SQL code.

The SQL properties you set here have the following implications for mapping design, validation, and generation:

- **External table**: If you create an external table based on a single flat file record type, the columns properties are based on the SQL properties you defined for the flat file. For more information about external tables, see "Using External Tables" on page 3-35.
- **Populating an Empty Mapping Object**: In a mapping, if you populate an empty relational object with the metadata, then the object inherits the SQL properties you defined for the flat file source.
- **Flat file target**: If you use the flat file as a target in a mapping, the target does not inherit the SQL properties. Instead, all fields inherit the default SQL*Loader data type.

SQL Type

Oracle Warehouse Builder supports many SQL data types such as CHAR, DATE, FLOAT, and BLOB.

The wizard assigns a default value for the SQL type based on SQL*Loader properties you set. If you accept the default SQL type, the type is updated if you later change SQL*Loader properties. However, if you override the SQL type by selecting a new SQL type from the list, it then becomes independent of the flat file SQL*Loader data type.

SQL Length

This property defines the length for the SQL column, if appropriate.

SQL Precision

This property defines the precision for the SQL column, if appropriate. For example, when defining NUMBER and FLOAT fields, the precision may be set.

SQL Scale

This property defines the scale for the SQL column, if appropriate. For example, when defining NUMBER and FLOAT fields, the scale may be set.

Select **Automatically update start and end positions for all fields** if you want all the field positions to be automatically recalculated based on changes made to any field.

Once you define the metadata of the new flat file, you can use it as a source or target file, or create an external table using a record from the file.

3.3 Importing Definitions from Flat Files Using Sampling

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

To use the Flat File Sampling Wizard:

1. Establish connectivity to the files you want to import.

   Because the Flat File Sample Wizard runs on the host running the Design Center client, ensure that the files to be sampled are accessible from that host. You can either mount a remote file system across your network using a method such as network file system (NFS) or Windows file sharing, or copy the files, or a representative section of the files, to a file system on or accessible from the Design Center client host.

2. Create a flat file module that contains the imported flat file definitions. See "Creating Flat File Modules" on page 3-4 for details.

   Either create a module for each folder in the file system from which you want to import files or use the same module to import file definitions from multiple folders.

   When you create a flat file module, the location corresponding to this module is a path in the file system which acts as the metadata and data location. Use the Connection Information Page of the Create Module Wizard to specify this path.

   A flat file location does not include subfolders of the specified folder.

3. Right-click the flat file module and select **Import**, **Flat File**. Alternatively, select the flat file module, and then from the main menu, select **File**, **Import**, **Flat File**.

The File Import dialog box is displayed.

4. Click **Add Sample File**, and select the files to import.

   You can add single or multiple files into a module. All the files you add are listed under **Sample File**. If you specify a file in the **Same As** field, then the definition of the sampled file is based on the definition of the file specified in the **Same As** field.

5. Click **Import**. The Flat File Sample wizard is started. The files are sampled in the order they were listed under Sample File.

6. The Flat File Sample wizard enables you to view a sample of the flat file while you are defining it. Each step of the wizard enables you to design the definition and verify that the definition is correct. See "Using the Flat File Sample Wizard" on page 3-13 for more information.

   For binary files, you may prefer using the Create Flat File wizard. See "Using the Create Flat File Wizard" on page 3-6.

   The wizard creates definitions for the files, stores the definitions in the flat file module, and inserts the file names under the flat file module in the Projects Navigator.

3.3.1 Using the Flat File Sample Wizard

Use the Flat File Sample Wizard as an aid in defining metadata for flat files.

This wizard samples delimited and fixed format files. It does not sample multibyte character file with a fixed record format. For these and other files containing non-displayable data, such as binary files, see "Using the Create Flat File Wizard" on page 3-6.

After you complete the Flat File Sample Wizard, the metadata is defined in the workspace and you can use the flat files as source or target operators in a mapping. For more information about mappings, see the *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

3.3.1.1 Flat File Wizard for Simple Flat Files

For simple flat files that are delimited and contain a single record type, the Flat File wizard guides you through the following tasks:

- "Describing the Flat File"
- "Specifying the Record Organization"
- "Specifying Field Properties"

3.3.1.2 Describing the Flat File

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

- **Name**: This name uniquely identifies the file in the module. By default, the wizard creates a name based on the name of the source file by replacing invalid characters with an underscore. For example, if the file name is `myfile.dat`, the wizard assign the workspace name `myfile_dat`.

  If you rename the file, do not include a space or any punctuation in the name. You can include an underscore. You can use uppercase and lowercase letters. Do not start the name with a digit. Do not start a name with the reserved prefix `OWB$`.

- **Description**: You can enter an optional description for the file.

- **Character set**: Character sets determine what languages can be represented in database objects and files. The default Globalization Support character set matches the character set defined for the computer hosting Oracle Warehouse Builder. If the character set differs from that of the source file, the data sample might appear unintelligible. You can display the data sample in the character set native to the source by selecting it from the list. For complete information about NLS character sets, see *Oracle Database Globalization Support Guide*.
- **Number of characters to sample**: This value specifies the number of characters that is read and displayed. The number of characters that is read cannot be canceled, so ensure that you pick a reasonable number of characters. If you are sampling a multi-record file, then ensure that the sample is large enough to include at least one of each type. By default, the wizard samples the first 10000 characters. To determine an optimum value for this field, see "Example: Flat File with Multiple Record Types" on page 3-18.
- **Advanced**: Do not click the **Advanced** button for simple flat files. The advanced option is required only for complex flat files as described in "Flat File Wizard For Complex Flat Files".

Click **Next** to continue with "Specifying the Record Organization". At all steps, the wizard updates the sample displayed at the bottom of the wizard page. You can use the scroll bars to see the sample data.

3.3.1.3 Specifying the Record Organization

Specify the following properties:

- **Records delimited by**: Select this option if the end of each record is designated by a delimiter. Then specify that record delimiter. You can accept the default record delimiter, carriage return (<CR>), or you can enter a new value. For symbols other than \n and <CR>, specify the hexadecimal value of the character used as the delimiter.
- **Record length (in characters)**: Select this option if each record in the file is the same length. Then specify the number of characters in each record. For files with multibyte characters, count a multibyte character as one character.
- **Field delimiter**: Field delimiters designate where one field ends and another begins. You can enter a field delimiter or select one from the list. The list displays common field delimiters. However, you may enter any character as a delimiter except the ones used for enclosures. The default is the comma (,). You can also specify multiple characters and hexadecimal characters as a field delimiter.
- **Enclosures (Left and Right)**: Some delimited files contain enclosures that denote text strings within a field. If the file contains enclosures, enter an enclosure character in the text box or select one from the list. The list displays common enclosures. However, you may enter any character. The default for both the left and right enclosure is the double quotation mark ("). Multiple characters and hexadecimal characters can be specified as field enclosures.

3.3.1.4 Specifying Field Properties

Use the Field Properties page in the Flat File Sample Wizard to define properties for each field. The wizard assigns a name to each field. It assigns 'C1'to the first field, 'C2' to the second, and so on. To rename fields, click a field and enter a new name.

For single record file types, you can instruct the wizard to use the first record to name the fields. Indicate this by selecting the **Use the first record as the field names** box.

The Field Properties page shows both "SQL*Loader Properties" and "SQL Properties". Use the scroll bar to scroll to the right and view all the properties.

The wizard deactivates properties that do not apply to a given data type. For example, you can edit the length for a CHAR, but precision and scale are not available. Deactivated properties are grayed out.

3.3.1.4.1 SQL*Loader Properties The first set of properties the wizard displays are for the SQL*Loader utility. When you use the flat file directly as a source in a mapping, SQL*Loader and the properties you set here are used. SQL*Loader properties include details of how the following are mapped to a relational table: "Type", "Length", "Precision", "Scale", "Mask", "NULLIF", and "DEFAULTIF". See *Oracle Database Concepts* for more details.

3.3.1.4.2 SQL Properties The second set of properties are the SQL properties that include mapping details for "SQL Type", "SQL Length", "SQL Precision", and "SQL Scale". These properties specify how the fields in a flat file translate to the columns in a relational table. See "SQL Properties" on page 3-11 for more details.

3.3.1.5 Flat File Wizard For Complex Flat Files

For complex files, the Flat File wizard guides you through the following tasks. The advanced mode enables you to define files with fixed length fields (in addition to delimited), files that contain multiple record types, or files that use logical records (multiple physical records per logical record).

- "Describing the Flat File"
- "Selecting the Record Organization"
- "Selecting the File Format"
- "Selecting the File Layout"
- "Selecting Record Types (Multiple Record Type Files Only)"
- "Specifying Field Lengths (Fixed-Length Files Only)"
- "Specifying Field Properties"

3.3.1.6 Describing the Flat File

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

- **Name**: This name uniquely identifies the file in the workspace. By default, the wizard creates a name based on the name of the source file by replacing invalid characters with an underscore. For example, if the file name is `myfile.dat`, the wizard assign the workspace name `myfile_dat`.

  If you rename the file, do not include a space or any punctuation in the name. You can include an underscore. You can use uppercase and lowercase letters. Do not start the name with a digit. Do not start a name with the reserved prefix `OWB$`.

- **Description**: You can enter an optional description for the file.

- **Character set**: Character sets determine what languages can be represented in database objects and files. The default Globalization Support character set matches the character set defined for the computer hosting Oracle Warehouse Builder. If the character set differs from that of the source file, the data sample might appear unintelligible. You can display the data sample in the character set native to the source by selecting it from the list. For complete information about NLS character sets, see *Oracle Database Globalization Support Guide.*

- **Number of characters to sample**: You can indicate the number of characters for the wizard to sample from the data file. By default, the wizard samples the first 10000 characters. To determine an optimum value for this field, see "Example: Flat File with Multiple Record Types" on page 3-18.

Click **Advanced** to continue with "Selecting the Record Organization". At all steps, the wizard updates the sample displayed at the bottom of the wizard page. You can use the scroll bars to see the sample data.

3.3.1.7 Selecting the Record Organization

Use the Record Organization page to indicate how records are organized in the file you are sampling. Select between the two options to indicate how the length of each record in the file is determined:

- **Records delimited by:** If the end of each record is designated by a delimiter, then specify that record delimiter. You can accept the default record delimiter, carriage return (<CR>), or enter a new value. You can specify multiple characters and hexadecimal characters as a record delimiter. If the delimiter is a symbol other than `\n` or `<CR>`, then specify the hexadecimal character of the symbol. The hexadecimal character format is `x'<hexadecimal string>'` or `X'<hexadecimal string>'`.
- **Record length (in characters):** Select this option if each record in the file is the same length. Then specify the number of characters in each record. For files with multibyte characters, count a multibyte character as one character.

3.3.1.7.1 Specifying Logical Records The Flat File Sample Wizard enables you to sample files composed of logical records that correspond to multiple physical records. If the file contains logical records, select **File contains logical records**. Then select one of the options to describe the file.

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

- **Number of physical records for each logical record:** The data file contains a fixed number of physical records for each logical record.

```
PHYSICAL_RECORD1
PHYSICAL_RECORD2
PHYSICAL_RECORD3
PHYSICAL_RECORD4
```

  In the preceding example, if the number of physical records for each logical record is 2, then `PHYSICAL_RECORD1` and `PHYSICAL_RECORD2` form one logical record and `PHYSICAL_RECORD3` and `PHYSICAL_RECORD4` form a second logical record.

- **End character of the current physical record**: The data file contains a variable number of physical records with a continuation character at the end that signifies that the record is continued in the next physical record.

  In the following example, the continuation character is a percentage sign (%) after the record.

```
PHYSICAL_RECORD1%
PHYSICAL_RECORD2       end log rec 1
PHYSICAL_RECORD3%
PHYSICAL_RECORD4       end log rec 2
```

- **Start character of the next physical record**: The data file contains a variable number of physical records with a continuation character at the beginning of each

physical record that signifies that the record is a continuation of the previous physical record.

The following example shows two logical records with a continuation character at beginning of the record.

```
PHYSICAL_RECORD1
%PHYSICAL_RECORD2      end log rec1
PHYSICAL_RECORD3
%PHYSICAL_RECORD4      end log rec 2
```

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

```
PHYSICAL_RECORD1
%PHYSICAL_RECORD2
%PHYSICAL_RECORD25
%PHYSICAL_RECORD26   (end log record 1)
PHYSICAL_RECORD3
%PHYSICAL_RECORD4
%PHYSICAL_RECORD45
%PHYSICAL_RECORD46 (end log record 2)
```

After you complete the logical record information, click **Next** to continue with the wizard.

3.3.1.8 Selecting the File Format

Use the File Format page to select between **Fixed Length** and **Delimited** formats for the fields in the file. The Flat File Sample Wizard does not sample multibyte character files with a fixed record format. For such files, use the Create Flat File wizard. For more details, see "Using the Create Flat File Wizard" on page 3-6.

When you select a file format, the wizard updates the sample displayed at the bottom of the wizard page. You can use the scroll bars to navigate the sample data.

Fields in a file can either be of fixed length or delimited.

For fixed length fields, select **Fixed Length**. If you select this option then you must define the field lengths in the Field Lengths page. See "Specifying Field Lengths (Fixed-Length Files Only)" on page 3-19.

When the fields are delimited, specify the following properties:

- **Field delimiter**: Field delimiters designate where one field ends and another begins. You can enter a field delimiter or select one from the list. The list displays common field delimiters. However, you may enter any character as a delimiter except the ones used for enclosures. The default is the comma (,). You can also specify multiple characters and hexadecimal characters as a field delimiter. The hexadecimal character format is `x'<hexadecimal string>'` or `X'<hexadecimal string>'`.
- **Enclosures (Left and Right)**: Some delimited files contain enclosures that denote text strings within a field. If the file contains enclosures, enter an enclosure character in the text box or select one from the list. The list displays common enclosures. However, you may enter any character. The default for both the left and right enclosure is the double quotation mark ("). Multiple characters and hexadecimal characters can be specified as field enclosures.

Click **Next** to continue with the wizard.

3.3.1.9 Selecting the File Layout

Use the File Layout page to specify the number of rows to skip and to select between a single record type versus multiple record types.

Indicate the number of records to skip in **Skip rows**. This is useful for skipping over unwanted header information. If one of the records includes field names, skip the preceding header records so that the record containing field names is displayed as the first record in the file. Later in the wizard, on the Field Properties page, you can instruct the wizard to use that record for field names if you are defining a single record file type.

Indicate whether the file contains a single record type or multiple record types. Later in the wizard you can instruct the wizard to scan the file for the record types. For more information about multiple record types, see "Selecting Record Types (Multiple Record Type Files Only)" on page 3-18.

3.3.1.10 Selecting Record Types (Multiple Record Type Files Only)

Use the Record Types wizard page to scan the flat file for record types, add or delete record types, and assign type values to the record types.

> **Note:** This step in not used for files with a single record type. If the data file has a single record type and fixed length file format, proceed to "Specifying Field Lengths (Fixed-Length Files Only)" on page 3-19. If the data file has a single record type and delimited file format, proceed to "Specifying Field Properties" on page 3-20.

3.3.1.10.1 Example: Flat File with Multiple Record Types In files with multiple record types, one of the fields distinguishes one record type from the next. When you use the Flat File Sample Wizard, you instruct the wizard to scan a specified field of every record for the record type values.

Figure 3–4 shows an example of a comma delimited file with two record types, "m" and "f". In this case, instruct the wizard to scan the third field. The wizard returns "m" and "f" as the type values.

Figure 3–4 Example of a File with Multiple Record Types

```
james,28,m,london
joan,24,f,paris
jimmy,30,m,delhi
tanya,36,f,sydney
jeevas,31,m,rhode island
hugh,27,m,dublin
rick,34,m,hobart
claire,24,f,kingston
brianna,24,f,lima
```

When you use the wizard to sample flat files with multiple record types, ensure that the sample size you specified on the Name page is large enough to include each record type at least once. The default is 10000 characters.

If you do not see all of the required record types in the display area, you must specify a larger sample size on the Name page. Ensure that the sample size is large enough to include all record types. If all record types do not appear within a reasonable number

of characters, you can mock up a sample file with rows selected from different parts of the master file to provide a representative set of data. If you know the record layout well, you can scan a representative sample and then manually add new record types.

3.3.1.10.2 Defining Multiple Record Organization in a Delimited File

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

To complete the Records Type page for a delimited file:

1. Select the one field that identifies the record types in the file.

   The wizard displays all the fields in a sample in the lower panel of the page. In the Field position, you can enter the position as it appears in the sample. Unless you specify otherwise, the wizard defaults to the first field in the file.

   If you click **Scan**, then the wizard scans the file for the field and displays the type values. The wizard assigns default record names (RECORD1, RECORD2...) to each type value.

2. You can edit the record names and the type value.

   Click a record name to rename it or select a different record name from the list. You cannot associate a record name with multiple record type values.

3. Click **Next** to continue with the wizard.

3.3.1.10.3 Defining Multiple Record Organization in a Fixed-Length File

When a fixed-length flat file contains several different types of records, you can use the scanning feature within the Flat File Sample Wizard to search for record types and assign a type value to each record type.

To complete the Records Type page for a fixed-length file:

1. Specify the one field that identifies the record types in the file. Use the ruler or enter values for the **Start position** and **End position.** To scan for records based on the first field, enter 0 for **Start Position.**

   The wizard indicates the selected field with a red check mark in the ruler in the file sample in the lower panel of the page.

2. Click **Scan.**

   The wizard scans the file field and displays the type values. The wizard assigns default record names (RECORD1, RECORD2...) to each type value.

3. You can edit the record names and type value.

   Click a record name to rename it or select a different record name from the list. You cannot associate a record name with multiple record type values.

4. Click **Next** to continue with the wizard.

3.3.1.11 Specifying Field Lengths (Fixed-Length Files Only)

When you use the Flat File Sample Wizard to define a fixed-length flat file, you must define the length of each field in the file.

> **Note:** This step is not necessary for delimited files. Proceed to "Specifying Field Properties" on page 3-20.

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

If you know the length of each field, enter the field length in **Field Lengths**. Separate each length by commas. The wizard displays the changes to the sample at the bottom of the wizard page.

To use the ruler, click any number or hash mark on the ruler. The wizard displays a red check mark on top of the ruler and marks the boundary with a red line. If you make a mistake, double-click the marker to delete it or move the marker to another position. Use the ruler to create markers for each field in the file.

3.3.1.11.1 Specifying Field Lengths for Multiple Record Files You can select the record type by name from **Record Name**. Or, you can select **Next Record Type** from the lower right corner of the wizard page. The number of records with unspecified field lengths is indicated on the lower left corner of the wizard page.

If the flat file contains multiple record types, the wizard prompts you to specify field lengths for each record type before continuing.

3.3.1.12 Specifying Field Properties

Use the Field Properties page in the Flat File Sample Wizard to define properties for each field. The wizard assigns a name to each field. It assigns 'C1'to the first field, 'C2' to the second, and so on. To rename fields, click a field and enter a new name.

For single record file types, you can instruct the wizard to use the first record in the file to name the fields. Indicate this by selecting the **Use the first record as the field names** box.

The Field Properties page shows both "SQL*Loader Properties" and "SQL Properties". Use the scroll bar to scroll to the right and view all the properties.

The wizard deactivates properties that do not apply to a given data type. For example, you can edit the length for a `CHAR`, but precision and scale are not available. Deactivated properties are grayed out.

3.3.1.12.1 SQL*Loader Properties The first set of properties the wizard displays are for the SQL*Loader utility. When you use the flat file directly as a source in a mapping, SQL*Loader and the properties you set here are used. SQL*Loader properties include "Type", "Length", "Precision", "Scale", "Mask", "NULLIF", and "DEFAULTIF". See "SQL*Loader Properties" on page 3-15 for more details.

3.3.1.12.2 SQL Properties The second set of properties are the SQL properties that include "SQL Type", "SQL Length", "SQL Precision", and "SQL Scale". These properties specify how the fields in a flat file translate to the columns in a relational table. See "SQL Properties" on page 3-15 for more details.

3.4 Importing Metadata Definitions from COBOL Copybooks

COBOL programmers create files by defining the physical files and the logical records that is used to build those files. The records may be defined within the COBOL program itself, but are usually defined in separate files, called copybooks. These copybooks specify the layout and format of the user data, but do not specify the physical characteristics of the file itself. The physical characteristics of the file identify how the file is organized and accessed. For example, whether records are terminated with CR or CR/LF, is not part of the user data definition and is therefore not included in the record definition.

With Oracle Warehouse Builder, you can import metadata from COBOL copybooks. Oracle Warehouse Builder automatically manages the following operations:

- Defining the fields for data storage
- Calculating the data positions
- Transforming COBOL data characteristics to the appropriate data type definitions in SQL*Loader

3.4.1 Understanding Data Hierarchy in COBOL Applications

COBOL records are defined as a set of data elements and groups. A data element is an atomic data item. A group is a container for data elements. Each item defined in a COBOL record is called a field whether it is a group or an elementary item. Each field definition contains a level number which reflects the hierarchy of the data within the record. Groups can contain other groups or elementary items. Items contained in a group are called subordinate elements. Field definitions for elementary items contain complete metadata for the item primarily specified in picture and usage clauses. A group inherits characteristics from its subordinate elements and does not generally contain metadata specification details. An example of a simple copybook is given in Example 3–1.

Example 3–1 COBOL Copybook

```
01 EMPLOYEE-RECORD.
   05 EMP-ID PIC 9(6).
   05 EMP-REGION PIC 9.
   05 EMP-DEPT PIC 999.
   05 EMP-HIRE-DATE.
      10 EMP-HIRE-DATE-MM PIC 99.
      10 EMP-HIRE-DATE-DD PIC 99.
      10 EMP-HIRE-DATE-YYYY PIC 9999.
   05 EMP-SALARY PIC 9(9).
   05 EMP-NAME PIC X(15).
```

The above example shows the definition of EMPLOYEE-RECORD. It contains 6 fields defined at the 05 Level. All of the fields except EMP-HIRE-DATE are elementary items. The elementary items contain picture clauses that define their data-characteristics. EMP-HIRE-DATE is a group field with subordinate elements at the 10 level. Each 10 level field is an elementary item that contains a picture clause defining its data characteristics. The date can be referenced as a whole by using the EMP-HIRE-DATE group field. This field includes the month, day, and year elements. Each subordinate field can also be referenced individually, allowing access to just the year for example.

3.4.1.1 COBOL Data Types

The USAGE and PICTURE clauses are used to define the format and characteristics of data elements. If a USAGE clause is not specified, the data is in DISPLAY format, either external numeric or external character. When considered, the picture and usage identify the data type. Detailed information concerning COBOL data types and how they are mapped to relational data types is described inTable 3–1.

3.4.1.2 Arrays Defined on Fields

COBOL provides support for both arrays and varying arrays. These complex structures are identified by the use of an OCCURS clause. Varying arrays are defined with the additional specification of a DEPENDING ON clause. For arrays, the OCCURS clause indicates the number of elements in the array. For varying arrays, the occurs specification includes a range of elements from x TO y and the DEPENDING ON clause identifies a field that contains the actual number of elements in the array.

An array or a varying array can be defined on an elementary field or a group. Example 3–2 provides an example of an array defined on elementary fields.

Example 3–2 Arrays Defined on Elementary Fields

```
01 EMPLOYEE-RECORD.
   05 EMP-ID PIC 9(6).
   05 EMP-REGION PIC 9.
   05 EMP-DEPT PIC 999.
   05 EMP-HIRE-DATE.
      10 EMP-HIRE-DATE-MM PIC 99.
      10 EMP-HIRE-DATE-DD PIC 99.
      10 EMP-HIRE-DATE-YYYY PIC 9999.
   05 EMP-SALARY PIC 9(9).
   05 EMP-NAME PIC X(15).
   05 EMP-SKILL-LEVEL PIC 99 OCCURS 4 TIMES.
   05 EMP-SKILL-ID PIC 9(4) OCCURS 4 TIMES.
```

Two independent arrays are defined, one on EMP-SKILL_LEVEL and one on EMP-SKILL_ID. In this example, the record contains four occurrences of EMP-SKILL-LEVEL, followed by four occurrences of EMP-SKILL-ID. In the file, each record is constructed as: EMP_SKILL_LEVEL, EMP_SKILL_LEVEL, EMP_SKILL_ LEVEL, EMP_SKILL_LEVEL, EMP_SKILL_ID, EMP_SKILL_ID, EMP_SKILL_ID, EMP_SKILL_ID.

Example 3–3 provides an example of a varying array defined on elementary fields.

Example 3–3 Varying Array Defined on Elementary Fields

```
01 EMPLOYEE-RECORD.
   05 EMP-ID PIC 9(6).
   05 EMP-REGION PIC 9.
   05 EMP-DEPT PIC 999.
   05 EMP-HIRE-DATE.
      10 EMP-HIRE-DATE-MM PIC 99.
      10 EMP-HIRE-DATE-DD PIC 99.
      10 EMP-HIRE-DATE-YYYY PIC 9999.
   05 EMP-SALARY PIC 9(9).
   05 EMP-NAME PIC X(15).
   05 EMP-SKILL-COUNT PIC 99.
   05 EMP-SKILL-LEVEL PIC 99 OCCURS 1 TO 4 TIMES.
      DEPENDING ON EMP-SKILL-COUNT.
   05 EMP-SKILL-ID PIC 9(4) OCCURS 1 TO 4 TIMES.
      DEPENDING ON EMP-SKILL-COUNT.
```

Two independent arrays are defined, one on EMP-SKILL_LEVEL and one on EMP-SKILL_ID. In this example, the value in EMP-SKILL-COUNT determines the number of occurrences in both arrays. In the file, a record with EMP_SKILL_COUNT equal to one is constructed as: EMP_SKILL_LEVEL, EMP_SKILL_ID. If the value of EMP-SKILL-COUNT is two, then two occurrences of EMP-SKILL-LEVEL are followed by two occurrences of EMP-SKILL-ID. In the file, a record with EMP_SKILL_COUNT equal to two is constructed as: EMP_SKILL_LEVEL, EMP_SKILL_LEVEL, EMP_ SKILL_ID, EMP_SKILL_ID.

Example 3–4 provides an example of an array defined on a group field.

Example 3–4 Array Defined on a Group Field

```
01 EMPLOYEE-RECORD.
   05 EMP-ID PIC 9(6).
```

```
   05 EMP-REGION PIC 9.
   05 EMP-DEPT PIC 999.
   05 EMP-HIRE-DATE.
      10 EMP-HIRE-DATE-MM PIC 99.
      10 EMP-HIRE-DATE-DD PIC 99.
      10 EMP-HIRE-DATE-YYYY PIC 9999.
   05 EMP-SALARY PIC 9(9).
   05 EMP-NAME PIC X(15).
   05 EMP-SKILLS OCCURS 4 TIMES.
      10 EMP-SKILL-LEVEL PIC 99.
      10 EMP-SKILL-ID PIC 9(4).
```

In this example an array is defined with four elements. Each element contains one occurrence of each field: EMP_SKILL_LEVEL and EMP_SKILL_ID. In the file, each record is constructed as: EMP_SKILL_LEVEL, EMP_SKILL_ID, EMP_SKILL_LEVEL, EMP_SKILL_ID, EMP_SKILL_LEVEL, EMP_SKILL_ID, EMP_SKILL_LEVEL, EMP_SKILL_ID.

Example 3–5 provides an example of a varying array defined on a group field.

Example 3–5 Varying Array Defined on a Group Field

```
01 EMPLOYEE-RECORD.
   05 EMP-ID PIC 9(6).
   05 EMP-REGION PIC 9.
   05 EMP-DEPT PIC 999.
   05 EMP-HIRE-DATE.
      10 EMP-HIRE-DATE-MM PIC 99.
      10 EMP-HIRE-DATE-DD PIC 99.
      10 EMP-HIRE-DATE-YYYY PIC 9999.
   05 EMP-SALARY PIC 9(9).
   05 EMP-NAME PIC X(15).
   05 EMP-SKILL-COUNT PIC 99.
   05 EMP-SKILLS OCCURS 4 TIMES DEPENDING ON EMP-SKILL-COUNT.
     10 EMP-SKILL-LEVEL PIC 99.
     10 EMP-SKILL-ID PIC 9(4).
```

In this example, one array is defined with up to four elements. The value in EMP-SKILL-COUNT defines the number of occurrences of the array. Therefore, if the value of EMP-SKILL-COUNT is set as one, then there is one occurrence of EMP_SKILLS. In the file, a record with EMP_SKILL_COUNT equal to one is constructed as: EMP_SKILL_LEVEL, EMP_SKILL_ID.

If the value of EMP-SKILL-COUNT is two, then there are two occurrences of EMP-SKILLS. In the file, a record with EMP_SKILL_COUNT equal to two is constructed as: EMP_SKILL_LEVEL, EMP_SKILL_ID, EMP_SKILL_LEVEL, EMP_SKILL_ID.

3.4.2 Multiple Definitions

In COBOL, data in a record may have multiple definitions. You can use any one of these definitions to access the data. There are three methods for getting multiple definitions:

- "Defining Multiple Records"
- "Defining Group Fields"
- "Redefining Fields and Groups"

3.4.2.1 Defining Multiple Records

COBOL generated files can contain multiple record types. Whenever there is multiple level 01 item in a file definition, each level 01 provides a separate definition of the data record area. Only one record is kept in the data record area at a time, so only one level 01 definition is used at a time. For example, a file may contain two types of records, department records and employee records. Level 01 items are defined for both department and employee records. The hierarchy for each record immediately follows the level 01 item for that record and provides the definitions for the entire record. Each record contains a field that identifies the record type. This record type is at the same position for all record definitions.

In the following example, the record type is in the first position:

Example 3–6 Copybook With Multiple Records

```
01   DEPARTMENT-RECORD.
      05 DEPT-RECORD-TYPE PIC X.
      05 DEPT-ID PIC 999.
      05 DEPT-NAME PIC X(30).
      05 DEPT-DESCRIPTION PIC X(160).
01   EMPLOYEE-RECORD.
      05 EMP-RECORD-TYPE PIC X.
      05 EMP-ID PIC 9(6).
      05 EMP-NAME PIC X(30).
      05 EMP-REGION PIC 9.
      05 EMP-DEPT PIC 999.
```

3.4.2.2 Defining Group Fields

As seen in the section on Data Hierarchy, fields can be organized in groups. These groups actually provide an additional definition of the fields and are used to access the data.

3.4.2.3 Redefining Fields and Groups

COBOL provides the ability to redefine a field or a group. Redefinition does not define data at a new location, but instead provides an additional definition of data characters that have been previously defined.

Example 3–7 Redefining a Field

```
01  EMPLOYEE-RECORD.
   05 EMP-ID PIC 9(6).
   05 EMP-ID-R REDEFINES EMP-ID.
      10 EMP-ID-GROUP PIC 99.
      10 EMP-ID-NUM PIC 9999.
   05 EMP-REGION PIC 9.
   05 EMP-DEPT PIC 999.
```

In the above example, the EMP-ID field is defined as a 6 digit numeric field. A redefinition is provided dividing the field into two fields: EMP-ID-GROUP is defined as the first two digits of the EMP-ID field. EMP-ID-NUM is defined as the last four digits of the EMP-ID field. EMP-ID-NUM and EMP-ID-GROUP both begin at position 1 in the record.

Example 3–8 Redefining a Group

```
01 EMPLOYEE-RECORD.
```

```
05 EMP-ID PIC 9(6).
05 EMP-ID-R REDEFINES EMP-ID.
10 EMP-ID-GROUP PIC 99.
10 EMP-ID-NUM PIC 9999.
05 EMP-REGION PIC 9.
05 EMP-DEPT PIC 999.
05 EMP-HIRE-DATE.
10 EMP-HIRE-DATE-MM PIC 99.
10 EMP-HIRE-DATE-DD PIC 99.
10 EMP-HIRE-DATE-YYYY PIC 9999.
05 EMP-SALARY PIC 9(9).
05 EMP-NAME PIC X(15).
05 EMP-SKILLS OCCURS 4 TIMES.
10 EMP-SKILL-LEVEL PIC 99.
10 EMP-SKILL-ID PIC 9(4).
05 EMP-SKILLS-R REDEFINES EMP-SKILLS.
10 EMP-SKILL-LEVEL1 PIC 99.
10 EMP-SKILL-ID1 PIC 9(4).
10 EMP-SKILL-LEVEL2 PIC 99.
10 EMP-SKILL-ID2 PIC 9(4).
10 EMP-SKILL-LEVEL3 PIC 99.
10 EMP-SKILL-ID3 PIC 9(4).
```

In the above example, the EMP-SKILL array has been redefined so that each element is expanded providing a field for each element. As in the previous example, the definitions for EMP-SKILLS-R is defining the same data area as EMP-SKILLS. The first occurrence of EMP-SKILL-LEVEL is at the same position in the record as EMP-SKILL-LEVEL1.

3.4.2.4 COBOL File Formats

COBOL programs can create files of different organization. These include the following:

- **Line Sequential**: Line sequential files are generally known as text files because the primary use of this file type is for display data. The records in these files can only be accessed in the order they were written. Line sequential files contain variable length records. A record delimiter separates each record in the file. The record delimiter that is used is operating system dependent and is inserted after the last character in each record.
- **Record Sequential**: Record sequential files are also accessed in the order they were written. This file organization is more flexible than line sequential. Records can be of fixed or variable length. The record sequential organization is used for sequential files that contain binary or packed data, or any data that may have other non-printable characters. In fixed length files, every record that is written to the file is the same length. If necessary, the record is padded with blanks to ensure uniform length. With variable length records, each record is written based on the actual size of the record. A Record Descriptor Word (RDW) is inserted at the beginning of each record. The RDW contains the actual length of the record. It is not considered part of the record and is not included in the data definition. In general, variable length records are used when there are many small records and few large records. Variable length records must be converted before they can be imported.
- **Relative Files**: Relative files can be accessed randomly in the order they are written. Records can be declared as variable, but they are written as fixed. The random access is not by key, but is instead by relative record number. Relative files must be converted to sequential before they can be imported.

- **Indexed Files**: Indexed files can be accessed by key field(s) in the order they are written. Records in indexed files can be fixed or variable. Indexed files consist of two physical files, one containing the data and the other containing the index. Indexed files are converted to sequential before they are imported.

The file format used for data storage is determined by the COBOL application. This information is not defined in the copybook.

3.4.3 Reinterpreting COBOL Data Structures Into Relational Data Structures

As you consider importing COBOL files into a relational database, you must plan how the data should be mapped into the relational database.

Records

At the highest level, each record type, level 01 structure, is considered mapping to a table. Records in files are often designed to be independent sources of information. This is a large difference between files and tables. Tables are generally designed to hold information that is closely related. When you consider the records in your file, you should consider if it would be better to define multiple tables for the information. Groups that are used to organize related information are often good candidates for independent tables. For example, you may have a group that is defined for address information, which might naturally fit into a name and address table. Similarly, arrays, whether varying or not, are also often good candidates for independent tables.

Arrays

In order to define arrays, Oracle Warehouse Builder normalizes the array by specifying each element in the array independently. This technique is also used for defining varying arrays. Not all varying arrays can be loaded using SQL*Loader. When the varying array is after the record, SQL*Loader may be able to load it. Records with embedded varying arrays are not necessarily physically stored as variable, therefore you may be able to use this technique for loading these records also. See "Example: Extracting Data from a Single Record Type Varying Array COBOL File" on page 3-29.

Mapping COBOL Data Types to SQL Data Types

The `USAGE` and `PICTURE` clauses are used to define the format and characteristics of data elements. Together the picture and usage identify the scalar data type, length, precision and scale. Table 3–1 shows data element definitions that apply to COBOL files. It describes the representation of each data type and identifies how that data type is mapped to SQL*Loader data type definitions. The PICTURE represents a mask that describes the data. The values identified within the parentheses are multiplication factors for the preceding picture element. So when n = 5, X(n) indicates that there are 5 characters of type X (alphanumeric data).

Table 3–1 COBOL Data Types and the Equivalent SQL *Loader Data Types

COBOL Data Type	SQL Loader Data Type	Description
X(n)	■ CHAR(n) ■ DATE(n) 'mask' when data contains valid date ■ TIME ■ TIMESTAMP ■ INTERVAL	Alphanumeric data. Each X identifies one allowable character from the specified character set.

Table 3–1 (Cont.) COBOL Data Types and the Equivalent SQL *Loader Data Types

COBOL Data Type	SQL Loader Data Type	Description
A(n)	CHAR(n)	Alphabetic data. Each A identifies any letter of the alphabet or space.
9(n)	▪ INTEGER EXTERNAL(n) ▪ DECIMAL EXTERNAL(n) ▪ ZONED EXTERNAL(n) ▪ DATE(n) 'mask' when data contains valid date	Numeric data. Each 9 identifies one digit.
+- mantissa +- exponent	FLOAT EXTERNAL (length)	External floating point data.
S9(n)v9(m) SIGN TRAILING	ZONED(precision, scale) where precision = n+m and scale = m	Numeric data. Each 9 identifies one digit. The v indicates the implied decimal position. The sign is carried in the last byte.
9(n)v9(m)	ZONED(precision, scale) where precision = n+m and scale = m	Numeric data. Each 9 identifies one digit. The v indicates the implied decimal position.
9(n)v9(m)	SMALLINT	Internal format data with a radix of 2.
S9(n)v9(m)	INTEGER(length 2,4,or 8)	The size of the field varies with the value m.
	May use SIGNED |	n+m = 1-4, length = 2
	UNSIGNED	n+m = 5-9, length = 4
	May require BYTEORDER clause	n+m = 10-18, length =8
	Scale handled with an expression	
Not allowed	FLOAT May require BYTEORDER clause.	Single-precision floating point number, 4 bytes long
Not allowed	DOUBLE May require BYTEORDER clause.	Double-precision floating point number, 8 bytes long
9(n)v9(m) S9(n)v9(m)	DECIMAL (precision, scale) where precision = n+m and scale = m	Internal format numeric data with a radix of 10. The clause indicates that each digit must use the minimum storage possible. Generally, each byte contains two digits with the last half-byte containing the sign.
X(n) 9(n)v9(m) S9(n)v9(m)	Not commonly used	The internal format of the data is not defined. It is often stored the same as BINARY, however the radix may be reversed.
G(n)	GRAPHIC(n)	Graphic data that does not contain Shift In and Shift Out characters

*Table 3–1 (Cont.) COBOL Data Types and the Equivalent SQL *Loader Data Types*

COBOL Data Type	SQL Loader Data Type	Description
05 V 49 V-LN PIC S9(4) COMP 05 V-DATA PIC X(n)	VARCHAR(max length), can only be loaded correctly between systems where SMALLINT is the same size	Variable length character field
05 V 49 V-LN PIC S9(4) COMP 05 V-DATA PIC G(n)	VARGRAPHIC(max length), can only be loaded correctly between systems where SMALLINT is the same size.	Variable length Graphic data that does not contain Shift In and Shift Out characters

3.4.4 Importing Metadata

To import metadata from COBOL, you must create a flat file module and then import the metadata definitions from a Cobol copybook.

Importing a Copybook

To import metadata from a COBOL copybook:

1. Create a flat file module as described in "Creating Flat File Modules" on page 3-4. Provide the location details of the COBOL copybook.
2. Right-click the newly created module and select **Import**, **Cobol**.

   The Cobol Import dialog box is displayed.

   An alternative way to open the Cobol Import dialog box is to select the newly created module and then select **File**, **Import**, **Cobol**.
3. Click **Add Copybook** to browse for the copybook you must import.

   You can add multiple copybooks simultaneously. When you add a copybook, the **Copybook** field of the Import Copybooks spread table displays the directory path of the copybook. Use the **File** field to edit the name of the imported copybook. Use the **Description** field to add an optional description for the imported copybook.
4. Click **View Copybook** to view the metadata structure of the copybook being imported.
5. Click **Session Options** to open the Import Cobol Session Options dialog box. For details of values to be specified in this dialog box, see "Import Cobol Session Options" on page 3-28.
6. Click **OK** and on the Cobol Import window, click **Import**.

3.4.4.1 Import Cobol Session Options

You can use this dialog box to specify a template file and other properties. Based on the copybook you are importing, determine if the default physical file properties are acceptable. Or else, select a template file from which the physical characteristics are imported. You can edit some physical properties of the file after the import as well.

To specify a template file, select a file from the **Copy Flat File Properties From** list.

If you retain the Default Properties option from the **Copy Flat File Properties From** list, the file is defined as fixed format (not delimited) with the following properties:

- Skip 0 records
- Use first row as column name set to false
- Record delimiter set to \n
- One physical record for each logical record

If you import a multi-record copybook, then it is set to multi record.

If you select a file from the Copy Flat File Properties From list, then the physical characteristics of this file are used for the newly imported copybook.

Do Not Import Groups: To suppress the definition of group items. This ensures that only the subordinate items get imported. For example, if you have the following fields:

```
05  EMP-HIRE-DATE.
    10 EMP-HIRE-MONTH PIC 99.
    10 EMP-HIRE-DAY PIC 99.
    10 EMP-HIRE-YEAR PIC 9999.
```

When you select the Do Not Import Groups option, only three fields are created, EMP-HIRE-MONTH, EMP-HIRE-DAY, and EMP-HIRE-YEAR. When the option is not selected, then four fields, EMP-HIRE-DATE, EMP-HIRE-MONTH, EMP-HIRE-DAY, EMP-HIRE-YEAR are created at the time of import.

Do Not Import Redefines: To avoid import of redundant redefined fields. For example, if you have the following copybook definition:

```
05 HIRE_DATE.
10 HIRE_MONTH PIC 99.
10 HIRE_DAY PIC 99.
10 HIRE_YEAR PIC 9999.
05 HIRE_DATE_ALPHA REDEFINES HIRE_DATE.
10 HIRE_MONTH PIC XX.
10 HIRE_DAY PIC XX.
10 HIRE_YEAR PIC XXXX.
```

To import all the fields, clear the **Do Not Import Redefinitions** option. When the option is selected, only the first four fields, HIRE_DATE, HIRE_MONTH, HIRE_DAY, HIRE_YEAR are imported.

At the time of importing the metadata, it is recommended that you inspect the copybook for unnecessary metadata including grouped fields and redefined fields. You can also edit the file later to remove unnecessary definitions.

3.4.5 Example: Extracting Data from a Single Record Type Varying Array COBOL File

To extract data from a COBOL data file, you must first import the corresponding copybook. Consider the following copybook for import:

```
01  EMPLOYEE-RECORD.
    05  EMP-RECORD-LENGTH PIC 9(4).
    05  EMP-ID PIC 9(6).
    05  EMP-REGION PIC 9.
    05  EMP-DEPT PIC 999.
    05  EMP-HIRE-DATE.
        10 EMP-HIRE-MM PIC 99.
        10 EMP-HIRE-DD PIC 99.
        10 EMP-HIRE-YEAR PIC 9999.
    05  EMP-SALARY PIC 9(9).
    05  EMP-NAME PIC X(15).
    05  EMP-SKILLS OCCURS 4 TIMES.
```

```
10 EMP-SKILL-LEVEL PIC 99.
10 EMP-SKILL-ID PIC 9999.
```

3.4.5.1 Importing The COBOL Copybook

Import the copybook as described in "Importing a Copybook" on page 3-28.

To set the file properties of the imported copybook, right-click the file on the Projects Navigator and click **Open**. On the Edit Flat File dialog box, define the file properties as given in "Defining the File Properties" on page 3-30.

3.4.5.2 Defining the File Properties

Use the Name, General, and Structure tabs to specify the file properties.

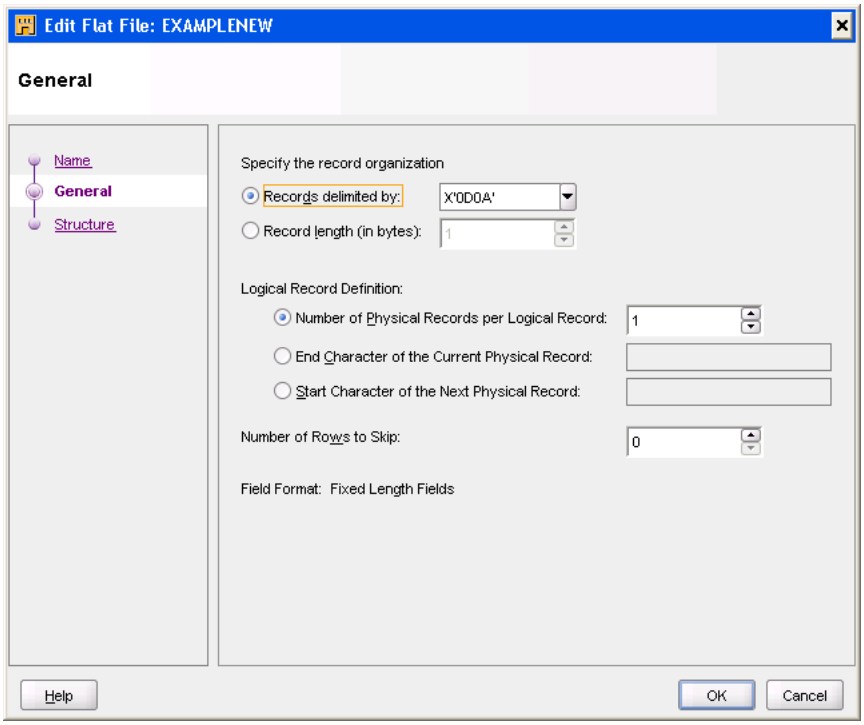
3.4.5.2.1 Name Tab Use the Name tab to specify the character set. This is an EBCDIC file, so the character must be set to `WE8EBCDIC500`.

3.4.5.2.2 General Tab Use the General tab to set the following file properties.

- **Record Delimiter**: The record delimiter for this file must be set to the binary value `X'0D0A'`.
- **Logical Record Definition**: In this example, there is one logical record per physical record.
- **Field Format**: Each field in this record is at a constant position. Therefore, the Fixed Length Fields option is selected by default.
- **Record Type**: This copybook contains a single type of record. Therefore, the Single Record option is selected by default.

The field properties are as shown in Figure 3–5.

Figure 3–5 The Field Properties of the Imported Cobol File



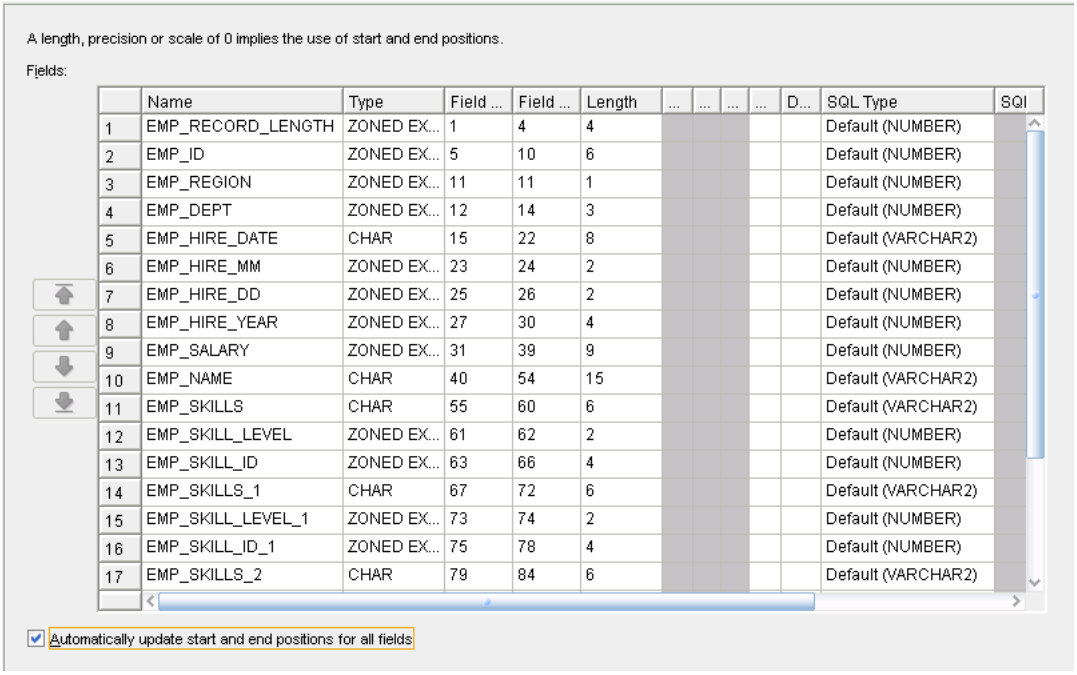
The figure shows the General Tab of the Edit Flat File dialog box. It lists out the field properties set for the file.

****

3.4.5.2.3 Structure Tab Use the Structure tab to review the field properties and make changes if required.

The structure of the imported file is as shown in Figure 3–6.

Figure 3–6 Structure of the Imported Cobol File

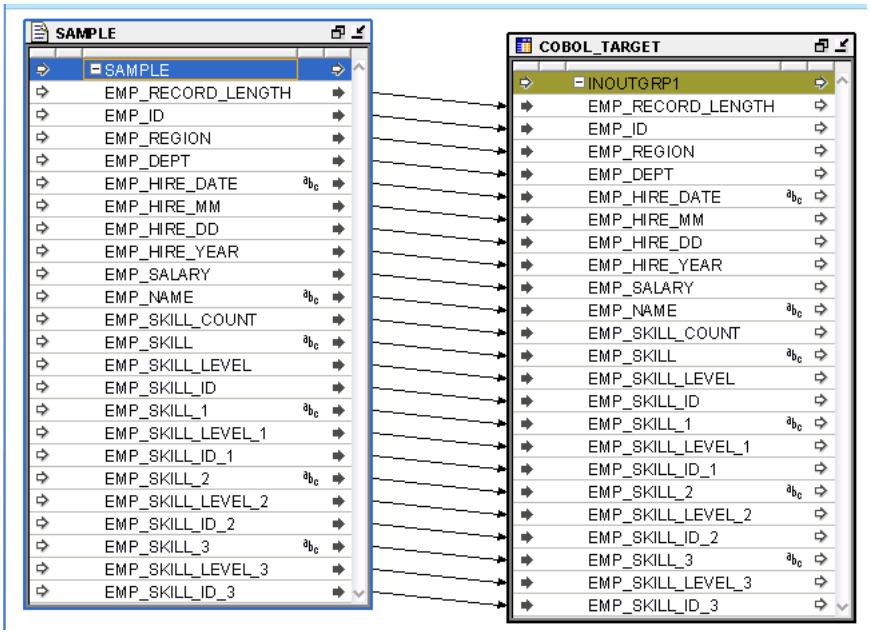


	Name	Type	Field ...	Field ...	Length	...	...	...	...	D...	SQL Type	SQI
1	EMP_RECORD_LENGTH	ZONED EX...	1	4	4						Default (NUMBER)	
2	EMP_ID	ZONED EX...	5	10	6						Default (NUMBER)	
3	EMP_REGION	ZONED EX...	11	11	1						Default (NUMBER)	
4	EMP_DEPT	ZONED EX...	12	14	3						Default (NUMBER)	
5	EMP_HIRE_DATE	CHAR	15	22	8						Default (VARCHAR2)	
6	EMP_HIRE_MM	ZONED EX...	23	24	2						Default (NUMBER)	
7	EMP_HIRE_DD	ZONED EX...	25	26	2						Default (NUMBER)	
8	EMP_HIRE_YEAR	ZONED EX...	27	30	4						Default (NUMBER)	
9	EMP_SALARY	ZONED EX...	31	39	9						Default (NUMBER)	
10	EMP_NAME	CHAR	40	54	15						Default (VARCHAR2)	
11	EMP_SKILLS	CHAR	55	60	6						Default (VARCHAR2)	
12	EMP_SKILL_LEVEL	ZONED EX...	61	62	2						Default (NUMBER)	
13	EMP_SKILL_ID	ZONED EX...	63	66	4						Default (NUMBER)	
14	EMP_SKILLS_1	CHAR	67	72	6						Default (VARCHAR2)	
15	EMP_SKILL_LEVEL_1	ZONED EX...	73	74	2						Default (NUMBER)	
16	EMP_SKILL_ID_1	ZONED EX...	75	78	4						Default (NUMBER)	
17	EMP_SKILLS_2	CHAR	79	84	6						Default (VARCHAR2)	

3.4.5.3 Creating a Mapping to Load Data

Create a mapping with the imported COBOL file as the source. Insert an unbound table operator in the mapping and map the desired fields from the COBOL file to the table operator.

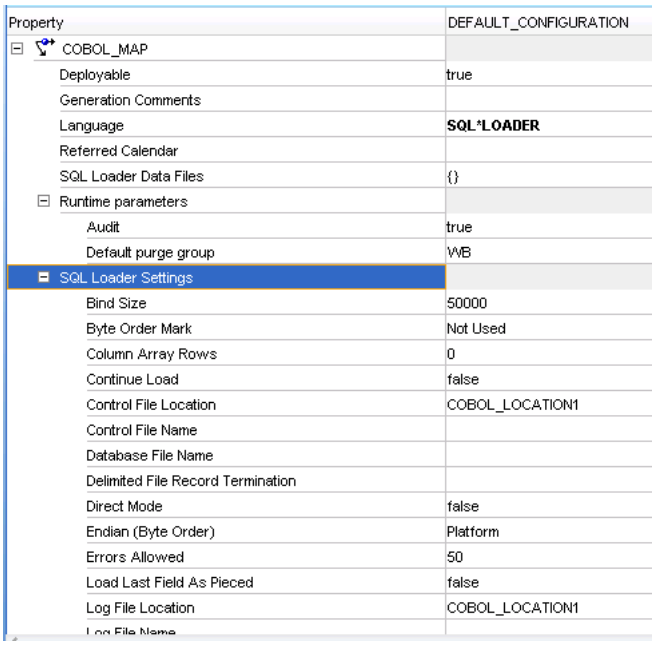
Figure 3–7 Mapping a Flat File to a Table



3.4.5.4 Configuring The Mapping

Right-click the mapping and click **Configure** to open the Configuration Properties dialog box. Select **SQL*LOADER** as the language for code generation. Tune any of the other SQL*Loader settings.

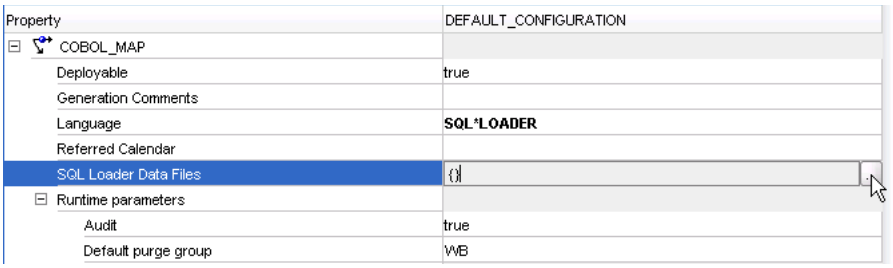
Figure 3–8 Configuration Properties Dialog Box



3.4.5.5 Specifying the Data File

Click **Sql Loader Data Files** and then click the ellipsis as shown in Figure 3–9 to open the SQL Loader Data Files dialog box.

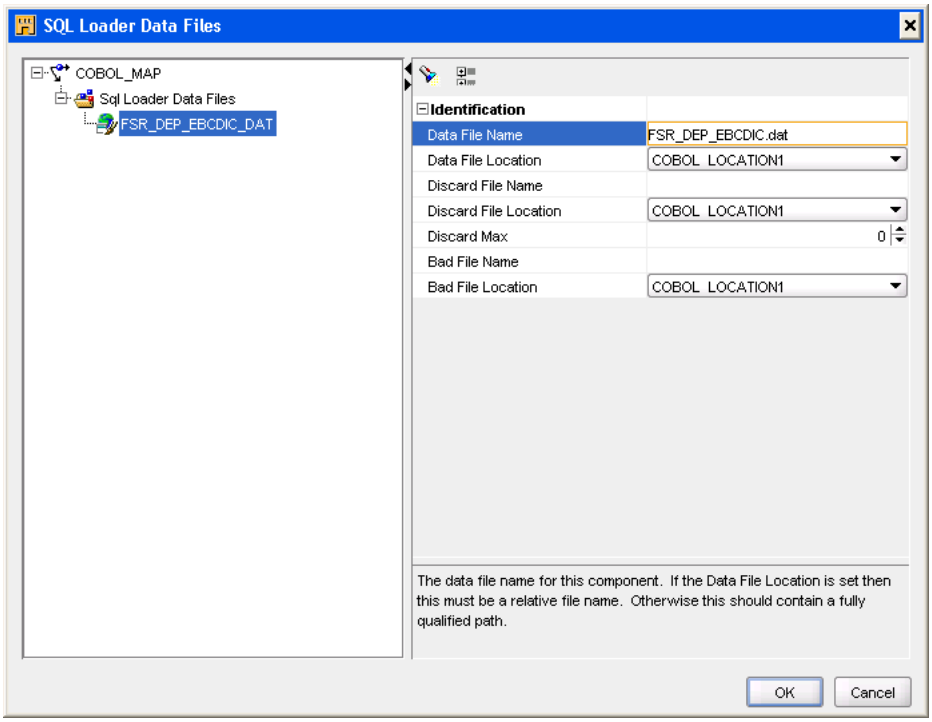
Figure 3–9 Configuration Properties for the Mapping



On the SQL Loader Data Files dialog box, right-click SQL Loader Data Files and select **Create**. A new data file node is added under Sql Loader Data Files.

Provide the data file name and select the data file location from where data is to be loaded. Specify a bad file to store those records that are not loaded into the target table due to error in the data. Specify a discard file to store those records that are not loaded due to SQL*Loader loading checks.

Figure 3–10 SQL Loader Data Files Dialog Box



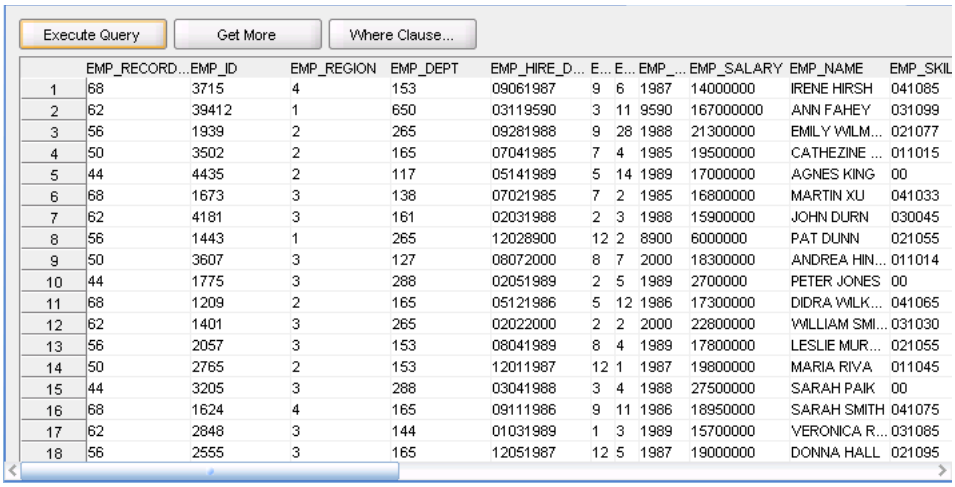
3.4.5.6 Executing the Mapping

After you have defined the configuration properties for the mapping, you can deploy the table and the mapping and then start the mapping to load the COBOL data into the target table.

To deploy the mapping, right-click the mapping and select **Deploy**. After deploying the mapping, start the mapping. Right-click the mapping and select **Start**. When you run the mapping, the data is read from the data file you provided while configuring the mapping, and loaded into the target table.

To view the data in the target table, right-click the table, and select **Data**. Figure 3–11 shows the data in the table.

Figure 3–11 Data in the Target Table



	EMP_RECORD...	EMP_ID	EMP_REGION	EMP_DEPT	EMP_HIRE_D...	E...	E...	EMP_...	EMP_SALARY	EMP_NAME	EMP_SKIL
1	68	3715	4	153	09061987	9	6	1987	14000000	IRENE HIRSH	041085
2	62	39412	1	650	03119590	3	11	9590	167000000	ANN FAHEY	031099
3	56	1939	2	265	09281988	9	28	1988	21300000	EMILY WILM...	021077
4	50	3502	2	165	07041985	7	4	1985	19500000	CATHEZINE ...	011015
5	44	4435	2	117	05141989	5	14	1989	17000000	AGNES KING	00
6	68	1673	3	138	07021985	7	2	1985	16800000	MARTIN XU	041033
7	62	4181	3	161	02031988	2	3	1988	15900000	JOHN DURN	030045
8	56	1443	1	265	12028900	12	2	8900	6000000	PAT DUNN	021055
9	50	3607	3	127	08072000	8	7	2000	18300000	ANDREA HIN...	011014
10	44	1775	3	288	02051989	2	5	1989	2700000	PETER JONES	00
11	68	1209	2	165	05121986	5	12	1986	17300000	DIDRA WILK...	041065
12	62	1401	3	265	02022000	2	2	2000	22800000	WILLIAM SMI...	031030
13	56	2057	3	153	08041989	8	4	1989	17800000	LESLIE MUR...	021055
14	50	2765	2	153	12011987	12	1	1987	19800000	MARIA RIVA	011045
15	44	3205	3	288	03041988	3	4	1988	27500000	SARAH PAIK	00
16	68	1624	4	165	09111986	9	11	1986	18950000	SARAH SMITH	041075
17	62	2848	3	144	01031989	1	3	1989	15700000	VERONICA R...	031085
18	56	2555	3	165	12051987	12	5	1987	19000000	DONNA HALL	021095

3.5 Viewing and Editing a File Definition

You can view and edit the definition of a file by using the Edit Flat File dialog box.

To update a file definition:

1. Select the file definition in the Projects Navigator.
2. Right-click the file and select **Open**.

   Oracle Warehouse Builder displays the Edit Flat File dialog box with the following tabs:

   Name Tab: Use this tab to edit the name and descriptive for the file.

   General Tab: Use this tab to change the general file properties, such as the physical record size, the number of physical records for each logical record, and the delimiter and enclosure characters.

   Record Tab: This tab is available only for flat files with multiple record types. Use this tab to change the record type position or add, delete, or edit record types.

   Structure Tab: Use this tab to edit field level attributes, SQL Loader and SQL Properties.

3.5.1 Name Tab

Use this tab to edit the name, default physical file name, description, and character set for the file. See "Describing a Flat File" on page 3-6 for more details.

3.5.2 General Tab

Use this tab to change the general properties, such as the physical record size, the number of physical records for each logical record, the delimiter and enclosure characters, the number of rows to skip, and the field format. See "Defining File Properties for a Flat File" on page 3-7 for more details about the general properties.

3.5.3 Record Tab

If the file contains multiple record types, then specify the field position that determines the record type in Record Type Location.

Field Position: This field displays the column that contains the record type indicator. You can change this value. For example, if you have a flat file with two record types that are distinguished from each other by the content of the third column as shown in the following list, then the value in this field is 3.

- Record Type 1: `2002 0115 E 4564564`
- Record Type 2: `2003 1231 D 659871 Q HKLIH`

Record type values: This table displays each record type, the value that distinguishes it from the other record types, and the name you have given to the record type. Table 3–2 shows an example of what the record type values for the two sample records earlier might be:

Table 3–2 Example of Record Type Values

Type Value	Record Name
E	Employee
D	Department

- To add new record types, click **New** and enter a Type Value and a Record Name describing the record type.
- To delete record types, select the field to the left of each record type you want to remove and click **Delete**.

For fixed length files containing multiple record types, Record Type Location consists of two fields to determine the record type indicator:

Start Position: The starting position of the field that specifies the record type.

Length: The length of the field.

3.5.4 Structure Tab

Use the Structure tab to edit a field name, data type, mask, "SQL*Loader Properties" and "SQL Properties". You can add or delete a field. You can also add a field mask, NULLIF condition, or DEFAULTIF condition.

If the file contains multiple record types, you can select each record type from the **Record Name** field. Oracle Warehouse Builder displays the field properties for the selected record. See "Defining Field Properties for a Flat File" on page 3-10 for more details.

3.6 Using External Tables

External tables are database objects available in Oracle Database9*i*, and higher.

External tables are tables that represent data from flat files in a relational format. They are read-only tables that act like regular source tables. When you create and define an external table, the metadata for the external table is saved in the workspace. You can load data from flat files to external tables, transform the data using mappings, and load the transformed data to target tables.

The following sections provide information about external tables:

- "Creating a New External Table Definition"
- "Importing an External Table"
- "Synchronizing an External Table Definition with a Record in a File"

- "Editing External Table Definitions"
- "Configuring External Tables"

3.6.1 Creating a New External Table Definition

Before you begin

Each external table you create corresponds to a single record type in an existing flat file. Before you begin, first define the file within the workspace as described in "Defining Character Data Files" on page 3-2.

To create a new external table definition:

1. From the Projects Navigator, expand the **Databases** node and then the **Oracle** node.
2. Expand the module where you want to create the external table.
3. Right-click **External Tables** and select **New External Table.**

   Oracle Warehouse Builder displays the Welcome page of the Create External Table Wizard. Use the wizard to complete the following pages:

   - "Name Page"
   - "File Selection Page"
   - "Locations Page"

3.6.1.1 Name Page

Use the Name page to define a name and an optional description for the external table. Enter the name in the Name field. In the physical naming mode, you must enter a name between 1 and 200 valid characters. Spaces are not allowed in physical mode. In the logical mode, you can enter a unique name up to 4000 characters in length. The external table name must be unique within the module. Spaces are allowed in the logical naming mode.

Use the Description field to enter an optional description for the external table.

3.6.1.2 File Selection Page

The wizard displays the File Selection page. The wizard lists all the flat files available in the workspace. Select a file on which to base the external table. To search through long lists of files, type the first few letters of the file name and click **Go.**

If you select a file that contains multiple record types, you must also select the record type name at the bottom of the File Selection page. An external table can represent only one record type.

You have the option of not specifying the file at this stage. If you do not specify a file in the wizard, you can later specify information such as record type, access parameters, and data files on the external table properties sheet.

3.6.1.3 Locations Page

You can select a location from the list of flat file locations. Alternatively, you can leave the location unspecified. If you do not specify a location in the wizard, you can later specify a location on the external table properties sheet.

Note: The location associated with an external table must be deployed before the external table itself can be deployed.

3.6.2 Importing an External Table

You can create an external table from a flat file, or import an existing external table into Oracle Warehouse Builder.

To import an external table:

1. On the Projects Navigator, right-click **External Tables**, and select **Import**.
2. Specify to import a database object or an Oracle Warehouse Builder metadata file.

3.6.3 Editing External Table Definitions

Use the External Table editor to edit an external table definition. To open the editor, right-click the name of the external table from the Projects Navigator and select **Open Editor**. The Edit External Table dialog box is displayed. The tabs and properties that you can edit depend on how you defined the external table in the workspace.

The External Table Properties window displays with the following tabs:

- "Name Tab"
- "Columns Tab"
- "File Tab"
- "Locations Tab"
- "Data Rules Tab"
- "Access Parameters Tab"

3.6.3.1 Name Tab

Use the Name tab to rename the external table. The same rules for renaming tables apply to external tables. For more information, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

3.6.3.2 Columns Tab

Use the Columns tab to add or edit columns. The same rules for adding columns to tables apply to external tables. For more information, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

3.6.3.3 File Tab

Use the File tab to view the name of the flat file that provides the metadata for the external table. If the source flat file has multiple record types, the File tab also displays the record name associated with the external table. You can update this relationship or change it to a different file and record by reconciling the external table. For more information, see "Synchronizing an External Table Definition with a Record in a File" on page 3-38.

The File tab displays under the following conditions:

- You used the New External Table Wizard to create the external table and you specified a file name.

- You did not specify a file name in the New External Table Wizard, but you reconciled the external table definition with a file and record.

3.6.3.4 Locations Tab

Use the Location tab to view or change the flat file location. The **Location** list displays the available locations. Select a location from this list.

3.6.3.5 Data Rules Tab

Use the Data Rules tab to define data rules for the external table. For more information about using data rules, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

3.6.3.6 Access Parameters Tab

Access parameters define how to read from the flat file when a file has not been specified for the external table. In some cases, the External Table editor displays the Access Parameters tab instead of the File tab.

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

- You imported an external table from another workspace. In this case, you can view and edit the access parameters.
- You created an external table in an Oracle Database and imported its definition. In this case, you can view and edit the access parameters.
- You use the Create External Table Wizard to create an external table and do not specify a reference file. The access parameters are empty. Before generating the external table, you must reconcile the external table definition with a flat file record or manually enter your own access specifications.

The access parameters describe how fields in the source data file are represented in the external table as columns. For example, if the data file contained a field `emp_id` with a data type of INTEGER(2), the access parameters could indicate that the field be converted to a character string column in the external table.

Although you can make changes to the access parameters that affect how the external table is generated and deployed, it is not recommended. Oracle Warehouse Builder does not validate the changes. For more information about external tables and the access parameters, see *Oracle Database Utilities*.

Note: When an external table is imported into Oracle Warehouse Builder, the access parameter definition of the table is truncated to 4000 characters. This can potentially cause DDL generation errors.

3.6.4 Synchronizing an External Table Definition with a Record in a File

Oracle Warehouse Builder enables you to update the external table definition with the metadata changes made to the file associated with the external table. You do this by synchronizing the external table with the source file.

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

1. In the Projects Navigator, right-click the external table to synchronize and select **Synchronize**.

   Oracle Warehouse Builder displays the Synchronize dialog box.

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

   By default, the flat file that was used to create the external table is displayed in this list. Expand the list to see a list of flat file modules and the flat files they contain.

3. Use the **Matching Strategy** list to specify how the search is performed for matches and the external table with the information from the flat file is updated. The options for match strategy are:

   Match By Object ID: This strategy compares the field IDs of that the external table columns references with the field IDs in the flat file.

   Match By Object Name: This strategy compares the physical names in the external table with the physical names in the flat file.

   Match By Object Position: This strategy matches by position, regardless of physical names and IDs. The first external table attribute is reconciled with the first record in the file, the second with the second, and so on. Use this strategy when you want to reconcile the external table with a new record.

4. Use the **Synchronize Strategy** list to indicate how differences in metadata between the existing external table definition and the record you specified are handled:

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

   Replace: Existing record metadata is deleted from the external table definition and the new file record metadata is added to the external table.

5. Click **View Synchronization Plan** to open the Synchronization Plan dialog box.

   You can view the actions performed during synchronization.

6. Select a new strategy and then click **Refresh Plan**.

   On the spread table, expand the Source node to view the action performed on each column.

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

3.6.5 Configuring External Tables

Configure the following properties for an external table:

- "Access Specification"
- "Reject"
- "Data Characteristics"
- "Parallel"
- "Field Editing"
- "Identification"
- "Data Files"

Note: When you import an external table into the workspace and when you manually define access parameters for an external table, some external table configuration properties are overruled by settings on the Access Parameters tab in the External Table Properties window.

To configure the physical properties for an external table:

1. Select an external table from the Projects Navigator.
2. From the Edit menu, select **Configure**. You can also click the **Configure** icon from the toolbar.

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

3.6.5.1 Access Specification

Under **Access Specification**, you can indicate the following file names and locations that Oracle Warehouse Builder uses to load the external table through SQL*Loader.

- **Bad File**: If you specify a name and location for a bad file, Oracle Database is directed to write to that file all records that were not loaded due to errors. For example, records written to the bad file include those not loaded due to a data type error in converting a field into a column in the external table. If you specify a bad file that exists, the existing file is overwritten.
- **Discard File**: If you specify a name and location for a discard file, Oracle Database is directed to write to that file all records that were not loaded based on a SQL*Loader load condition placed on the file. If you specify a discard file that exists, the existing file is overwritten.
- **Log File**: If you specify a name and location for a log file, then Oracle Database is directed to log messages related to the external table to that file. If you specify a log file that exists, new messages are appended.

For each of these files, you can either specify a file name and location, select **Do not use**, or select **Use default location.**

3.6.5.2 Reject

Under **Reject,** you can indicate how many rejected rows to allow. By default, the number of rejected rows allowed is unlimited. If you set **Rejects are unlimited** to false, enter a number in **Number of rejects allowed.**

3.6.5.3 Parallel

Parallel: Enables parallel processing. If you are using a single system, set the value to `NONPARALLEL` to improve performance. If you are using multiple systems, accept the default `PARALLEL`. The access driver attempts to divide data files into chunks that can be processed separately. The following file, record, and data characteristics make it impossible for a file to be processed in parallel:

- Sequential data sources (such as a tape drive or pipe).
- Data in any multibyte character set whose character boundaries cannot be determined starting at an arbitrary byte in the middle of a string. This restriction does not apply to any data file with a fixed number of bytes for each record.
- Records with the VAR format

3.6.5.4 Data Characteristics

If you imported the external table into the workspace or created the external table without specifying a source file, do not configure these properties. Data characteristics properties are overruled by settings on the Access Parameters tab in the External Table Properties window.

Under **Data Characteristics** you can set the following properties:

- **Endian**: The default for the Endian property is Platform. This indicates that it is assumed that the endian of the flat file matches the endian of the platform on which it resides. If the file resides on a Windows platform, the data is handled as little-endian data. If the file resides on Sun Solaris or IBM MVS, the data is handled as big-endian. If you know the endian value for the flat file, you can select big or little-endian. If the file is UTF16 and contains a mark at the beginning of the file indicating the endian, then that endian is used.
- **String Sizes in**: This property indicates how data with multibyte character sets, such as UTF16, is handled. By default, the lengths for character strings in the data file are assumed to be in bytes. You can change the selection to indicate that strings sizes are specified in characters.

3.6.5.5 Field Editing

If you imported the external table into the workspace or created the external table without specifying a source file, do not configure these properties. Field editing properties are overruled by settings on the Access Parameters tab in the External Table Properties window.

Under **Field Editing**, you can indicate the type of whitespace trimming to be performed on character fields in the data file. The default setting is to perform no trim. All other trim options can reduce performance. You can also set the trim option to trim blanks to the left, right, or both sides of a character field.

Another option is to set the trim to perform according to SQL*Loader trim function. If you select SQL*Loader trim, fixed-length files are right trimmed and delimited files specified to have enclosures are left trimmed only when a field is missing an enclosure.

You can indicate how to handle missing fields in a record. If you set the option Trim Missing Values Null to true, fields with missing values are set to NULL. If you set the property to false, fields with missing values are rejected and written into the specified bad file.

3.6.5.6 Identification

See *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for details.

3.6.5.7 Data Files

If a file is associated with an external table, and it has a file name specified, that name is used. The user can configure if a different file must be specified or if multiple files must be specified.

To add a data file:

1. Right-click the **Data Files** node and select **Create.**

   Enter a name for the data file such as `DATAFILE1`. Your entry displays as a new node in the right panel of the Configuration Properties dialog box.

2. Enter the following values for each data file you define:

   Data File Location: Location for the flat file.

   Data File Name: The name of the flat file including its extension. For example, enter `myflatfile.dat`.

4

Connecting to Non-Oracle Data Sources Through Gateways

Gateways provide a transparent connection between Oracle and non-Oracle databases. Once a database is connected through a gateway, you can access it just as you access an Oracle database. As a result, you can use any of the transformation operators available in Oracle Warehouse Builder to transform the data before loading it into a target database.

Using Oracle Database Gateways, you can connect to different non-Oracle databases such as SQL Server, Sybase, Informix, Teradata, and DRDA. DRDA connectivity enables connection to DB2 database. Gateways provide a specific connection agent for each of these databases. For example, for a Sybase data source, the agent is a Sybase-specific gateway. You must install and configure this agent to support communication between the two systems. After creating a gateway connection to the database, you can import metadata into Oracle Warehouse Builder.

Note: Before attempting to import metadata from a non-Oracle database through a gateway, ensure that the specific gateway for that database is installed on the system. See *Oracle Database Heterogeneous Connectivity Administrator's Guide* for more information about Oracle Database Gateways.

See Also: For installation instructions and information about specific gateway agents for different databases, go to

`http://www.oracle.com/pls/db112/gateways`

This chapter describes the use of Oracle Database Gateways for connectivity among databases, from the perspective of Oracle Warehouse Builder. It provides background information on gateway-based connectivity, and then a few examples to illustrate the most common sources and targets. This chapter contains the following topics:

4.1 Connecting to a DB2 Database

To create a gateway connection to a DB2 database, ensure that Oracle Database Gateway for DRDA is installed. Once the gateway connection is established, you can access the DB2 database just as you access an Oracle database.

4.1.1 Creating a DB2 Module

To import metadata from a DB2 database, create a module in the DRDA node. The DRDA node is available under the Databases node in the Projects Navigator. To create a DRDA module:

1. Right-click **DRDA** and select **New DRDA Module**.

   The Create Module Wizard is displayed.

2. In the Name and Description page, provide a name, description (optional), and select a module status.

3. In the Connection Information page, either select an existing location or click **Edit** to open the "Edit Non-Oracle Location Dialog Box" and create a new location.

4. In the Summary page, verify the details and click **Finish**.

The newly created DB2 module is now available under the DRDA node.

4.1.1.1 Edit Non-Oracle Location Dialog Box

Select the **Connection Type** to be one of **Host:Port:Service**, **Database Link**, or **SQL*Net**.

Host:Port:Service

If you selected Host:Port:Service, then provide the following connection details:

- **User Name/Password**: User name and password to access the database.
- **Host**: Computer on which the database is hosted.
- **Port**: SQL port number of the database. The default port number to access DB2 is set to 1111.
- **Service Name**: The service name of the database.
- **Schema**: Browse to select a schema to import.

Database Link

If you selected Database Link, then provide the following connection details:

- **From Location**: An existing location where the database link is defined
- **Database Link Name**: The object name of the database link
- **Schema**: The schema where the source data is stored or the target objects is deployed.

SQL*Net

For SQL*Net, provide the following connection details:

- **User Name**: The database user credential that has permission to access the schema location.

  When connecting to a database that does not have user names, enter any text as a mock user name.

- **Password**: The password associated with user name.

  When connecting to a database that does not require a password, enter any text as a mock password.

- **Net Service Name**: The name of the predefined connection.
- **Use Global Name**: The unique name of the database, which is composed of the database name and the domain in the form *database_name.database_domain*. For example, `db2.us.example.com` identifies the `db2` database in the `us.example.com` domain. Select this option when connecting to a database in a different network.

Click **Test Connection** to verify that you have provided the correct details and click **OK** to set the connection details.

4.2 Connecting to a SQL Server Database

To create a gateway connection to a SQL Server database, ensure that Oracle Database Gateway for SQL Server is installed.

4.2.1 Creating a SQL Server Module

The SQL Server node is available under the Databases node in the Projects Navigator. To create a SQL Server module:

1. Right-click **SQL Server** and select **New SQL Server Module**.

   The Create Module Wizard is displayed.

2. In the Name and Description page, provide a name, description (optional), and select a module status. Select **SQL Server Gateway** as the access method.
3. In the Connection Information page, either select an existing location or click **Edit** to open the "Edit Non-Oracle Location Dialog Box" and create a new location.
4. In the Summary page, verify the details and click **Finish**.

The newly created SQL Server module is now available under the SQL Server node.

4.2.1.1 Edit Non-Oracle Location Dialog Box

Select the **Connection Type** to be one of **Host:Port:Service**, **Database Link**, or **SQL*Net**.

Host:Port:Service

If you selected Host:Port:Service, then provide the following connection details:

- **User Name/Password**: User name and password to access the database.
- **Host**: Computer on which the database is hosted.
- **Port**: SQL port number of the database. The default port number to access SQL Server is set to 1526.
- **Service Name**: The service name of the database.
- **Schema**: Browse to select a schema to import.

Database Link

If you selected Database Link, then provide the following connection details:

- **From Location**: An existing location where the database link is defined
- **Database Link Name**: The object name of the database link

- **Schema**: The schema where the source data is stored or the target objects is deployed.

SQL*Net

For SQL*Net, provide the following connection details:

- **User Name**: The database user credential that has permission to access the schema location.

  When connecting to a database that does not have user names, enter any text as a mock user name.

- **Password**: The password associated with user name.

  When connecting to a database that does not require password, enter any text as a mock password.

- **Net Service Name**: The name of the predefined connection.

- **Use Global Name**: The unique name of the database, which is composed of the database name and the domain in the form *database_name.database_domain*. For example, `sqlserver.us.example.com` identifies the `SQL Server` database in the `us.example.com` domain. Select this option when connecting to a database in a different network.

Click **Test Connection** to verify that you have provided the correct details and click **OK** to set the connection details.

4.3 Connecting to an ODBC Data Source

Oracle Warehouse Builder can access any data source that supports open database connectivity (ODBC). This is possible using the ODBC support provided by Oracle Database. You can use ODBC access for any data source not supported by a specific gateway agent.

4.3.1 Setting up ODBC Connectivity

While ODBC connectivity is widely available, it is most commonly used to connect to data sources on Windows platforms. The Oracle Database configuration process should be substantially similar for connecting to any ODBC source. ODBC setup on the source system can vary greatly in different environments. The following discussion describes the common case of setting up ODBC connectivity to a source on a Windows platform using a System DSN. If you are on a non-Windows platform, consult the documentation from your vendor on ODBC setup.

4.3.1.1 Installing ODBC Driver

To enable ODBC connectivity to a data source, you must first install the ODBC driver for that particular data source.

4.3.1.2 Creating a System DSN

After installing the driver, create a system DSN using the Microsoft ODBC Administrator.

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

   This opens the ODBC Data Source Administrator dialog box.

2. Navigate to the System DSN tab and click **Add** to open the Create New Data Source dialog box.
3. Select the driver for which you want to set up the data source.

   Click **Finish** to open the ODBC Setup dialog box.
4. Specify a name for the data source, and other relevant connection details for the data source. The connection details that you provide here differs for each type of data source.

4.3.1.3 Configuring the Database

Next, you must configure Oracle Database to connect to the data source. Oracle Warehouse Builder can then use this configuration to extract metadata from the data source. This involves:

- "Creating a Heterogeneous Services Initialization File"
- "Configuring the listener.ora File"

4.3.1.3.1 Creating a Heterogeneous Services Initialization File To configure an ODBC connection agent, you must create a heterogeneous services initialization file for the agent. Create this file in the *OWB_HOME*\hs\admin directory of the database.

The file name should start with `init` and have the extension `.ora`. It should be of the format `initSID.ora`, where *SID* is the system identifier for the ODBC agent. The value to be entered in this file is:

```
HS_FDS_CONNECT_INFO = DSN_NAME
```

DSN_NAME is the data source name that you provide while creating the system DSN. Other relevant data that must be provided in this file depends on the data source you are connecting to.

4.3.1.3.2 Configuring the listener.ora File Add a new *SID* description in the `listener.ora` file. This file is available in the *OWB_HOME*\network\admin directory.

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = SID)
      (OWB_HOME = owbhome)
      (PROGRAM = program)
    )
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (OWB_HOME = owbhome)
      (PROGRAM = program)
      )
  )
```

The *SID_NAME* parameter must contain the name of the configuration file you created in the previous step. However, it must not contain the init prefix. For example, if the configuration file you created in the previous step was `initdb2.ora`, then the value of the *SID_NAME* parameter should be `db2`.

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

The value associated with the `PROGRAM` keyword defines the name of the executable agent, and differs for each type of data source.

Restart the listener service after making these modifications.

The steps involved in setting up the ODBC connectivity are similar for all data sources. However, the data to be provided while "Creating a System DSN", "Creating a Heterogeneous Services Initialization File", and "Configuring the listener.ora File" differs for each data source.

See Chapter 5, "Connecting to Microsoft Data Sources Through ODBC Connection" for specific examples of connecting to an Excel worksheet and SQL Server database.

> **See Also:** For more information about heterogeneous connectivity using Oracle Gateways, see *Oracle Database Heterogeneous Connectivity Administrator's Guide*.

4.3.2 Creating an ODBC Module

After you set up the ODBC connectivity to a data source, you must create an ODBC module to import the metadata from this data source. The ODBC node is available under the Databases node in the Projects Navigator. To create an ODBC module:

1. Right-click **ODBC** and select **New ODBC Module**.

   The Create Module Wizard is displayed.

2. In the Name and Description page, provide a name, description (optional), and select a module status.

3. In the Connection Information page, either select an existing location or click **Edit** to open the "Edit Non-Oracle Location Dialog Box" and provide the connection details.

4. In the Summary page, verify the details and click **Finish**.

The newly created ODBC module is now available under the ODBC node.

4.3.2.1 Edit Non-Oracle Location Dialog Box

Select the **Connection Type** to be one of **Host:Port:Service**, **Database Link**, or **SQL*Net**.

Host:Port:Service

If you selected Host:Port:Service, then provide the following connection details:

- **User Name/Password**: You can provide a dummy user name and password as you are not connecting to an Oracle database.
- **Host**: Computer on which the database is hosted.
- **Port**: SQL port number of the database.
- **Service Name**: The *SID_NAME* that you specify in the `listener.ora` file.
- **Schema**: You can leave this field empty if you are not importing from a schema.

Database Link

If you selected Database Link, then provide the following connection details:

- **From Location**: An existing location where the database link is defined
- **Database Link Name**: The object name of the database link
- **Schema**: The schema where the source data is stored or the target objects is deployed.

SQL*Net

For SQL*Net, provide the following connection details:

- **User Name**: The database user credential that has permission to access the schema location.

  When connecting to a database that does not have user names, enter any text as a mock user name.

- **Password**: The password associated with user name.

  When connecting to a database that does not require password, enter any text as a mock password.

- **Net Service Name**: The name of the predefined connection.

- **Use Global Name**: The unique name of the database, which is composed of the database name and the domain in the form *database_name.database_domain*. Select this option when connecting to a database in a different network.

Click **Test Connection** to verify that you have provided the correct details and click **OK** to set the connection details.

4.4 Importing Database Objects

After establishing connection to a DB2, SQL Server, or any other generic ODBC database, you can import metadata from the data objects in the database. The database objects that you can import are tables, views, and sequences. To import database objects:

1. Right-click the newly created module, and select **Import**, **Database Object**.

   The Import Metadata Wizard is displayed.

2. In the Filter Information page, select the object types to be imported. You can also narrow down the selection of objects by specifying a string pattern.

3. In the Object Selected page, move the required objects from the Available list to the Selected list. Use the **Find Objects** button to search for objects in the Available list. Also specify whether dependent objects must be imported.

4. In the Summary and Import page, verify the objects to be imported. Click **Advanced Import Options** to open the Advanced Import Options dialog box. By default, the following options are selected:

   - Preserve workspace added column
   - Preserve workspace added constraints
   - Import descriptions

   Retain or deselect any of the options.

You can now use any of the imported objects in a mapping to load data.

5

Connecting to Microsoft Data Sources Through ODBC Connection

This chapter provides examples of using ODBC connectivity to connect to Microsoft Excel and Microsoft SQL Server as sources. It contains the following topics:

- "Connecting to Excel Spreadsheets Through ODBC"
- "Connecting to SQL Server Database Through ODBC"

5.1 Connecting to Excel Spreadsheets Through ODBC

Scenario

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

Solution

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

5.1.1 Case Study

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

Step 1: Install ODBC Driver for Excel

To read data from Microsoft Excel, you need the ODBC driver for Excel. By default, the ODBC driver for Excel is installed on a Windows system.

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

If you want to delimit the data to be imported from the Excel file, then define a name for the range of data being sourced:

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

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

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

Step 3: Create a System DSN

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

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

   This opens the ODBC Data Source Administrator dialog box.

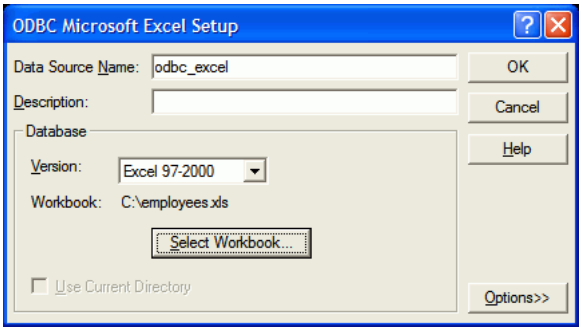
2. Navigate to the System DSN tab and click **Add** to open the Create New Data Source dialog box.

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

   Click **Finish** to open the ODBC Microsoft Excel Setup dialog box.

   The ODBC Microsoft Setup dialog box is shown in Figure 5–1.

Figure 5–1 ODBC Microsoft Excel Setup Dialog Box



4. Specify a name for the data source. For example, `odbc_excel`.

5. Click **Select Workbook** to select the Excel file from which you want to extract data.

6. Verify that the Version field lists the correct version of the source Excel file.

Step 4: Create the Heterogeneous Services Initialization File

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

Create the `initexcelsid.ora` file in the *`OWB_HOME`*`\hs\admin` directory as follows:

```
HS_FDS_CONNECT_INFO = odbc_excel
HS_AUTOREGISTER = TRUE
HS_DB_NAME = dg4odbc
```

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

Step 5: Modify the listener.ora file

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

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = excelsid)
      (OWB_HOME = C:\oracle11g\product\11.2.0\db_1)
      (PROGRAM = dg4odbc)
    )
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (OWB_HOME = C:\oracle11g\product\11.2.0\db_1)
      (PROGRAM = extproc)
      )
  )
```

1. For the *SID_NAME* parameter, use the *SID* that you specified when creating the initialization parameter file for the Heterogeneous Services, which, in this case, is `excelsid`..
2. Ensure that the *OWB_HOME* parameter value is the path to your Oracle Database home directory.
3. The value associated with the `PROGRAM` keyword defines the name of the agent executable.

Remember to restart the listener after making these modifications.

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

Step 6: Create an ODBC Source Module

Use the following steps to create an ODBC source module:

1. From the Projects Navigator, create an ODBC source module.

   ODBC is listed under the Databases node. See "Creating an ODBC Module" on page 4-6.

2. To provide connection information, on the Connection Information page, click **Edit** to open the Edit Non-Oracle Location dialog box and provide the following details:

   Ensure that the service name you provide equals the *SID_NAME* you specified in the `listener.ora` file.

   Enter the host name and the port number in the **Host** and **Port** fields respectively.

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

   The **Schema** field can be left empty because you are not importing metadata from a schema.

   Click **Test Connection** to verify the connection details.

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

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

1. Select **employee_details** and use the right arrow to move it to the list of selected objects.
2. Click **Finish** to import the metadata.

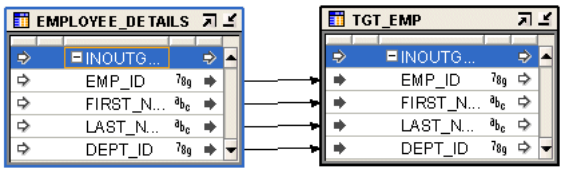
   The data from the employee_details worksheet is now stored in a table called `employee_details` in the ODBC source module.

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

Create a mapping in the module that contains the target table. Use the `employee_details` table imported in the previous step as the source and map it to the target table.

Figure 5–2 displays the mapping used to load data into the target table.

Figure 5–2 Mapping to Load Data Into the Target Table



Step 9: Deploy the Mapping

Use the Control Center Manager or Design Center to deploy the mapping you created in step 8. Ensure that you first deploy the source module before you deploy the mapping. See *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for more information about mappings.

5.1.2 Troubleshooting

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

Error

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

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

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

Probable Cause

Providing the same SID name as that of your database.

Action

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

Error

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

Probable Cause

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

Action

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

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

5.2 Connecting to SQL Server Database Through ODBC

Scenario

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

Solution

One of the ways to connect to an SQL Server database from Oracle Warehouse Builder is to use an ODBC gateway. Once connected, you can import metadata and load data.

Case Study

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

1. "Creating an ODBC Data Source" on page 5-5
2. "Configuring the Oracle Database Server" on page 5-6
3. "Adding the SQL Server as a Source in Oracle Warehouse Builder" on page 5-7

If you encounter problems implementing this solution, see "Troubleshooting" on page 5-7.

5.2.1 Creating an ODBC Data Source

You must create an ODBC data source to connect to the SQL Server database using ODBC. You must set up a System Data Source Name (DSN):

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

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

8. Select the database file and click **Next**.
9. Accept the default values in the next screen and click **Finish**.
10. Test the data source to verify the connection.

5.2.2 Configuring the Oracle Database Server

Next, you must configure the Oracle database to connect to the SQL Server database. Oracle Warehouse Builder can then use this configuration to extract metadata from the SQL Server database. This involves the following steps:

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

5.2.2.1 Creating a Heterogeneous Service Configuration File

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

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

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

Enter the following in the file:

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

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

5.2.2.2 Editing the listener.ora file

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

Modify the file as shown:

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = sqlserver)
      (OWB_HOME = c:\oracle10g\owb_home)
      (PROGRAM = dg4odbc)
    )
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (OWB_HOME = c:\oracle10g\owb_home)
      (PROGRAM = extproc)
      )
  )
```

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

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

The value associated with the PROGRAM keyword defines the name of the executable agent, which, in this case, is dg4odbc.

Restart the listener service after making these modifications.

5.2.3 Adding the SQL Server as a Source in Oracle Warehouse Builder

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

To add an ODBC source module in Oracle Warehouse Builder:

1. Within a project in the Projects Navigator, navigate to the Databases node.
2. Right-click **ODBC** and select **New ODBC Module**.
3. Create a new ODBC module using the Create Module Wizard.
4. Use the Connection Information page to provide the location details. To create a new location, click **Edit** to open the Edit Non-Oracle Location dialog box.
5. In the Edit Location dialog box, ensure that you enter user name and password within double quotation marks ("). For example, if the user name is matt, then enter "matt".
6. For Service Name, enter the SID name you provided in the listener.ora file. Also select the schema from which you want to import the metadata.
7. Click **Test Connection** to verify the connection details.

To import metadata into the ODBC module:

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

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

5.2.4 Troubleshooting

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

Error

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

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

Probable Cause

Creating the DSN from the User DSN tab.

Action

Create the DSN from the System DSN tab.

Error

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

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

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

Probable Cause

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

Action

Enter the user name and password within double quotation marks.

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

6

Connecting to Data Sources Through JDBC

You can connect to a wide variety of non-Oracle databases and import metadata and data from these sources using JDBC connectivity.

This chapter provides connection details for a non-Oracle data sources that can be accessed through JDBC, and generic connection information. It contains the following topics:

- "Connecting to DB2 Database"
- "Connecting to SQL Server Database"

JDBC connectivity is used with code templates-based mappings. See *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for more details about these mappings.

6.1 Generic Connection Using JDBC

You can connect to any data source that supports JDBC connectivity. To connect to the data source, you require the JDBC driver for that data source and the URL format to set up the connection. For any database, download the required JDBC driver into *OWB_HOME*/owb/lib/ext.

6.2 Connecting to DB2 Database

You can connect to a DB2 database using a JDBC connection. Ensure that the following prerequisites are met before connecting to a DB2 database.

JDBC Connection Drivers for DB2

You must download the following jar files into *OWB_HOME*/owb/lib/ext on your client system:

- db2jcc.jar
- db2jcc_license_cu.jar

> **See Also:** *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for more information about using code template mappings.

6.2.1 Creating a DB2 Module

Before you can import metadata from a DB2 database, you must create a DB2 module to store the metadata.

To create a DB2 module:

1. Right-click **DB2** under the Databases node in the Projects Navigator and select **New DB2 Module**.

   The Create Module Wizard is displayed.

2. Click **Next** to open the Name and Description page.

   Provide a name, description (optional), and the access method. Select the access method **Native Database Connection**, which implies using a JDBC driver to make the connection.

3. Click **Next** to open the Connection Information page.

   You can select from an existing location or provide new location details.

4. To provide a new location, click **Edit** on the Connection Information page to open the Edit DB2 Location dialog box.

   Provide the following details in the Edit DB2 Location dialog box:

   - **User Name**: The user name to connect to the host system.
   - **Password**: The password for the user name.
   - **Host**: The host system where the database resides.
   - **Port**: The port number is usually 50000 for the DB2 connection.
   - **Database**: The database name.
   - **Schema**: The schema from which objects are to be imported.
   - **Version**: The database version.

5. Click **Test Connection** to ensure that the connection is valid.
6. Click **OK** to return to the Connection Information page.
7. Click **Next**, and then click **Finish** in the Summary page after verifying the specified details.

The newly created module is available under the DB2 node in the Projects Navigator. A DB2 module supports the following data objects:

- Transformations
- Tables
- Views
- Sequences

> **Note:** Temporary tables can be created in the work schema when code template mappings run. To change the work schema, edit the location and use the Advanced tab.

6.2.2 Importing Metadata into a DB2 Module

To import metadata into a module:

1. Right-click the DB2 module and select **Import**, then **Database Object**.

   The Import Metadata Wizard is displayed.

2. In the Filter Information page, select the object types to be imported.

3. In the Object Selection page, select the objects to be imported. Also, specify whether dependent objects should be selected.
4. In the Summary page, verify the objects you selected.
5. Click **Finish** to begin the import.
6. In the Import Results dialog box, click **OK** to confirm the import action. Click **Undo** to cancel the import.

6.3 Connecting to SQL Server Database

You can connect to a SQL Server database using a JDBC connection.

JDBC Connection Driver for SQL Server

To connect using JDBC, you must place the jar file `sqljdbc.jar` into *`OWB_HOME`*`/owb/lib/ext` on your client system.

This jar file is available for download at the Microsoft download center.

6.3.1 Creating a SQL Server Module

Before you can import metadata from a SQL Server database, you must create a SQL Server module to store the metadata.

To create a SQL Server Module:

1. Right-click **SQL Server** under the Databases node in the Projects Navigator and select **New SQL Server Module**.

   The Create Module Wizard is displayed.
2. Click **Next** to open the Name and Description page.

   Provide a name, description (optional), and the access method. Select the access method **Native Database Connection**, which implies using a JDBC driver to make the connection.
3. Click **Next** to open the Connection Information page.

   You can select from an existing location or provide new location details.
4. To provide a new location, click **Edit** on the Connection Information page to open the Edit SQL Server Location dialog box.

   Provide the following details in the Edit SQL Server Location dialog box:

   - **User Name**: The user name to connect to the host system.
   - **Password**: The password for the user name.
   - **Host**: The host system where the database resides.
   - **Port**: The port number is usually 1433 for the SQL Server connection.
   - **Database**: The database name.
   - **Schema**: The schema from which objects are to be imported.
   - **Version**: The database version.
5. Click **Test Connection** to ensure that the connection is set.
6. Click **OK** to return to the Connection Information page.

7. Click **Next**, and then click **Finish** in the Summary page after verifying the specified details.

The newly created module is available under the SQL Server node in the Projects Navigator. A SQL Server module supports the following data objects:

- Transformations
- Tables
- Views
- Sequences

6.3.2 Importing Metadata into a SQL Server Module

To import metadata into a module:

1. Right-click the SQL Server module and select **Import**, then **Database Object**.

   The Import Metadata Wizard is displayed.

2. In the Filter Information page, select the object types to be imported.
3. In the Object Selection page, select the objects to be imported. Also specify whether dependent objects should be selected.
4. In the Summary page, verify the objects you selected.
5. Click **Finish** to begin the import.
6. In the Import Results dialog box, click **OK** to confirm the import action. Click **Undo** to cancel the import.

After you create a JDBC connection and import metadata from data objects, you can extract data from these objects and load it to target data sources by performing extraction, transformation, and loading (ETL) operations. However, to perform ETL operations on JDBC connected-data objects, you can use only code template mappings. A limited set of operators are supported for code template mappings. All operators are supported for Oracle target in code template mappings. For more information about code templates, see Chapter 12, "Using Code Templates to Load and Transfer Data". For more information about ETL operations and different types of mappings including PL/SQL mappings and code template mappings, see the *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

6.4 Importing Metadata from Other Databases Using JDBC

To import metadata definitions from other databases using JDBC drivers, you must create a new platform in Oracle Warehouse Builder. A platform refers to a data source. See Chapter 11, "Creating New Platforms" for more information about platforms.

7

Extracting Data from SAP Applications

Many companies implement the SAP Enterprise Resource Planning (ERP) system and Oracle Warehouse Builder enables easy access to the data in these SAP systems.

This chapter describes how you can extract data from SAP systems. It describes why you require an SAP connector, how to import metadata from SAP tables, use them in a mapping, generate ABAP code for the mappings, and deploy them to an SAP system. The chapter also describes the various methods by which you can extract data from the SAP system and load this data into a target table on Oracle Warehouse Builder system.

This chapter contains the following topics:

7.1 Why SAP Connector

The SAP R/3 system operates differently compared to SQL-based systems like Oracle E-Business Suite and Oracle's PeopleSoft systems.

The major differences include:

- The native data manipulation language is ABAP, which is a proprietary SAP language.
- Table names are cryptic compared to those in SQL-based ERP systems.
- In addition to database tables, SAP contains logical tables called *pool tables* and *cluster tables*. These tables contain multiple physical tables and must be managed differently from SQL-based tables.

The SAP connector assists you in managing all these issues. Furthermore, the SAP connector enables you to follow the administrative and security processes of the SAP environment.

7.2 Supported SAP Versions

For information about the SAP R/3 versions supported by Oracle Warehouse Builder 11*g*, log in to `https://support.oracle.com/`, and go to the **Certify** link.

7.3 Overview of SAP Objects

This section provides a brief overview of the different types of tables in SAP, and how data is organized within an SAP system. The section consists of the following topics:

7.3.1 SAP Object Types

With the SAP connector, you can import metadata definitions for the following SAP table types:

- **Transparent**: A transparent table is a database table that stores data. You can access the table from non-SAP systems as well, for example, using SQL statements. However, Oracle Warehouse Builder uses ABAP code to access transparent tables.
- **Cluster**: A cluster table is usually used to store control data. It can also be used to store temporary data or documentation. Because cluster tables are data dictionary tables and not database tables, you can access these tables only by using ABAP.
- **Pooled**: This is a logical table that must be assigned to a table pool in the database. A table pool consists of multiple pooled tables. A pooled table is used to store control data such as program parameters. You require ABAP code to access pooled tables.

7.3.2 SAP Business Domains

SAP application systems logically group tables under different business domains. In SAP, a business domain is an organizational unit in an enterprise that groups product and market areas. For example, the Financial Accounting business domain represents data describing financial accounting transactions. These transactions might include General Ledger Accounting, Accounts Payable, Accounts Receivable, and Closing and Reporting.

When you import SAP definitions, you can use a graphical navigation tree in the Business Domain Hierarchy dialog box to search the business domain structure in the SAP source application. This navigation tree enables you to select SAP tables from the SAP application server.

7.4 Overview of the Interaction Between Oracle Warehouse Builder and SAP

Moving data from the SAP system to an Oracle database using Oracle Warehouse Builder consists of the following tasks:

1. Connecting to the SAP system.
2. Importing metadata from SAP data objects.
3. Creating an extraction mapping in Oracle Warehouse Builder that defines:
   - The SAP source tables from which data is to be imported.

- The transformation operators that operate on the source tables to extract data based on certain criteria.
- The target table in Oracle Warehouse Builder to store the data imported from the SAP source tables.

4. Deploying the mapping.

   This creates the ABAP report for the mapping.

5. Starting the mapping.

   This results in the following sequence of tasks, all of which are performed automatically by Oracle Warehouse Builder:

   - Transfer of the ABAP report to the SAP server.
   - Compiling of the ABAP report.
   - Execution of the ABAP report, which results in the generation of a data file (this file has a `.dat` extension).
   - Transfer of the data file to the Oracle Warehouse Builder server using FTP.
   - Loading data from the data file into the target table in Oracle Warehouse Builder. The loading takes place using SQL*Loader.

7.4.1 SAP Function Modules

To access SAP data from non-SAP systems, you typically use a function module to execute an ABAP program that extracts the data. A function module in SAP is a procedure that is defined in a special ABAP program known as a function group. After the function group is defined, the function module can then be called from any ABAP program.

SAP contains a predefined function module called `RFC_ABAP_INSTALL_AND_RUN` to execute ABAP report. To upload the Oracle Warehouse Builder-generated ABAP report and execute it in SAP, you need access rights to this function module.

Alternatively, you can ask the SAP administrator to create a customized function module that executes a specific ABAP program. You can then use this function module to execute the ABAP report generated by Oracle Warehouse Builder.

7.4.2 Data Retrieval Mechanisms

Data retrieval from the SAP system can be "Completely Managed by Oracle Warehouse Builder", "Managed by Oracle Warehouse Builder with SAP Verification", or "Manual". This depends on whether the SAP administrator provides the Oracle Warehouse Builder user with access rights to the predefined function module `RFC_ABAP_INSTALL_AND_RUN` or creates a customized function module to execute the ABAP report.

Completely Managed by Oracle Warehouse Builder

In this mechanism, Oracle Warehouse Builder has access to upload and execute the generated ABAP using the default function module `RFC_ABAP_INSTALL_AND_RUN`, and to use FTP to import the generated data file from the SAP system.

Thus the entire process of retrieving data from the SAP system and creating a target table is managed by the Oracle Warehouse Builder and can be completely automated. It is therefore the simplest method of data retrieval. See "Automated System" on page 7-17 for more details on implementing this data retrieval mechanism.

Managed by Oracle Warehouse Builder with SAP Verification

In this mechanism, as an Oracle Warehouse Builder user, you do not have access rights to the default function module `RFC_ABAP_INSTALL_AND_RUN` that executes the ABAP report in the SAP system. Instead, the SAP administrator first verifies the ABAP report that you generate using Oracle Warehouse Builder, and then creates a customized function module to execute this ABAP report. You can then run the ABAP code on the SAP system using this customized function module.

See "Semi-Automated System" on page 7-19 for more details about implementing this data retrieval mechanism.

Manual

In this mechanism, as a Oracle Warehouse Builder user, you cannot directly run the ABAP code on the SAP system. Instead, you generate the ABAP report for the mapping, and send it to the SAP administrator, who runs the code on the SAP system. You then import the generated data file using FTP and load the target table.

The tasks involved in retrieving data using FTP and creating the Oracle table are implemented using a Process Flow. See "Manual System" on page 7-20 for more details about implementing this system.

7.5 Implementing the SAP Data Retrieval Mechanism

As a Oracle Warehouse Builder user, you must be aware of certain restrictions while trying to extract data from an SAP system.

Because the SAP and Oracle Warehouse Builder systems are totally independent systems, as an Oracle Warehouse Builder user, you may only have restricted access rights to the SAP data (especially in the production environment). You therefore have to interact with the SAP administrator to extract data from the system.

Access privilege to the SAP system is most often determined by whether it is the development, test, or the production environment. Each of the data retrieval mechanisms can be implemented in the development, test, or production environment depending on the privileges granted by the SAP system administrator.

Development Environment

Typically, in the development environment, the SAP administrator gives you access rights to use the predefined function module `RFC_ABAP_INSTALL_AND_RUN`. Therefore, in this environment, you can implement a completely "Automated System" for data retrieval.

Test and Production Environment

Typically, in the test and production environments, the SAP administrator may not give you access rights to use the predefined function module `RFC_ABAP_INSTALL_AND_RUN`. Instead, the SAP administrator verifies the ABAP report, and either creates a customized function module that you can use, or runs the ABAP report on the SAP system, and enables you to extract the resultant data. You can therefore implement either a "Semi-Automated System" or a "Manual System" for data retrieval.

A typical data retrieval system may therefore consist of any of the three mechanisms implemented in the different environments.

Scenario 1

You run the automated system in the SAP development environment. After you verify the ABAP report in this environment, you then move the ABAP report to the SAP test

environment and test the code using a customized function module. You then finally move this to the SAP production environment.

This implementation is recommended by Oracle, as it automates and simplifies the data retrieval task.

Scenario 2

Depending on the access rights to the development, test, and production environments, you implement any one of the data retrieval mechanisms in each of the environments.

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

1. "Connecting to the SAP System" on page 7-5
2. "Importing Metadata from SAP Tables" on page 7-10
3. "Creating SAP Extraction Mappings" on page 7-13
4. "Deploying and Executing SAP ABAP Mappings" on page 7-17.

7.6 Connecting to the SAP System

To connect to the SAP system from Oracle Warehouse Builder, you must use certain SAP-specific DLL files. After you establish connection, you can then import metadata from SAP tables into SAP modules in Oracle Warehouse Builder.

This section contains the following topics:

- "Required Files for SAP Connector"
- "Creating SAP Module Definitions"
- "Troubleshooting Connection Errors"
- "Creating SAP Module Definitions"

7.6.1 Required Files for SAP Connector

Different sets of files are required depending on whether you are working on a Windows or a UNIX system.

Files Required in Windows

The SAP connector requires a DLL file named `librfc32.dll` to use remote function calls on the client computer. You must copy `librfc32.dll` to the location specified in `java.library.path` on your client system.

To find this location, click **MyComputer**, **Properties**, and then click **Advanced**. Next click **Environment Variables**, and under System variables, check the locations specified for the variable *Path*.

You can copy the `librfc32.dll` file to any one of the multiple locations specified in Path. One of the locations correspond to *`OWB_HOME`*, and is therefore the preferred location. This location is usually *`OWB_HOME`*`\owb\bin`.

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

Table 7–1 Required Files for Windows

Required Files	Path	Description
librfc32.dll	*OWB_HOME*\owb\bin	This file is available on the SAP Application Installation CD.
sapjcorfc.dll	*OWB_HOME*\owb\bin	Copy this file to the same location where you placed librfc32.dll.
sapjco.jar	*OWB_HOME*\owb\lib\int	

Restart the client after copying these files.

Files Required in UNIX

The SAP connector requires a DLL file named librfccm.so to use remote function calls on the client computer. You must copy this file to the location specified by the UNIX environment variable path LD_LIBRARY_PATH on your client system.

By default, *OWB_HOME*/owb/bin/admin is the location specified in LD_LIBRARY_PATH. If it is not, then ensure that you add *OWB_HOME*\owb\bin\admin to LD_LIBRARY_PATH.

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

Table 7–2 Required Files for UNIX

Required Files	Path	Description
librfcccm.so	*OWB_HOME*\owb\bin\admin	This file is available on the SAP Application Installation CD.
libsapjcorfc.so	*OWB_HOME*\owb\bin\admin	Copy this file to the same location where you placed librfcccm.so.
sapjco.jar	*OWB_HOME*\owb\lib\int	

Restart the client after copying these files.

> **Note:** Different versions of SAP R/3 might require different versions of the DLL, SO, and JAR files. The correct versions are available in the SAP installation CD. The files can also be downloaded from:
>
> http://service.sap.com/patches

7.6.2 Troubleshooting Connection Errors

The most common errors while connecting to the SAP system are listed in Table 7–3:

Table 7–3 SAP Connection Errors

Error Message	Possible Reason
Connection failed. You are not authorized to logon to the target system (error code 1).	Incorrect user name or password to connect to the SAP server.
Connection failed. Connect to SAP gateway failed.	Incorrect application server, system number, or client details.

Table 7–3 (Cont.) SAP Connection Errors

Error Message	Possible Reason
Some Location Details are missing. Please verify the location information is completely specified.	Missing DLL files, or DLL files placed in the wrong location.
Missing `saprfc32.dll`	Missing `saprfc32.dll` file, or file placed in the wrong location.

Note: If you create the SAP source module and import SAP tables but cannot see the columns in the tables, then you have an incompatible `librfc32.dll` file. Download the correct version of the DLL file from the SAP Web site.

7.6.3 Creating SAP Module Definitions

Use the Create Module Wizard to create the SAP source module that stores data from the SAP source.

To create a SAP Module:

1. Right-click **SAP** and select **New SAP**.

   The Create Module Wizard is displayed.

2. On the Name and Description page, provide a name for the SAP module. Select the module status and optionally also provide a description. Click **Next**.

3. On the Connection Information page, either select from an existing location or click **Edit** to open the Edit SAP Location dialog box. Specify the details as described in "Connecting to the SAP System" on page 7-8. Click **Next**.

4. On the Summary page, click **Finish**.

   A new sap module is now available on the Projects Navigator.

Note: Before you create a SAP location, ensure that you have all the necessary information. You can provide the location information either while creating the module or before importing metadata into the module. You require the following information to create the location: server name, user name, password, system number, and client number. Obtain these details from the system administrator.

When you set the connection information, you can choose the connection types described in the subsequent sections.

Remote Function Call (RFC)

A remote function call enables you to call a function module on a remote system. This method requires specific IP Address information for the SAP application server.

SAP Remote Function Call (SAPRFC.INI)

You can also specify the connection information in a file called `SAPRFC.INI`, and copy this file to the following location: *`OWB_HOME`*`\owb\bin\admin`.

Using the SAPRFC.INI file requires prior knowledge of ABAP parameters, because you must specify the values for certain parameters to make the SAP connection, and is not the recommended connection method if you are not familiar with ABAP.

Note: The SAPRFC.INI file comes with the SAP installation CD.

The Create Module Wizard creates the module for you based on the metadata contained in the SAP application server.

7.6.3.1 Connecting to the SAP System

1. Select one of the following connection types:
   - Remote Function Call (RFC)

     This is the recommended connection type, and is selected by default in Oracle Warehouse Builder.
   - SAP Remote Function Call (SAPRFC.INI)

   For more information about these connection types, see "Creating SAP Module Definitions" on page 7-7.
2. Enter the connection information in the appropriate fields. The fields displayed on this page depend on the connection type you choose.

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

   You must obtain the connection information for your SAP application server from the system administrator before you can complete this step.

   RFC Connection type requires the following connection information:

   Application Server: The alias name or the IP address of the SAP application server.

   System Number: The SAP system number. This must be provided by the SAP system administrator.

   Client: The SAP client number. This must be provided by the SAP system administrator.

   User Name: The user name with access rights to the SAP system. This name is supplied by the SAP system administrator.

   Language: EN for English or DE for German. If you select DE, the description text is displayed in German and all other text is displayed in English.

   The SAPRFC connection type requires the following connection information:

   RFC Destination: Enter the alias for the SAP connection information.

   In addition, both the connection types require the following connection information if the ABAP report is to be executed in SAP using a function module and the data file is to be transferred by FTP to Oracle Warehouse Builder:

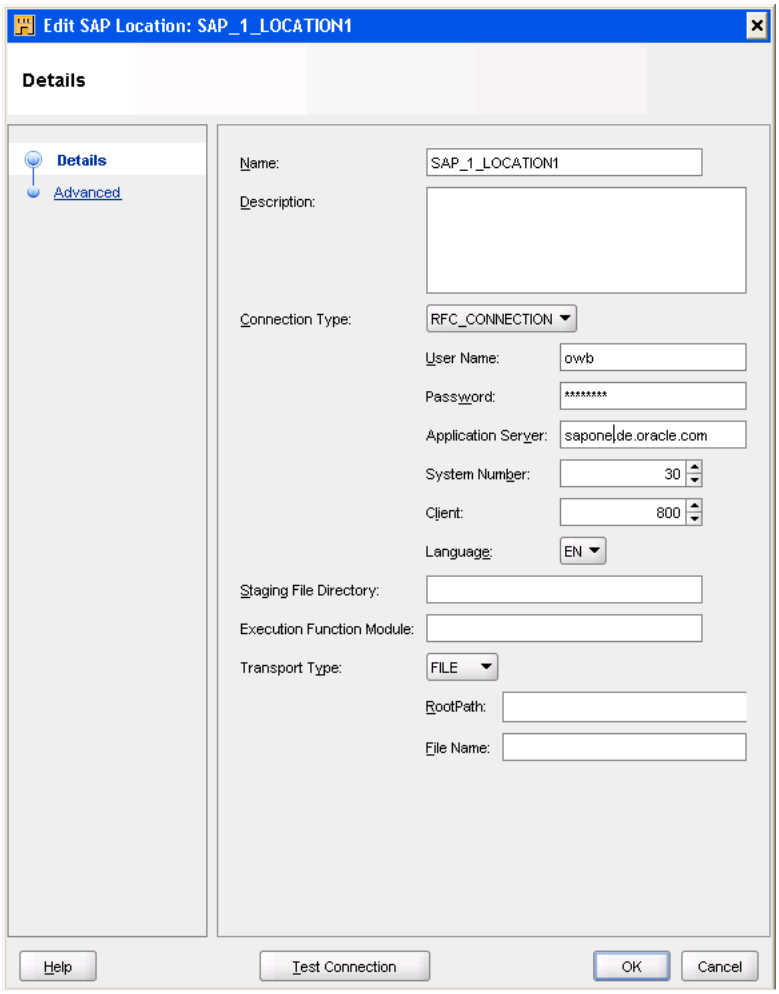
   Host Login User Name: A valid user name on the system that hosts the SAP application server. This user must have access rights to copy the data file using FTP.

FTP Directory: The directory in the SAP server that stores the data file generated when the ABAP report is executed. For systems where the FTP directory structure is identical to the operating system directory structure, this field can be left blank. For systems where the file system directory structure is mapped to the FTP directory structure, enter the FTP directory path that is mapped to the staging file directory in the file system directory structure. For example, on a computer that runs Windows, if the staging file directory `"C:\temp"` is mapped to `"/"` in the FTP directory structure, then enter `"/"` in this field.

Execution Function Module: In the SAP instance, if a remote function module other than the SAP delivered function module `RFC_ABAP_INSTALL_AND_RUN` is used to remotely execute ABAP reports through RFC connections, then enter the remote function module name here.

3. Click **Test Connection** to verify that the connection information you provided is correct.
4. Click **OK** to go back to the Connection Information page of the Create Module Wizard.

Figure 7–1 Edit SAP Location Dialog Box



7.7 Importing Metadata from SAP Tables

After you establish a connection with the SAP server, you can import metadata from SAP tables.

This section contains the following topics:

- "Importing SAP Metadata Definitions"
- "Analyzing Metadata Details"

7.7.1 Importing SAP Metadata Definitions

After creating the SAP source module, you can import metadata definitions from SAP tables using the Import Metadata Wizard. This wizard enables you to filter the SAP tables to import, verify those tables, and reimport them. You can import metadata for transparent tables, cluster tables, or pool tables.

To import SAP metadata:

1. From the Projects Navigator, expand the **Applications** node.
2. Right-click the SAP source module into which you want to import metadata and select **Import**.

   Oracle Warehouse Builder displays the Welcome page for the Import Metadata Wizard.
3. Click **Next.**
4. Complete the following tasks:
   - "Filtering SAP Metadata"
   - "Selecting Objects for Metadata Import"
   - "Reviewing Import Summary"

7.7.1.1 Filtering SAP Metadata

You can filter objects to import by business domain or by text strings. Select a filtering method and click **Next**.

7.7.1.1.1 Filtering SAP Metadata by Business Domain

1. Select **Business Domain** and click **Browse** to display the SAP R/3 Business Domain Hierarchy dialog box.

   The Import Metadata wizard displays the Loading Progress dialog box while it is retrieving the business domains.
2. The Business Domain Hierarchy dialog box lists the available SAP business domains.

   Note: It may take a few minutes to list the SAP business domains depending on factors such as the network location of the SAP application server, the type of LAN used, and the size of the SAP application database.

   Use the Business Domain Hierarchy dialog box to select the SAP business domains that contain the metadata tables you want to import.

3. Select a folder and click **Show Tables** to view the tables available in a business domain.

   The Import Metadata Wizard displays a list of tables in the selected business domain in the Folder dialog box.

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

   Some business domains can contain more than 1000 tables. Importing such a large amount of metadata can take time, depending on the network connection speed and the processing power of the source and target systems.

5. Click **OK.**

   The wizard displays the Filter Information page with the SAP business domain displayed in the Business Domain field.

7.7.1.1.2 Filtering SAP Metadata by Text String

1. Select **Text String, where object** and use the **Name matches** or **Description matches** entry field to enter a string and obtain matching tables from the SAP data source.

   The **Description matches** field is case sensitive, the **Name matches** field is not.

   Create a filter for object selection by using the wildcard characters `%` for zero or more matching characters, and _ for a single matching character.

   For example, to search the business domain for tables whose descriptions contain the word CURRENCY, select **Description matches** and enter `%CURRENCY%`. You can also search for tables by their names.

2. Specify the number of tables you want to import in the Maximum number of objects displayed field.

7.7.1.2 Selecting Objects for Metadata Import

The Object Selection page contains a description of the tables and enables you to select the tables you want to import into the SAP module.

To select the tables:

1. Move the tables from the available list to the selected list.

   The Import Metadata Wizard also enables you to choose whether you want to import tables with foreign key relationships for each table that you choose to import. You can select one of the following:

   None: Import only the tables in the Selected list.

   One Level: Import the tables in the Selected list and any tables linked to them directly through a foreign key relationship.

   All Levels: Import the tables in the Selected list and all tables linked to them through foreign key relationships.

2. Click **Next.**

   If you select One Level or All Levels, the Confirm Import Selection dialog box is displayed.

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

3. Click **OK.**

   The selected tables appear in the Selected list of the Table Selection page.

4. Click **Next.**

   The wizard displays the Summary and Import page.

7.7.1.3 Reviewing Import Summary

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

You can edit the descriptions for each table by selecting the **Description** field and entering a new description.

Review the information about the Summary and Import page and click **Finish.**

The SAP Connector reads the table definitions from the SAP application server and creates the metadata objects in the workspace.

The time it takes to import the SAP metadata into the workspace depends on the size and number of tables and the connection between the SAP application server and the workspace. It is a best practice to import small batches of tables to allow better performance.

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

7.7.1.4 Reimporting SAP Tables

To reimport SAP tables, follow the importing procedure using the Import Metadata Wizard. Before starting the import, the wizard checks the source for tables with the same name as those you are importing. The tables that have been imported appear in bold in the Object Selection page. On the Summary and Import page, the Action column indicates that these tables are reimported. The wizard then activates the **Advanced Synchronize Options** button so that you can control the reimport options.

Note: To undo the reimport, click **Undo**. This ensures that no changes are made to the existing metadata.

7.7.2 Analyzing Metadata Details

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

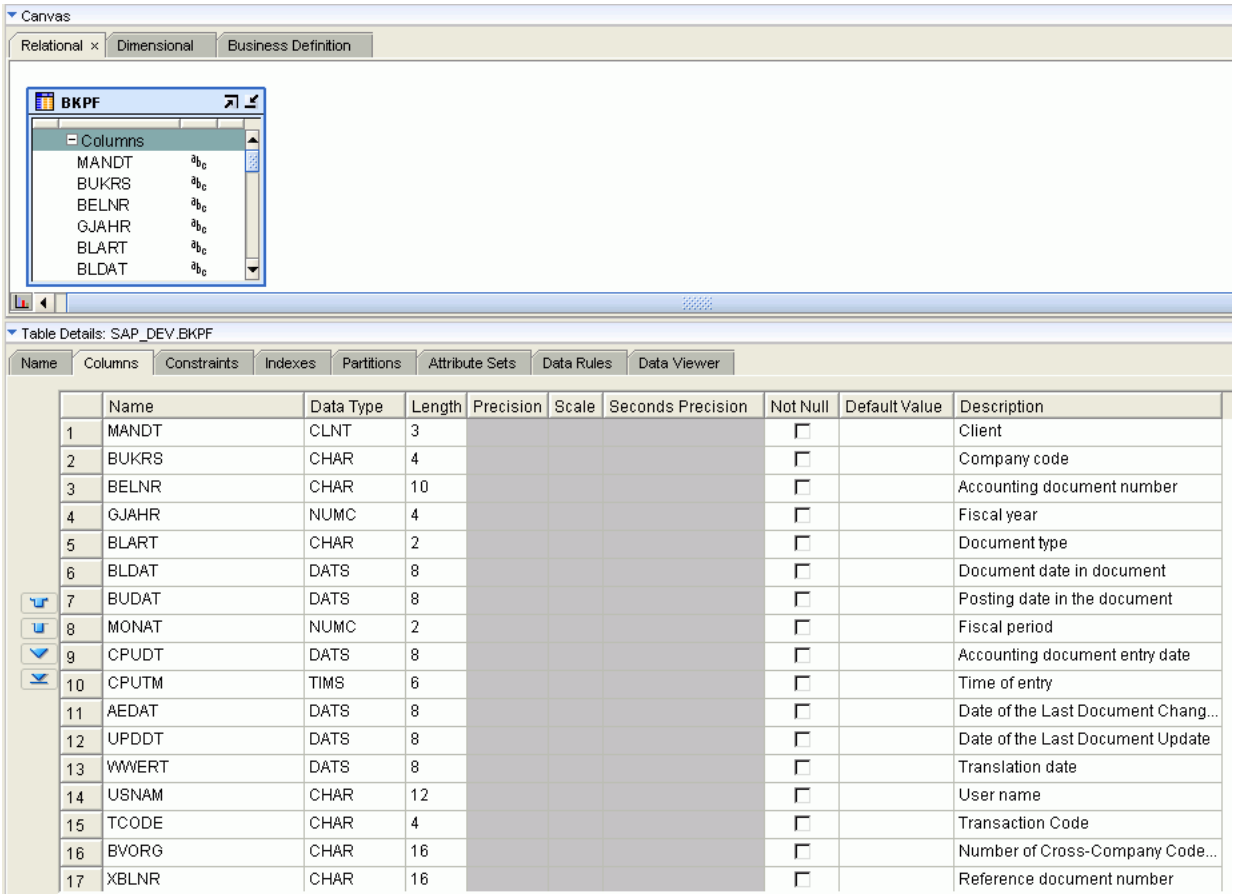
Column Descriptions

You can view the column description of each of the columns in a table. This is valuable because the column names in SAP can be nondescriptive, and difficult to interpret if you have not previously seen the data in the table.

To view the descriptions, double-click the table to open the data object editor for the table, and then click the **Columns** editor.

The description for the columns of the table are visible as shown in Figure 7–2.

Figure 7–2 The Columns Editor with the Description for the Columns of SAP Table



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

Constraints Details

The other benefit of the data object editor is that you can get information about the primary and foreign keys within the table. To view the key constraints, click the **Constraints** editor.

Note: It is also a useful practice to display the business names of the SAP tables in the Projects Navigator. Business names provide a description of the tables and are therefore more intuitive than the physical names. To view the business names for tables in Oracle Warehouse Builder, from the main menu, click **Tools**, **Preferences**, **OWB, Naming**, and then select **Business Names** in the Naming Mode field.

7.8 Creating SAP Extraction Mappings

After importing metadata from SAP tables, you must define the extraction mapping to extract data from the SAP system.

Note: For details of mappings in Oracle Warehouse Builder, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

7.8.1 Defining the SAP Extraction Mapping

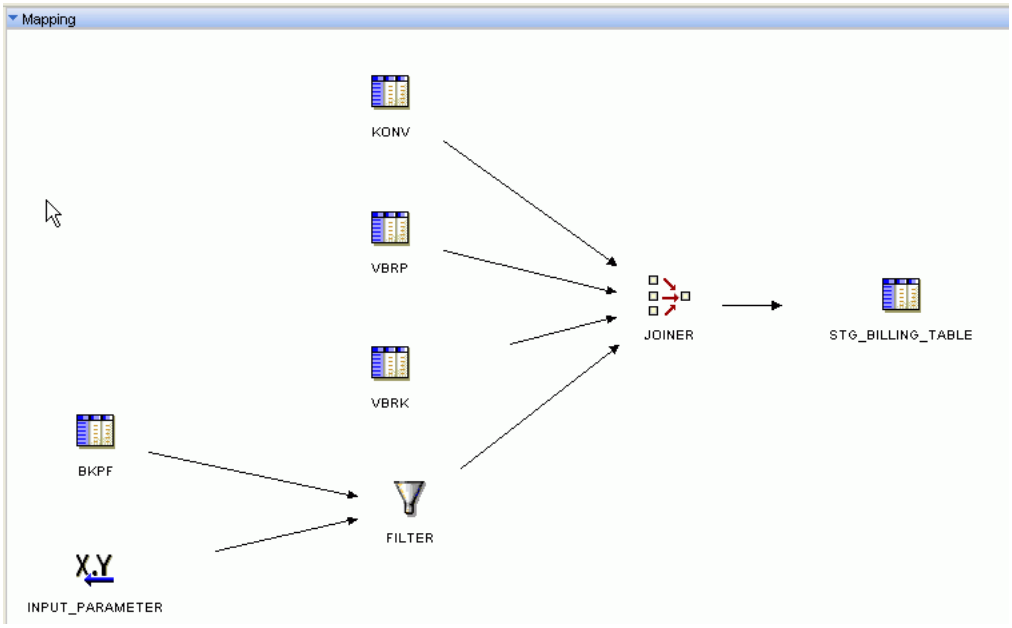
You can use the Mapping Editor to create a mapping containing SAP tables. Creating a mapping with SAP tables is similar to creating mappings with other database objects. However, there are restrictions on the operators that can be used in the mapping. You can only use table, filter, joiner, and mapping input parameter mapping operators in a mapping containing SAP tables.

A typical SAP extraction mapping consists of one or more SAP source tables (transparent, cluster, or pooled), one or more filter or joiner operators, and a non-SAP target table (typically an Oracle table) to store the imported data.

This is as shown in Figure 7–3.

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

Figure 7–3 SAP Extraction Mapping



In this mapping, the input parameter holds a date value, and the data from table `BKPF` is filtered based on this date. Because this is defined as an input parameter, you can enter a value for the date when the mapping is run. The joiner operator enables you to join data from multiple tables, and the combined data set is stored in a staging table.

This section contains the following topics:

- "Adding SAP Tables to the Mapping"
- "Setting the Loading Type"
- "Setting Configuration Properties for the Mapping"

7.8.1.1 Adding SAP Tables to the Mapping

To add an SAP table to a mapping:

On the mapping editor, drag and drop the required SAP table onto the Mapping Editor canvas.

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

7.8.1.2 Setting the Loading Type

Use the operator properties panel of the Mapping Editor to set the SQL*Loader properties for the tables in the mapping.

To set the loading type for the SAP Source Table:

1. On the Mapping Editor, select the SAP source table. The Table Operator Properties panel displays the properties of the SAP table operator.
2. Select a loading type from the Loading Type list. With ABAP code as the language for the mapping, the SQL*Loader code is generated as indicated in Table 7–4.

Table 7–4 SQL*Loader Code Generated in ABAP

Loading Type	Resulting Load Type in SQL*Loader
INSERT	APPEND
CHECK/INSERT	INSERT
TRUNCATE/INSERT	TRUNCATE
DELETE/INSERT	REPLACE
All other types	APPEND

7.8.1.3 Setting Configuration Properties for the Mapping

- Use the Configuration Properties dialog box to define the code generation language as described in "Setting the Language Parameter".
- Set ABAP specific parameters, and the directory and initialization file settings in the Configuration Properties dialog box as described in "Setting Run-time Parameters".

Setting the Language Parameter

The language parameter enables you to choose the type of code you want to generate for a mapping. For mappings containing SAP source tables, Oracle Warehouse Builder enables you to select either PL/SQL or ABAP.

If the SAP system uses a non-Oracle database to store data, then you must select ABAP to generate code. If the SAP data is stored on an Oracle database, then you can specify PL/SQL. However, with PL/SQL, you cannot extract pool or cluster tables. Therefore, in all instances it is desirable to set the language to ABAP.

Setting Run-time Parameters

With the language set to ABAP, you can expand the Runtime Parameters node in the Configuration Properties dialog box to display settings specific to ABAP code generation.

Some of these settings come with preset properties that optimize code generation. It is recommended that these settings be retained, because altering them may slow down the code generation process.

The following run-time parameters are available for SAP mappings:

- **ABAP Report Name**: Specifies the name of the ABAP report generated by the mapping. This is required only when you run a custom function module to execute the ABAP report.
- **Background Job**: Select this option to run the ABAP report as a background job in the SAP system. Enable this option for the longer running jobs. Foreground batch jobs that run for a long duration are considered to be hanging in SAP after a certain time. Therefore it is ideal to have a background job running for such extracts.
- **Control File Name**: By default, the control file name equals the data file name specified in the Data File Name field. This implies that the ABAP code generates a single control file that contains both, the SQL*Loader control information and the data (since the log files are the same). You can assign different file names for the control file and the data file, in which case different files are generated for the control information and data, and both the files are transferred by FTP.
- **Data File Name**: Specifies the name of the data file that is generated when the ABAP report is executed in the SAP system.
- **File Delimiter for Staging File**: Specifies the column separator in a SQL data file.
- **Include FTP**: If this is set to `true`, then the data file is moved to the Oracle Warehouse Builder system using FTP. If this is set to `false`, then the file is not transferred.
- **INSTALL ONLY**: When you set this to `true` and run the mapping, the ABAP report gets installed on the SAP system, but does not get executed. To execute the ABAP report, you must run the mapping again and use the execution function module. Setting this option to `true` enables you to generate the mapping, install the ABAP report on the SAP system, and subsequently make modifications to the ABAP report and then execute it. You can use this option when you want the ABAP report to be stored on the SAP system. If this option is set to `false`, then the ABAP report is loaded and executed on the SAP system. However, the ABAP report is not stored on the SAP system after it is executed.
- **SAP Location**: Specifies the location of the SAP instance from where the data can be extracted.
- **SAP System Version**: Specifies the SAP system version number to which you want to deploy the ABAP report. The characteristics of the generated ABAP report depends on the version number. For MySAP ERP and all other versions, select SAP R/3 4.7. Different ABAP report is generated for versions before 4.7.
- **SQL Join Collapsing**: If this is set to `true`, then it specifies the following hint, if possible, to generate ABAP report.

  ```
  SELECT < > INTO < > FROM (T1 as T1 inner join T2 as T2) ON <condition >
  ```

- **Staging File Directory**: Specifies the location of the directory in the SAP system where the data file generated by ABAP report resides.
- **Timeout**: This specifies the duration, in seconds, for which Oracle Warehouse Builder waits for the SAP system to execute the ABAP report and return a data file. If the SAP system completes the execution within this duration, then Oracle Warehouse Builder automatically retrieves the resultant data file from the SAP system. If the execution takes longer than the specified duration, then you must manually retrieve the data file.

7.8.1.4 Setting the Join Rank

You must set this parameter only if the mapping contains the joiner operator, and you want to explicitly specify the driving table. Unlike SQL, ABAP code generation is rule-based. Therefore, you must design the mapping in such a way that the tables are loaded in the right order. Or, you can explicitly specify the order in which the tables are to be joined. From the Configuration Properties dialog box, expand **Table Operators**, and then for each table, specify the Join Rank. The driving table must have the Join Rank value set to 1, with increasing values for the subsequent tables.

You can also let Oracle Warehouse Builder decide the driving table, and the order of joining the other tables. In such cases, do not enter values for Join Rank.

7.9 Deploying and Executing SAP ABAP Mappings

After designing the extraction mapping, you must validate, generate, and deploy the mapping, as you do with all mappings in Oracle Warehouse Builder.

To generate the script for the SAP mapping:

1. Right-click the SAP mapping and select **Generate.**

   The Generation Results window is displayed.

2. On the Script tab, select the script name and select **View Code.**

   The generated code is displayed in the code viewer.

   You can edit, print, or save the file using the code editor. Close the code viewer to return to the Generation Results window.

3. To save the file, click **Save as File** and save the ABAP program to the hard drive.

After you generate the SAP mapping, you must deploy the mapping to create the logical objects in the target location.

To deploy an SAP mapping, right-click the mapping and select **Deploy**. You can also deploy the mapping from Control Center Manager.

For detailed information about deployment, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

When an SAP mapping is deployed, an ABAP mapping is created and stored in the Oracle Warehouse Builder run time schema. Oracle Warehouse Builder also saves the `ABAP` file under *`OWB_HOME`*`\owb\deployed_files`, where *`OWB_HOME`* is the location of the Oracle home directory of your Oracle Warehouse Builder installation.

If you are using the Oracle Warehouse Builder installation that comes with Oracle Database, then this equals the database home.

Depending on whether data retrieval from the SAP system is fully automated, semi-automated, or manual, you must perform the subsequent tasks described in the following sections:

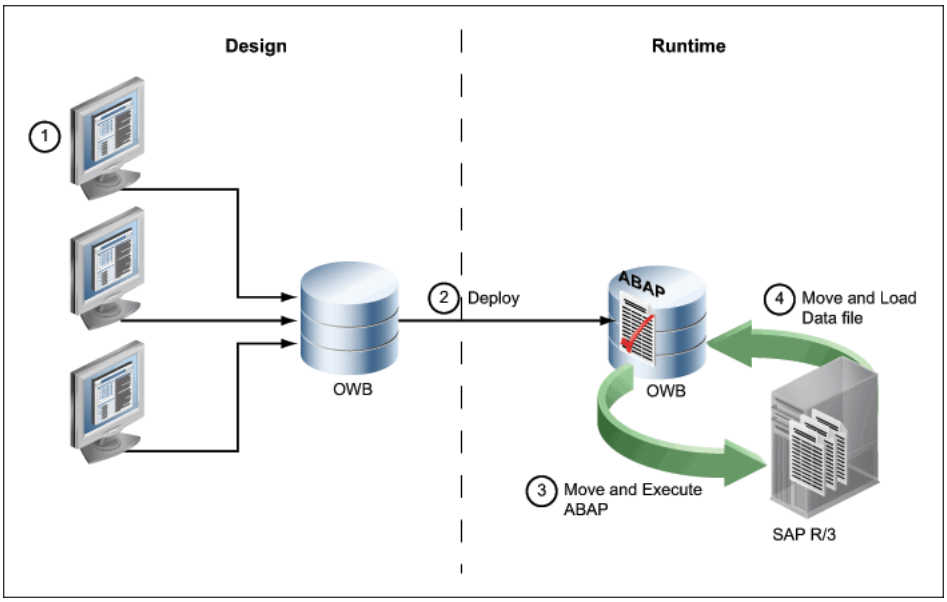
- "Automated System" on page 7-17
- "Semi-Automated System" on page 7-19
- "Manual System" on page 7-20

7.9.1 Automated System

In a completely automated system, as an Oracle Warehouse Builder user you have access to the predefined function module in the SAP system. It enables you to execute

any ABAP report and extract data directly from the SAP system without being dependent on the SAP administrator, as shown in Figure 7–4.

Figure 7–4 Automated Data Retrieval



You can automate the process of transferring the ABAP report to the SAP system and generating the data file on the SAP system. After the data file is generated, Oracle Warehouse Builder uses FTP to transfer the data file to the Oracle Warehouse Builder system. The data file is then loaded into the target file by using SQL*Loader.

An automated system works as follows:

1. You design the extraction mapping and generate the ABAP report for this mapping.
2. Before deploying the mapping, you must ensure that the following configuration properties are set for the mapping:
   - **ABAP Report Name**: The file that stores the ABAP code generated for the mapping.
   - **SAP Location**: The location on the SAP system from where data is extracted.
   - **Data File Name**: The name of the data file to store the data generated by the execution of ABAP report.

   Also ensure that you have provided the following additional connection details for the SAP location:

   - **Execution Function Module**: Provide the name of the predefined SAP function module. When executed, this function module takes the ABAP report name as the parameter, and executes the ABAP code.
   - **FTP Directory**: A directory on the SAP system. The data file generated upon the execution of the function module is saved to this directory. Oracle Warehouse Builder uses FTP to transfer the file from this directory to the Oracle Warehouse Builder system. This requires an FTP server to be located in the SAP system.
   - Also provide a user name with *Write* permissions on the FTP directory.

3. You then start the mapping, after which the following tasks are automatically performed:

   - Oracle Warehouse Builder deploys the ABAP report and the function module RFC_ABAP_INSTALL_AND_RUN is used to load and execute the ABAP report in the SAP system.

     The ABAP report is sent to the SAP system using a "Remote Function Call (RFC)".

4. In the SAP system, the ABAP report extracts data from the source tables and creates a data file.

   This data file is stored in the location specified by the Staging File Directory. See "Setting Run-time Parameters" on page 7-15 for details of the staging file directory.

5. Oracle Warehouse Builder uses FTP to transfer this data file back to the Oracle Warehouse Builder system.

   The file is stored in the location specified in the FTP Directory field.

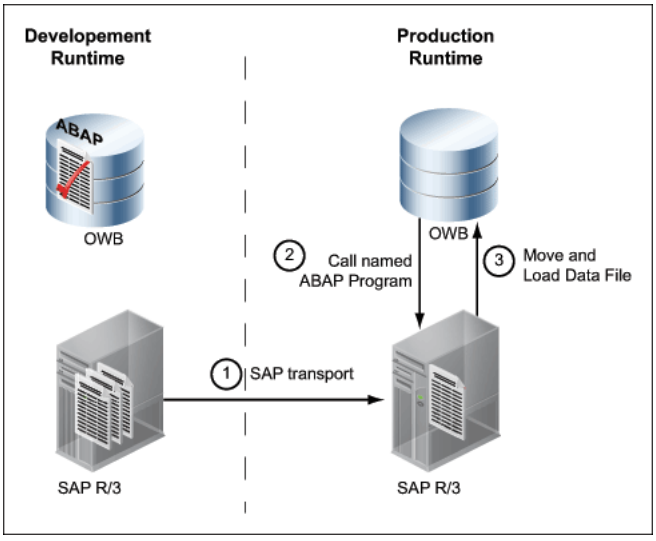
6. Using SQL*Loader, Oracle Warehouse Builder loads the target table specified in the mapping with the data from the data file.

The advantage of this system is that you can create a fully automated end-to-end solution to extract SAP data. As a user, you create the extraction mapping, and run it from Oracle Warehouse Builder, which then creates the ABAP report, sends it to the SAP system, extracts the resultant data file, and loads the target table with the extracted data.

7.9.2 Semi-Automated System

In a semi-automated system, as a Oracle Warehouse Builder user, you do not have access to the predefined function module `RFC_ABAP_INSTALL_AND_RUN`, and therefore cannot use this function module to execute ABAP report. Instead, you must create an extraction mapping, deploy it, and then send the ABAP report to the SAP administrator who verifies the code before allowing you to run it in the SAP system, as shown in Figure 7–5.

Figure 7–5 Semi-Automated Implementation



A semi-automated system works as follows:

1. You design the extraction mapping and generate the ABAP report for this mapping.

   You can then test this report in the development environment.

2. You then send the ABAP report to the SAP administrator, who tests the report, and loads it to the SAP repository in the production environment.

3. The SAP administrator can create a new report or use the same report that you send.

4. If the SAP administrator creates a new report, then obtain the name of the new report and use it in your mapping to extract data from the production environment.

5. Before you run the mapping in the production environment, ensure that you have set the following configuration properties for the mapping:

   - **ABAP Report Name**: The SAP administrator provides the name of the ABAP report after verifying the code. You must then use this report name to extract data.
   - **SAP Location**: The location on the SAP system from where data is extracted.
   - **Data File Name**: Name of the data file to store the data generated during execution of ABAP report.

   Also ensure that you have provided the following additional connection details for the SAP location:

   - **Execution Function Module**: Provide the name of the custom function module created by the SAP administrator. On execution, this function module takes the ABAP report name as the parameter, and executes the ABAP code. You must obtain the function module name from the SAP administrator.
   - **FTP Directory**: A directory on the SAP system. The data file generated by the execution of the ABAP report is saved to this directory. Oracle Warehouse Builder imports the data file using FTP. The FTP server resides on the SAP system.
   - Also provide a user name with *Read* permissions on the FTP directory.

6. In the production environment, when you run the mapping, Oracle Warehouse Builder sends the ABAP report name and the custom function module to the SAP system using a "Remote Function Call (RFC)".

7. In the SAP system, the ABAP report gets executed and a data file is generated. The ABAP report gets executed only if the ABAP report name and the function module are available.

   This data file is stored in the location specified by the Staging File Directory.

8. Oracle Warehouse Builder imports the data file using FTP. An FTP server must be available on the SAP server.

9. Oracle Warehouse Builder uses SQL*Loader to load the target table with data from the data file.

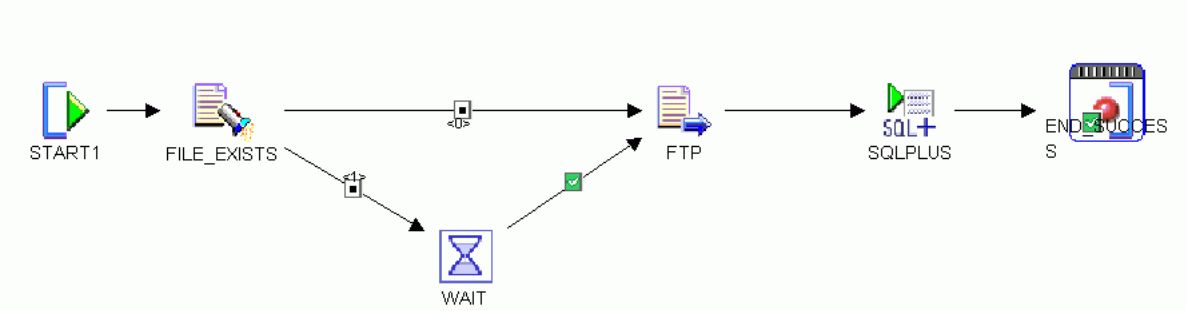
7.9.3 Manual System

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

1. You create an extraction mapping, and generate the ABAP report for the mapping.
2. While designing the mapping, ensure that you specify the Data File Name to store the data file.
3. You send the ABAP report to the SAP administrator.
4. The SAP administrator executes the ABAP report in the SAP system.
5. On execution of the code, a data file is generated.

You can then create a process flow to import the data file. The process flow may typically consist of the activities shown in Figure 7–6. For more details on implementing process flows, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

Figure 7–6 Process Flow to Import SAP Data



1. A File Exists activity checks for the availability of the data file.
2. If the file exists, then an FTP activity transfers the file to the Oracle Warehouse Builder system.
3. If the file does not exist, then it must wait till the file is made available, and then perform an FTP.
4. Using SQL*Loader, the target table is loaded with data from the data file.

In most production environments, the SAP administrator may not allow any other user to access the SAP system. In such cases, implementing the manual system may be the only viable option.

Using SQL*Loader in the Process Flow

To use SQL*Loader in the process flow, insert a SQL*Plus activity as shown in Figure 7–6. To use the SQL*Loader, use the HOST command. Once you insert the SQL*Plus activity, insert the following value for SCRIPT:

```
HOST sqlldr ${Target.User}/${Target.Password} CONTROL=${Working.RootPath}\C.CTL
quit
```

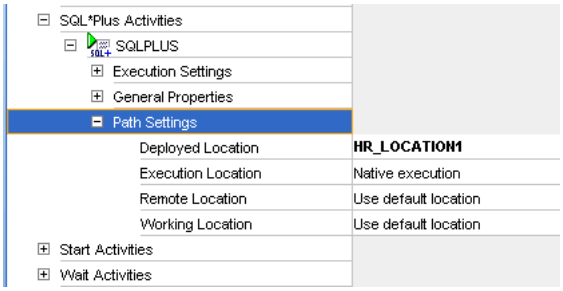
Insert the relevant value for the control (`.ctl`) file name.

Then configure the path settings for the SQL*Plus activity. Right-click the process flow and select **Configure**.

Under SQL*Plus Activities, expand the SQLPLUS node and provide the required values under Path Settings as shown in Figure 7–7.

Deployed Location refers to the location of the target table. Working location refers to the location of the control file.

Figure 7–7 Path Setting for the Process Flow



8

Integrating with Oracle ERP Applications

Oracle Warehouse Builder can integrate with Oracle's enterprise applications including E-Business Suite, PeopleSoft, and Siebel. Metadata for objects in the applications' underlying database schemas, such as tables, views and sequences, can be imported into a project, and then used like any other database objects in ETL mappings as sources and targets.

This chapter contains the following topics:

8.1 Importing Metadata from Oracle E-Business Suite Applications

Oracle E-Business Suite of applications provide comprehensive business solutions that can be implemented across a wide variety of industry functions including customer relationship management (CRM), project management, supply chain management (SCM), product life cycle management, financial management, and so on.

Before You Begin

Contact the database administrator for the E-Business Suite application and request a user name and password for accessing the APPS schema. The DBA may have previously created a user by running the script `owbebs.sql` as described in *Oracle Warehouse Builder Installation and Administration Guide*. If not, you must provide the DBA with a list of the tables, views, sequences, and keys from which you want to extract data. This is so that the DBA can grant object access privileges for these tables, views, sequences, and keys to the specific user. The user must be granted object privilege because the objects are accessed using PL/SQL in the extraction mappings.

Depending on the preference of the DBA, there may be a single user who extracts the metadata and the data. Or, there may be two separate users to access the metadata and data respectively.

8.1.1 Importing E-Business Suite Metadata Definitions

After creating the E-Business Suite source module, you can import metadata definitions from E-Business Suite objects using the Import Metadata Wizard. This wizard enables you to filter the E-Business Suite objects you want to import and verify those objects. The E-Business Suite module contains nodes for tables, views, and sequences. You can also create a User Folder within the module to organize the objects

that you import. See "Creating User Folders" on page 2-8 for more information about User Folders.

To import E-Business Suite metadata:

1. Create an E-Business Suite module as described in "Creating an E-Business Suite Module".
2. Import metadata from the E-Business Suite application as described in "Importing E-Business Suite Metadata".

8.1.1.1 Creating an E-Business Suite Module

To create an E-Business Suite module

1. Right-click **Oracle E-Business Suite** under the Applications node and select **New Oracle EBS Module**.

   The Create Module Wizard is displayed.

2. Click **Next** to display the Name and Description page.
3. Specify a name and an optional description for the source module. Also select the access method. You can either make a native heterogeneous connectivity or a Gateways connection. Click **Next**.

   The Connection Information page is displayed.

4. Specify the connection information for the E-Business Suite source module and click **Next**.

   Ensure that the location associated with the module contains information needed to connect to the E-Business Suite source. If you created a location earlier, associate that location with the module being created by selecting the location on the Connection Information page.

   To create a new location, click **Edit** on the Connection Information page of the Create Module Wizard. The Edit Non-Oracle Location dialog box is displayed.

   See "Edit Non-Oracle Location Dialog Box" for details of the values to be entered in this dialog box.

5. On the Summary page, review the options entered on the previous wizard pages. Click **Back** to modify any selections. Click **Finish** to create the E-Business Suite source module.

8.1.1.2 Edit Non-Oracle Location Dialog Box

Use the Edit Non-Oracle Location dialog box to create a location for the source module.

Name

Provide a name for the location.

Description

Provide an optional description.

Connection Type

Lists the connections available to access a location. You cannot change the connection type after creating a location.

- **HOST:PORT:SERVICE**: Makes a connection using the Easy Connect Naming method, which requires no prior setup:
  - **User Name**: The database user credential that has permission to access the schema location.

    When connecting to an application that does not have user names, enter any text as a mock user name.
  - **Password**: The password associated with user name.

    When connecting to an application that does not require a password, enter any text as a mock password.
  - **Host**: The name of the system where the application is installed.

    If the application is installed on the same system as the Oracle Warehouse Builder, you can enter `localhost` instead of the computer name.
  - **Port**: The SQL port number to access the application.
  - **Service Name**: The service name of the application.
  - **Use Global Name**: The unique name of the application, which is composed of the application name and the domain in the form *application_name.application_domain*. Select this option when connecting to an application on a different network.
- **Database Link**: Makes a connection to another database using an existing database link. Select this method only when you do not have privileges that enable you to make a direct connection. You cannot deploy to a location that uses a database link.

  A database link is a schema object that contains information for connecting to a remote database. Database links are used in distributed database environments and enable a client to access two physical databases as one logical database.
  - **From Location**: An existing location where the database link is defined.
  - **Database Link**: The object name of the database link.
- **SQL*NET Connection**: Makes a connection using a net service name previously defined using a tool such as Oracle Net Configuration Assistant. The net service name provides a convenient alias for the connection information.
  - **User Name**: The user credential that has permission to access the schema location.

    When connecting to an application that does not have user names, enter any text as a mock user name.
  - **Password**: The password associated with user name.

    When connecting to an application that does not require a password, enter any text as a mock password.
  - **Net Service Name**: The name of the predefined connection.
  - **Use Global Name**: The unique name of the database, which is composed of the database name and the domain in the form *application_name.application_domain*. Select this option when connecting to an application on a different network.

Schema

The schema where the source data is stored or the target objects are deployed. The schema must be registered. By default, it is the *User Name* schema.

When connecting to a type of application that does not have schemas, leave this field empty.

8.1.1.3 Importing E-Business Suite Metadata

To import metadata into the module:

1. Right-click the E-Business Suite source module and select **Import**, and then **Database Objects**.

   The Import Metadata Wizard is displayed.

2. Complete the following tasks using the wizard.
   - "Filtering E-Business Suite Metadata"
   - "Selecting the Objects"
   - "Reviewing Import Summary"

8.1.1.4 Filtering E-Business Suite Metadata

The Import Metadata Wizard includes a Filter Information page that enables you to select the metadata. Oracle Warehouse Builder provides two filtering methods:

- Business Domain

  This filter enables you to browse E-Business Suite business domains to locate the metadata you want to import. You can view a list of objects contained in the business domain and the names of the objects in the E-Business Suite application. For more information, see "Filtering E-Business Suite Metadata by Business Domain" on page 8-4.

- Object Type

  This filter enables you to search by the object type such as tables, views, and sequences or to search by entering text string information in the field provided in the Filter Information page. This is a more specific search method if you are familiar with the contents of your E-Business Suite application database. For more information, see "Filtering E-Business Suite Metadata by Object Type" on page 8-5.

Select a filtering method and click **Next** to proceed with the importing of metadata.

8.1.1.4.1 Filtering E-Business Suite Metadata by Business Domain To filter metadata by the business domain:

1. Select **Business Domain** and click **Browse** to open the Business Domain Hierarchy dialog box.
2. The Business Domain Hierarchy dialog box lists the available E-Business Suite business domains.

   Note: The time it takes to list the business domains depends on the network location of the E-Business Suite application server, the type of LAN used, and the size of the E-Business Suite application database.

   Use the Business Domain Hierarchy dialog box to select the E-Business Suite business domains that contain the metadata objects you want to import.

3. Select a business domain and click **Show Entities**.

   The Folder dialog box displays a list of objects available in the selected business domain.

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

   Some business domains can contain more than 1000 objects. The time required to import such a large amount of metadata can vary depending on the network connection speed and the processing power of the source and target systems. It is recommended that you review the list of objects and import only the ones that are required.

5. Click **OK.**

   The wizard displays the Filter Information page with the E-Business Suite business domain displayed in the Business Domain field.

8.1.1.4.2 Filtering E-Business Suite Metadata by Object Type

1. Select **Object Type**.
2. Select the type of objects you want to import. You can select Tables, Views, and Sequences.

   To select specific objects, enter the object name in the text field. Create a filter for object selection by using the wildcard characters (%) for zero or more matching characters, and (_) for a single matching character.

   For example, to search the business domain for tables whose names contain the word CURRENCY, then type `%CURRENCY%`. To refine the search to include only tables named CURRENCY and followed by a single digit, then type `%CURRENCY_`.

8.1.1.5 Selecting the Objects

The Object Selection page contains a description of the objects and enables you to select the objects you want to import into the E-Business Suite module.

To select the objects:

1. Move the objects from the available list to the selected list.

   The Import Wizard also enables you to choose to import tables with foreign key relationships for each object that you choose to import. You can select one of the following:

   None: Import only the objects in the Selected list.

   One Level: Import the objects in the Selected list and any tables linked to it directly through a foreign key relationship.

   All Levels: Import the objects in the Selected list and all tables linked to it through foreign key relationships.

   The foreign key level you select is the same for all tables selected for importing.

   > **Note:** Selecting **All Levels** increases the time it takes to import the metadata because you are directing the wizard to import tables that are related to each other through foreign key constraints. Select this option only if it is necessary.

2. Click **Next.**

   If you select **One Level** or **All Levels**, then the Confirm Import Selection dialog box is displayed if other related tables are to be imported due to foreign keys or other dependencies.

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

3. Click **OK.**

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

4. Click **Next.**

   The Import Wizard displays the Summary and Import page.

8.1.1.6 Reviewing Import Summary

The Import Metadata Wizard imports definitions for the selected objects from the E-Business Suite application server, stores them in the E-Business Suite source module, and then displays the Summary and Import page.

You can edit the descriptions for each object by selecting the description field and entering a new description.

Review the information about the Summary and Import page and click **Finish** to import the selected objects.

The E-Business Suite integrator reads the table definitions from the E-Business Suite application server and creates the metadata objects in the module.

The time it takes to import the E-Business Suite metadata to the workspace depends on the size and number of tables and the connection between the E-Business Suite application server and the workspace, especially if you are connecting servers located in different local area networks (LANs).

When the import completes, the Import Results dialog box is displayed. Click **OK** to finish importing or click **Undo** to cancel the import.

8.2 Importing Metadata from PeopleSoft Applications

Oracle's PeopleSoft Enterprise applications provide comprehensive business solutions in a wide variety of industry functions including Human Resource Management System (HRMS), Financials, Customer Relationship Management (CRM), and Material Management. A PeopleSoft application consists of numerous modules, each pertaining to a specific area in an enterprise.

8.2.1 Importing PeopleSoft Metadata Definitions

To import metadata from PeopleSoft applications, create a PeopleSoft module from the Projects Navigator. After creating the module, you can import metadata definitions from the PeopleSoft application using the Import Metadata Wizard. This wizard enables you to filter the objects you want to import and verify those objects. The PeopleSoft module contains nodes for tables, views, and sequences. You can also create a User Folder within the module to organize the objects that you import. You can import metadata for tables, views, and sequences.

To import PeopleSoft metadata:

1. Create a PeopleSoft module as described in "Creating a PeopleSoft Module".

2. Import metadata from the PeopleSoft application as described in "Importing PeopleSoft Metadata".

8.2.1.1 Creating a PeopleSoft Module

To create a PeopleSoft module

1. Right-click **PeopleSoft** under the Applications node and select **New Peoplesoft Module**.

   The Create Module Wizard is displayed.

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

2. Click **Next** to display the Name and Description page.

3. Specify a name and an optional description for the source module. Also specify whether you want to connect through native database connectivity (using JDBC drivers) or through Oracle Gateways.

   If you select Oracle Gateways, then select the database from which the metadata is to be imported. Click **Next**.

   The Connection Information page is displayed.

4. Specify the connection information for the PeopleSoft source module and click **Next**.

   Ensure that the location associated with the module contains information needed to connect to the PeopleSoft source. If you created a location earlier, associate that location with the module being created by selecting the location on the Connection Information page.

   To create a new location, click **Edit** on the Connection Information page of the Create Module Wizard. This opens the Edit Non-Oracle Location dialog box. See "Edit Non-Oracle Location Dialog Box" on page 8-2 for more information about the values to be entered in the dialog box.

5. On the Summary page, review the options entered on the previous wizard pages. Click **Back** to modify any selections. Click **Finish** to create the PeopleSoft source module.

   See Also: Chapter 6, "Connecting to Data Sources Through JDBC" for more information about native database connectivity.

   Chapter 4, "Connecting to Non-Oracle Data Sources Through Gateways" for more information about Gateways connectivity.

8.2.1.2 Importing PeopleSoft Metadata

To import metadata into the module:

1. Right-click the PeopleSoft module and select **Import**, **Database Objects**.

   The Import Metadata Wizard is displayed.

2. Complete the following tasks:

   - "Filtering PeopleSoft Metadata"
   - "Selecting the Objects"
   - "Reviewing Import Summary"

8.2.1.3 Filtering PeopleSoft Metadata

The Import Metadata Wizard includes a Filter Information page that enables you to select the metadata. Oracle Warehouse Builder provides two filtering methods:

- Business Domain

  This filter enables you to browse PeopleSoft business domains to locate the metadata you want to import. You can view a list of objects contained in the business domain. For more information, see "Filtering PeopleSoft Metadata by Business Domain" on page 8-8.

- Object Type Matching

  This filter enables you to search by the object type such as tables, views, and sequences or to search by entering text string information in the field provided on the Filter Information page. This is a more specific search method if you are familiar with the contents of your PeopleSoft application database. For more information, see "Filtering PeopleSoft Metadata by Object Type" on page 8-8.

Select a filtering method and click **Next** to proceed with the importing of metadata.

8.2.1.3.1 Filtering PeopleSoft Metadata by Business Domain To filter by the business domain:

1. Select **Business Domain** and click **Browse** to open the Business Domain Hierarchy dialog box.

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

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

   Note: The time it takes to list the business domains depends on the network location of the PeopleSoft application server, the type of network used, or the size of the PeopleSoft application database.

   Use the Business Domain Hierarchy dialog box to select the PeopleSoft business domains that contain the metadata objects you want to import.

3. Select a folder and click **Show Entities**.

   The Import Wizard displays a list of objects in the selected business domain in the Folder dialog box.

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

   Some business domains can contain more than 1000 objects. The time it takes to import such a large amount of metadata can vary depending on the network connection speed and the processing power of the source and target systems. It is recommended that you review the list of objects and import only the ones that are required.

5. Click **OK.**

   The wizard displays the Filter Information page with the PeopleSoft business domain displayed in the Business Domain field.

8.2.1.3.2 Filtering PeopleSoft Metadata by Object Type

1. Select **Object Type**.

2. In the Object Type section, select the types of objects you want to import. You can select Tables, Views, and Sequences.

   To select specific objects, type the object name in the text field. Create a filter for object selection by using the wildcard characters (%) for zero or more matching characters, and (_) for a single matching character.

   For example, to search the business domain for tables whose names contain the word CURRENCY, then type `%CURRENCY%`. To refine the search to include only tables named CURRENCY and followed by a single digit, then type `%CURRENCY_`.

8.2.1.4 Selecting the Objects

The Object Selection page contains a description of the objects and enables you to select the objects you want to import into the PeopleSoft module.

To select the objects:

1. Move the objects from the Available list to the Selected list.

   The Import Wizard also enables you to choose to import tables with foreign key relationships for each object that you choose to import. You can select one of the following:

   None: Import only the objects in the Selected list.

   One Level: Import the objects in the Selected list and any tables linked to it directly through a foreign key relationship.

   All Levels: Import the objects in the Selected list and all tables linked to it through foreign key relationships.

   The foreign key level you select is the same for all tables selected for importing.

   > **Note:** Selecting **All Levels** increases the time it takes to import the metadata because you are directing the wizard to import tables that are related to each other through foreign key constraints. Select this option only if it is necessary.

2. Click **Next.**

   If you select **One Level** or **All Levels**, the Confirm Import Selection dialog box is displayed.

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

3. Click **OK.**

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

4. Click **Next.**

   The wizard displays the Summary and Import page.

8.2.1.5 Reviewing Import Summary

The Import Metadata Wizard imports definitions for the selected tables from the PeopleSoft application server, stores them in the PeopleSoft source module, and then displays the Summary and Import page.

You can edit the descriptions for each object by selecting the description field and entering a new description.

Review the information about the Summary and Import page and click **Finish.**

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

The time taken to import PeopleSoft metadata to the workspace depends on the available objects including tables, views, and sequences and the connection between the PeopleSoft application server and the workspace. Importing a large number of objects can result in delays so it is recommended that you import in smaller batches.

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

8.3 Importing Metadata from Siebel Applications

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

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

8.3.1 Importing Siebel Metadata Definitions

Before you import metadata definitions from Siebel, you must create a Siebel source module. You can then import metadata definitions from Siebel using the Import Metadata Wizard. This wizard enables you to filter the Siebel objects you want to import and verify those objects. The Siebel module contains nodes for tables, views, and sequences. You can also create a User Folder within the Siebel module to organize the objects that you import. You can import metadata for tables, views, and sequences.

To import metadata definitions from Siebel:

1. Create a Siebel source module, as described in "Creating a Siebel Source Module" on page 8-10.
2. Import metadata from Siebel, as described in "Importing Siebel Metadata" on page 8-11.

8.3.1.1 Creating a Siebel Source Module

1. From the Projects Navigator, click the **Applications** node to expand it.
2. Right-click **Siebel** and select **New Siebel Module**.

   The Create Module Wizard is displayed.
3. Specify a name and an optional description for the source module. Also specify whether you want to connect through native database connectivity (using JDBC drivers) or through Oracle Gateways.

   If you select Oracle Gateways, then select the database from which the metadata is to be imported. Click **Next**.

   The Connection Information page is displayed.
4. Specify a name and an optional description for the Siebel source module and click **Next**.

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

Ensure that the location associated with the Siebel module contains information needed to connect to the Siebel source. If you created a location earlier, associate that location with the module being created by selecting the location on the Connection Information page.

Or create a new location by clicking **Edit** on the Connection Information page to open the Edit non-Oracle dialog box. See "Edit Non-Oracle Location Dialog Box" on page 8-2 for more details.

6. On the Summary page, review the options entered on the previous pages. Click **Back** to modify any selections. Click **Finish** to create the Siebel source module.

See Also: Chapter 6, "Connecting to Data Sources Through JDBC" for more information about native database connectivity.

Chapter 4, "Connecting to Non-Oracle Data Sources Through Gateways" for more information about Gateways connectivity.

8.3.1.2 Importing Siebel Metadata

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

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

2. Click **Next.**

   The Filter Information page is displayed.

3. Select the objects to be imported and click **Next**.

   You can search for specific object types such as tables, sequences, and views. Oracle Warehouse Builder also enables you to specify the name of the object to be imported. You can search for specific object names by entering text string information in the field provided in the Filter Information page. This is a more specific search method if you are familiar with the contents of your Siebel application database.

4. On the Objects Selection page, select the objects to be imported into the Siebel module and click **Next**.

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

   None: Import only the objects in the Selected list.

   One Level: Import the objects in the Selected list and any tables linked to it directly through a foreign key relationship.

   All Levels: Import the objects in the Selected list and all tables linked to it through foreign key relationships.

   The foreign key level you select is the same for all tables selected for importing.

   Note: Selecting **All Levels** increases the time it takes to import the metadata because you are directing the wizard to import tables that are related to each other through foreign key constraints. Select this option only if it is necessary.

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

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

8.4 Importing Metadata from Applications Implemented on non-Oracle Databases

You can import metadata from PeopleSoft and Siebel applications that are implemented on non-Oracle databases such as DB2, SQL Server, Sybase, Informix, and so on.

With earlier releases of Oracle Warehouse Builder, you required an Oracle Gateways connection to import metadata from an application such as PeopleSoft implemented on a non-Oracle database, such as SQL Server. With Oracle Warehouse Builder 11*g* Release 2 (11.2), you can define a CMI definition and use a native heterogeneous connectivity using JDBC drivers to import from these applications.

See "Connecting to a PeopleSoft Application Implemented on SQL Server" on page 8-13 for more information about using a CMI definition to import metadata from a PeopleSoft application implemented on SQL Server.

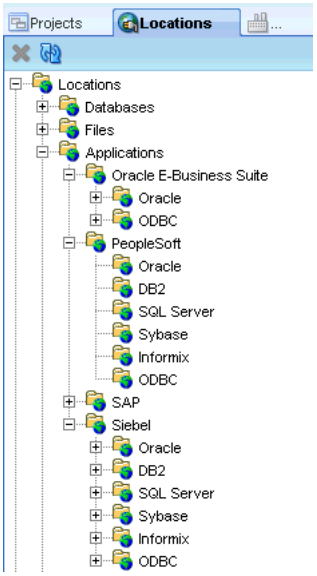
On the Locations Navigator, click an application to see the databases from which you can import metadata for that application as shown in Figure 8–1. Create a location based on the database on which the application is implemented. For example,

To define a location corresponding to a PeopleSoft application on SQL Server:

1. On the Locations Navigator, navigate to Applications, PeopleSoft.
2. Right-click **SQL Server** and select **New SQL Server PeopleSoft Location**.

   The Create SQL Server PeopleSoft Location dialog box is displayed.
3. Provide the connection details corresponding to the SQL Server database. See "Connecting to SQL Server Database" on page 6-3 for more information about the details to be specified to connect to an SQL Server database.

Figure 8–1 The Locations Navigator for the Oracle Applications



8.4.1 Connecting to a PeopleSoft Application Implemented on SQL Server

To import metadata from a PeopleSoft application that is implemented on a SQL Server database, first define a CMI definition for PeopleSoft on SQL Server.

To leverage on a CMI mechanism, you must create a CMI_DEFINITION for the PeopleSoft application.

CMI definitions must be created from the `root` context. You can switch to the `root` context only from the OMB Plus console. You cannot switch to the `root` context using the OMB*Plus view from within the Oracle Warehouse Builder UI. Therefore it is recommended that you use the OMB Plus console for the implementation.

For more information about OMB*Plus scripting, see *Oracle Warehouse Builder API and Scripting Reference*.

> **See Also:** *Oracle Warehouse Builder OMB*Plus Command Reference* for a list of all OMB*Plus commands.

To use the OMB Plus console on a Windows system, select **Start**, then **All Programs**, **<OWB>**, **Warehouse Builder**, and then **OMB Plus**.

To switch to the `root` context, use the following command:

```
OMBCONNECT <repository user>/<password>@<host>:<port number>:<service name>
```

For example,

```
OMBCONNECT rep_user/password@localhost:1521:orcl
```

Where `rep_user/password` is the user name/password to connect to the repository, `localhost` indicates a local installation, `1521` is the port number, and `orcl` is the service name of the database.

Example 8–1 lists the `.tcl` script to create a CMI. Run the script from the OMB Plus console.

Example 8–1 Script to Create a CMI Definition

```
set platformname SQLSERVER
set application PEOPLESOFT

set cmi $platformname\_$application
puts "Creating Custom Application Metadata Import"
OMBCREATE CMI_DEFINITION '$cmi' USING DEFINITION_FILE 'c:\\cmi\\platformdef4_
miv.xml'
OMBALTER CMI_DEFINITION '$cmi' SET PROPERTIES (MIV_TYPE) VALUES ('Applications')
puts "Created Custom Application Metadata Import"
puts "        - $application on $platformname"
puts ""
```

Save the changes using the command `OMBCOMMIT`.

The file `platformdef4_miv.xml` contains the CMI_DEFINITION for PeopleSoft on SQL Server as listed in Example 8–2. Ensure that you provide the correct file location for the `.xml` file in the `.tcl` script listed in Example 8–1.

Example 8–2 CMI Definition File for PeopleSoft on SQL Server

```
<?xml version="1.0"?>
<miv>
  <miv_tables type="SQLStatement" default="true" >
```

```
select
REC.RECNAME             as TABLE_NAME,
TAB.TABLE_NAMEas RES_NAME,
rtrim(REC.RECDESCR)as TABLE_DESC
  from
PSRECDEFN REC left outer join
 PS_EO_BCOWNRID_VW  M on ( REC.OBJECTOWNERID=M.OBJECTOWNERID),
INFORMATION_SCHEMA.TABLES TAB
where (REC.RECNAME =TAB.TABLE_NAME or 'PS_'+REC.RECNAME =TAB.TABLE_NAME)
and REC.RECTYPE =0 </miv_tables>

<miv_columns type="SQLStatement" default="true" />

  <miv_capabilities type="ResultSet">
    <table_supported>true</table_supported>
    <view_supported>false</view_supported>
    <sequence_supported>false</sequence_supported>
    <table_name_filter_supported>false</table_name_filter_supported>
    <view_name_filter_supported>false</view_name_filter_supported>
    <sequence_name_filter_supported>false</sequence_name_filter_supported>
    <business_area_supported>false</business_area_supported>
    <business_area_table_supported>false</business_area_table_supported>
    <business_area_view_supported>false</business_area_view_supported>
    <business_area_sequence_supported>false</business_area_sequence_supported>
    <application_owner_supported>true</application_owner_supported>
    <table_fklevel_supported>false</table_fklevel_supported>
    <reimport_supported>true</reimport_supported>
    <data_object_at_leaf_levels>true</data_object_at_leaf_levels>
    <multiple_tree_supported>false</multiple_tree_supported>
    <function_supported>false</function_supported>
   <function_name_filter_supported>false</function_name_filter_supported>
  </miv_capabilities>

</miv>
```

This MIV file is created using elements defined in an XML schema definition (XSD) file.

After you have created a CMI definition, create a PeopleSoft module from the Projects Navigator to connect to the PeopleSoft application implemented on the SQL Server database.

CMI_DEFINITION must have it's name defined as *platform_application* to work. The illustration creates an example called SQLSERVER_PEOPLESOFT. You get the list of platform names using the command `OMBLIST PLATFORMS`.

To create the PeopleSoft module:

1. Under the Application node, right-click **PeopleSoft** and select **New PeopleSoft Module**.

   The Create Module Wizard is displayed.

2. On the Name and Description page, provide a name and an optional description. Select the access method to be **Native Database Connection**. Also select **SQL Server** as the platform. This option is now available because you have created a CMI definition for PeopleSoft on SQL Server. Click **Next**.

   The Connection Information page is displayed.

3. Click **Edit** to define the location details of the PeopleSoft application on SQL Server. See "Creating a SQL Server Module" on page 6-3 for more information about the connection details to an SQL Server database.

   You require the JDBC drivers to connect to the SQL Server. For more information about the JDBC driver requirements, see "Connecting to SQL Server Database" on page 6-3.

After creating the module, you can import metadata from the PeopleSoft application.

9

Importing Oracle Warehouse Builder Data into Business Intelligence Applications

Oracle Warehouse Builder provides an end-to-end business intelligence (BI) solution by enabling you to import, design, and deploy metadata from different sources into a data warehouse, and by making that data available to business analytical tools for decision making and business reporting. It fully integrates relational, dimensional, and business metadata and provides access to business analytical tools for decision making and business reporting.

Oracle Warehouse Builder contains BI objects that enable you to integrate seamlessly with BI applications to perform data analysis. You can define BI objects that enable you to store definitions of business metadata. You can then deploy these definitions to Oracle's BI tools, such as Oracle Business Intelligence Suite Enterprise Edition (Oracle BI Suite EE) and Oracle BI Discoverer, thereby extending the functionality of your data warehouse.

This section contains the following topics:

9.1 Creating Business Definitions for Oracle BI Discoverer

Business definitions are the equivalent of Discoverer End User Layer (EUL) objects. They are BI objects that enable you to integrate Oracle Warehouse Builder data with Oracle BI Discoverer. Business definitions facilitate data analysis of the data stored in a data warehouse. You can define and then deploy business definitions to Oracle BI Discoverer. You can either create new business definitions or derive them from existing schemas. For information about creating business definitions, see "Creating Oracle Discoverer Module" on page 9-2. For information about deriving business definitions, see "Deriving BI Objects" on page 9-41.

This section contains the following topics:

9.1.1 Creating Oracle Discoverer Module

Before you derive business definitions to deploy to Discoverer, you must create an Oracle Discoverer module to store your business definitions.

To create an Oracle Discoverer module:

1. From the Projects Navigator, click to expand the **Project** node.
2. Expand the Business Intelligence node.
3. Right-click **Oracle Discoverer** and select **New Oracle Discoverer**.

   Oracle Warehouse Builder opens the Create Module Wizard.
4. Follow the wizard steps by clicking **Next.**

9.1.1.1 Naming the Oracle Discoverer Module

In the Name and Description page, enter a name and an optional description for the Oracle Discoverer module. Also, indicate the type of module you are creating.

For more information about naming conventions, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

9.1.1.2 Setting the Connection Information

On the Connection Information page, define the location to deploy the Oracle Discoverer module. For example, this may be the system where you are currently running Oracle BI Discoverer.

To use a deployment location you previously created, you can select it from the **Location** list. The connection information for this location displays on the wizard page.

You can also choose to create this location later and skip to the next page. You cannot deploy the Oracle Discoverer module unless you provide the connection information for the target location.

The wizard initially creates a default target location for the module you are creating. For example, if your module is named `DISCOVERER_OBJECTS`, then the location is called `DISCOVERER_OBJECTS_LOCATION1`. You can choose to provide the connection information for this location by clicking **Edit**. The Edit Oracle Discoverer Location dialog box is displayed. Provide the required information to connect with your target system and click **OK**. For more information about the Edit Oracle Discoverer Location dialog box, see "Defining Discoverer Locations" on page 9-3.

9.1.1.2.1 Defining Discoverer Locations A Discoverer location provides connection details of the system where you deploy the Oracle Discoverer modules. Oracle BI Discoverer EUL Release 10.1.2 or later should be installed on this system.

To define a Discoverer location, enter the following details on the Edit Oracle Discoverer Location dialog box:

- **Name:** The name of the Discoverer location. Oracle Warehouse Builder assigns a default name for the location. You can edit this name.
- **Description:** An optional description for the Discoverer location.
- **User Name:** The name of the EUL owner to which you want to deploy your business definitions. You can also specify a user who has administrator privileges.
- **Password:** The password for the user.

  User password is required only for direct integration.
- **Connection Type:** The type of connection used to connect to the Discoverer EUL. The options you can select are **Host:Port:Service** or **SQL*Net Connection**.

  When you select **SQL*Net Connection**, specify the net service name in the **Net Service Name** field. When you select **Host:Port:Service**, specify the following additional details.

  Host: The host name of the system on which the EUL exists.

  Port: The default port number is 1521.

  Service Name: The service name of the Oracle Database installation.
- **Integration Type**: Direct or Indirect depending on the connection. Direct indicates that the deployment is made directly to the schema. Indirect provides file transfer options through file, FTP, HTTP, or HTTPS.

  Depending on the mode of file transfer, you must provide the following details:

  FILE

  - **Root Path**: Directory of the `.eex` file.
  - **File Name**: The name of the `.eex` file.

  FTP

  - **Host Name**: The system credentials where the Oracle Discoverer server resides.
  - **Host Login Port**: Login port number, which is initially set to 0. You must change this according to your local configuration.
  - **Transfer Format**: Select from ASCII and IMAGE.

- **Host Login User**: User name to run FTP.
- **Host Login Password**: User password to run the FTP command.
- **File Name**: The name of the `.eex` file along with the complete path.

HTTP and HTTPS

- **Host Name**: The system credentials where the Oracle Discoverer server resides.
- **Host Login Port**: Login port number, which is initially set to 0. You must change this according to your local configuration.
- **Host Login User**: User name for the HTTP/HTTPS command.
- **Host Login Password**: User password for the HTTP/HTTPS command.
- **File Name**: The name of the `.eex` file along with the complete path.

- **Version:** Represents the version of Discoverer to which the business definitions should be deployed. The list contains only one value, 10.1. Use this option to deploy to Oracle BI Discoverer 10*g* Release 2. This includes all Oracle BI Discoverer 10.1.x versions.

After you specify these details, click **Test Connection** to verify the connection.

9.1.1.3 Reviewing the Summary Information

In the Summary page, review the name and location information for the Oracle Discoverer module. Click **Back** to make any changes or click **Finish** to finish creating the Oracle Discoverer module.

After the Oracle Discoverer module is created, you can locate it under the Oracle Discoverer node on the Projects Navigator. Expand the module to see that Oracle Warehouse Builder provides a representation for the different objects types that comprise the Discoverer EUL. You can define the following types of Discoverer EUL objects:

- Item Folders
- Business Areas
- Drill Paths
- Lists of Values
- Alternative Sort Orders
- Drills to Detail
- Registered Functions

9.1.2 About Item Folders

Item Folders are equivalent to Folder objects in Oracle BI Discoverer that map to database tables, external tables, or views. They represent a result set of data, similar to a database view. Item Folders store information just like tables. For example, they are used to store details of employees or customers of an organization. An Item Folder contains entities called Items that correspond to columns in a table. Each item has a name and contains a specific type of information. For example, the Item Folder that contains details about employees may include items such as employee name, start date, and department.

There are two types of Item Folders: Simple and Complex. Simple Item Folders contain items that reference a single table in an Oracle module. Complex Item Folders, like database views, provide a method to group items from multiple Item Folders within the same Oracle Discoverer module. Item Folders also contain joins, calculated items, and conditions.

> **Note:** Oracle Warehouse Builder does not support the Discoverer custom folders.

Oracle Warehouse Builder creates Item Folders when you derive business definitions from warehouse design objects in your Oracle module, as described in "Deriving BI Objects" on page 9-41. You can also manually create a customized Item Folder using the Create Item Folder Wizard or the Graphical Navigator. The Graphical Navigator can also be used to edit Item Folders.

The following sections contain more information related to Item Folders:

- "Editing an Item Folder"
- "Creating an Item Folder"

9.1.3 Editing an Item Folder

After you derive your design object definitions, an Item Folder is created as part of the derived business definitions.

Oracle Warehouse Builder provides the document editors that enable you to edit the name and description of an Item Folder, view its source design objects, edit the items it contains, and specify or edit any joins or conditions.

To edit an Item Folder:

1. From the Projects Navigator, expand the **Oracle Discoverer** module node, then expand the **Item Folders** node.
2. Right-click the Item Folder name and select **Open**. Or double-click the Item Folder name. This displays the Item Folder editors.
3. Click each of the editors to edit the Item Folder using the guidelines described in the subsequent sections.

9.1.3.1 Name Editor

The Name editor enables you to edit the name and description for the Item Folder.

9.1.3.2 Source Items Editor

The Source Items editor displays the available source items for your Item Folder.

When you are editing an existing item folder, the Selected column displays the source items that were selected at the time of creating the Item Folder. To select different items as the source, use the left arrow to return the items from the Selected column to the Available column. Then use the right arrow to move the new source item from the Available column to the Selected column.

Your selected objects can contain related items from multiple Item Folders.

To change the selected items, then use the left arrow to return the previously selected items. Now select an initial folder item from any of the available Item Folders within the same Oracle Discoverer module. You can then select additional folder items that

have a relationship with the previously selected item. You cannot select items from unrelated Item Folders. The relationship between Item Folders are defined by the joins between them. To create a join between Item Folders, use the Joins editor to specify the relationships between the two Items Folders.

9.1.3.2.1 Deleting Items You can also delete items using the Source Items editor. To delete an item using the Source Items editor, select and move the item from the Selected section to the Available section.

9.1.3.3 Items Editor

The Items editor displays the details and properties of all items in an Item Folder. You can view, create, and edit the following for an item:

9.1.3.3.1 Item Details

- **Name:** Represents the name of an item. To change the current item, double-click the name and retype the new name.
- **Description:** Optionally enter a description for this item.

If you select an item name, the property inspector displays the following properties for that item:

- **Alignment:** The default alignment used for this item in a Discoverer report.
- **Business Name**: Business name of item.
- **Case Storage:** Select the case storage method.
- **Content Type:** Describes the content of multimedia data in this item when used in drilling. If the column contains file names, set this property to FILE. Else set it to the file extension (`avi`,`wav`,`jpg`) to define the application that should process the data.
- **Created By**: Created by.
- **Creation Time**: Time of creation.
- **Database Column**: Specifies the schema, table, and column based on which the item was created or derived.
- **Datatype:** Select the data type for the item. All the data types are supported by Discoverer.
- **Default Aggregate:** Indicate if the item defaults to an aggregate in the Discoverer report.
- **Default Position:** Select the position of this Item on a Discoverer report.
- **Default Width:** The default width of the item when it is displayed in a Discoverer report. The width is in characters.
- **Display Case:** Select in what case the item information is displayed in a Discoverer report.
- **Format Mask:** The format mask for this item when it is used in a work sheet.
- **Formula**: You can provide a formula for any calculated items you want to specify. Click the **Ellipsis** button in this field to open the Formula dialog box. This dialog box contains a subset of the options in the Expression Builder. Use the Formula dialog box to create your calculation. This field is populated after you close the Formula dialog box. For more information about the Expression Builder, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

- **Heading:** The title for the item in a Discoverer report.
- **Item Class:** Assign an Item Class that enables you to define properties for the item. The **Item Class** list contains Lists of Values, Alternative Sort Orders, and Drills to Detail. You can also remove a reference to an Item Class.
- **Last Update Time**: Last updated time.
- **Max Char Fetched:** The maximum amount of data that is fetched from `LONG`, `LONG RAW`, and `BLOB` data types.
- **Replace NULL With:** The value to use instead of the Item value if the value is NULL.
- **Updated By**: Updated by.
- **Visible**: Specifies whether the item is visible to a Discoverer user.
- **Word wrap:** The default word wrap setting used for this item in a Discoverer report.

9.1.3.3.2 Adding Items by Using the Items Editor

To add an item by using the Items editor:

1. Double-click the Item Folder in the Projects Navigator. The item folder editor is displayed. Click **Items** to open the Item Details section.
2. In the Item Details, add items by entering the names of the items.
3. Click **Composition** to open the composition editor.

   The editor displays the item folder and the source object for the item folder. The editor associates an item with a column or another item.
4. For all the items that you added, map the appropriate element from the source object to the item on the item folder.

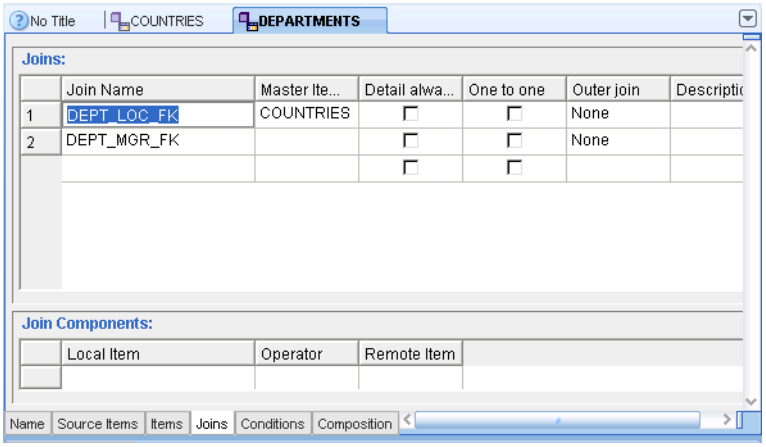
9.1.3.4 Joins Editor

Joins enable you to associate data between two Item Folders. During data analysis, you may require information that resides in multiple folders. Joins enable users to perform business analysis and run reports across multiple Item Folders. After you create joins between Item Folders and deploy them to your Discoverer EUL, they are available for analysis in Discoverer Plus and Discoverer Viewer.

The Joins editor displays the relationships or joins between two Item Folders. You can define new joins by clicking on a new row and providing the required information. You can delete a join by right-clicking the box at the left of each join row that specifies the join's number and selecting **Delete.**

Figure 9–1 shows the contents of the Joins editor.

Figure 9–1 Creating and Editing Joins



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

- **Join Name:** Enter a name for the join you are creating.
- **Master Item Folder:** Select the Item Folder that is the Master. In the above example, DEPARTMENTS is the local item folder and COUNTRIES is the master item folder. A join is created between these two item folders.
- **Detail always has Master:** Select this to indicate if your detail Item Folder always have this master.
- **One to one:** Select this to indicate a one-to-one relationship between the two Item Folders.
- **Outer join:** Indicate from the list if there is an outer join in this relationship and its type.
- **Description:** Optionally describe the join.

For each join, you can specify the Join Components by clicking in the field below and providing the following information:

- **Local Item:** This list is populated with the items contained in the current Item Folder. Select an item from this list.
- **Operator:** Select the relationship between the Local Item you selected and the Remote Item you select from the Master Item Folder.
- **Remote Item:** Select an Item from your Master Item folder to join with the Local Item from your current Item Folder.

If you select a Join name, the property inspector displays the following values for the join:

- **Business Name**: Business name of the Join.
- **Description**: Description of the Join.
- **External Foreign Key**: Specifies the external foreign key for the join.
- **Physical Name**: Provide a physical name that is different from the default.

Also see "Adding Joins Using the Graphical Navigator" on page 9-12 for an alternate way to add joins.

9.1.3.5 Conditions Editor

The Conditions editor enables you to define or edit a condition that restricts selection on the chosen Item Folder. Use this editor to provide or edit a condition. This editor contains the following:

- **Condition Name:** The name of the condition.
- **Condition:** Click the **Ellipsis** button in this field to display the Expression Builder. Use this to create or edit a condition. For more information about the Expression Builder, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.
- **Description:** Optionally describe the condition.
- **Mandatory:** Select this to specify if the condition is mandatory. A mandatory condition is always applied to filter data that is being retrieved for this item folder. Non-mandatory conditions can be switched on and off by the user.
- **Case Sensitive:** Specifies whether the case should match for character data types.

9.1.3.6 Composition Editor

The Composition editor enables you to view the components of an Item Folder. It displays the objects that contribute to the composition of this item folder. These are database objects for Simple Item Folders and contributing Item Folders for Complex Item Folders.

9.1.3.6.1 Adding Items Through the Composition Editor To add an item through the Composition editor:

1. Double-click the Item Folder in the Projects Navigator. The item folder editor is displayed. Click **Composition**.

   The editor canvas displays the item folder and the source object from which the item folder was created. If no source object was selected while creating the item folder, then the canvas displays only the item folder.

2. Right-click anywhere on the canvas, and select **Add**, and then select the type of object (for example, Table, View, or External Table).
3. The Add a New or Existing Object dialog box is displayed. Select the object you want to reference and click **OK**.

   The selected object is now visible on the canvas.

4. Drag the referenced element to an item in the item folder to create the composition for that item. For each column, an item is added in the item folder.

9.1.3.6.2 Deleting Items To delete an item using the Composition editor, right-click the item and select **Delete**.

9.1.3.7 General Properties of an Item Folder

Whenever an Item Folder is selected, the property inspector displays a list of properties for that Item Folder. These properties include Business Name, Description, External Object (Read Only), Folder Type (Read Only), and Visible.

9.1.4 Creating an Item Folder

When you derive intelligence objects, Item Folders are created as part of the derived business definitions. However, to define a customized Item Folder, you can create an Item Folder using the Create Item Folder Wizard.

Item Folders are Discoverer objects and may be Simple or Complex. Each Item Folder contains items that you can delete or edit, as described in "Editing an Item Folder" on page 9-5.

9.1.4.1 Using the Create Item Folder Wizard

To create an Item Folder using the Create Item Folder Wizard:

1. Expand the Oracle Discoverer module in which you want to create an Item Folder.
2. Right-click **Item Folders** and select **New Item Folder**.

   Oracle Warehouse Builder opens the Create Item Folder Wizard.
3. Follow the wizard steps by clicking **Next.**

9.1.4.2 Naming and Describing the Item Folder

In the Name and Description page, enter a name and optional description for the Item Folder.

Oracle Warehouse Builder distinguishes Simple Item Folders from Complex Item Folders in the same way as Discoverer. A Simple Item Folder is directly based on columns from a single table in an Oracle module and calculated items based on constants or items from that Item Folder. A Complex Item Folder can contain items from multiple Item Folders within the same Oracle Discoverer module, and calculated items.

9.1.4.3 Selecting Source Items

Select items for your Item Folder.

For a Simple Item Folder, you can select exactly one table, view, or external table from any module or user folder in the Oracle module, to be referenced by the Item Folder. Expand the selected object and proceed to select columns within the selected object, to your selected items. You can multi-select these referenced items by pressing the **Ctrl** key and using the right arrow to move them to the list of selected Items.

A Complex Item Folder can contain items from multiple Item Folders within the same Oracle Discoverer module. You can select the initial folder items from Item Folder A within an Oracle Discoverer module. You can then select additional folder items from another Item Folder B within the same module. However, the two Item Folders A and B must be related. You cannot select items from unrelated Item Folders. Thus, complex Item Folders combine multiple Item Folders that must be joined. You can define the joins using the Graphical Editor for an Item Folder. For more information about creating joins, see "Joins Editor" on page 9-7.

9.1.4.4 Selecting the Join

For a Complex Item Folder, if there are multiple join paths between the item folders selected as the item sources, then the Join Selection page is displayed. The list on this page displays all the joins between the item folders. Select the join to be used for the Complex Item Folder being created.

9.1.4.5 Reviewing the Summary

In the Summary page, review the name and type of your Item Folder and items to be included in your Item Folder. Click **Back** to make any changes or click **Finish** to create the Item Folder.

You can locate the Item Folder on the Projects Navigator under the Item Folders node in the Oracle Discoverer module. This Item Folder contains all the selected items. You can edit the Item Folder properties, create joins and conditions, and edit item properties using the Graphical Editor, as described in "Editing an Item Folder" on page 9-5.

After creating the Item Folder, you can use the following editors from the graphical editor:

- "Name Editor"
- "Source Items Editor"
- "Items Editor"
- "Joins Editor"
- "Conditions Editor"
- "Composition Editor"

For more information about how you specify the details on each editor, refer to the description of these editors in the "Editing an Item Folder" section on page 9-5.

9.1.4.6 Using the Graphical Navigator to Create Item Folders

An alternate way of creating item folders is to use the Graphical Navigator. You can use the editor menu or the editor canvas of the Graphical Navigator to create an item folder.

9.1.4.6.1 Using the Menu To create an item folder using the menu, click anywhere on the Graphical Navigator. The main menu on the toolbar changes dynamically. From the main menu, select **Graph**, **Add**, **Oracle Discoverer**, **Item Folder**. The Add a New or Existing Item Folder dialog box is displayed. Follow the steps listed in "Steps to Create an Item Folder" on page 9-11.

9.1.4.6.2 Using the Canvas To create an Item Folder using the canvas, drag and drop an Item Folder icon from the Component Palette onto the canvas. Or right-click a blank area on the canvas and from the shortcut menu, select **Add**, **Oracle Discoverer**, **Item Folder**. The Add a New or Existing Item Folder dialog box is displayed. Follow the steps listed in "Steps to Create an Item Folder" on page 9-11.

9.1.4.6.3 Steps to Create an Item Folder Use the following steps to create an Item Folder:

1. Select the **Create a New Item Folder** option.
2. In the **New Item Folder Name** field, specify a name for the Item Folder.
3. In the Oracle Module list, select the name of the Oracle Discoverer module to which the Item Folder should belong.
4. Click **OK**.

The Item Folder is added to the editor canvas. Double-click the item folder to open the editors for the item folder. For more information about the contents of these editors, see "Editing an Item Folder" on page 9-5.

9.1.4.6.4 Adding Items to an Item Folder After you add an Item Folder to the Graphical Navigator, you can add individual items to the Item Folder. On the Graphical Navigator, right-click **Items** and select **Add a Folder Item**. On the Add Folder Item dialog box provide a name for the new item.

9.1.4.7 Adding Joins Using the Graphical Navigator

An alternate way to create a Join is to use the Graphical Navigator.

To add a Join:

1. Ensure that the item folder is available on the Graphical Navigator.
2. Right-click Joins, and select **Add a Join**.

   The Add Join dialog box is displayed.
3. Specify a name for the join, and click **OK**.
4. The Joins editor completes the definition of the Join or use the graphical navigator for this.

Similarly, you can also drag an Item from the Items node to the Joins node to create a local item. You can also drag an Item to the Joins node of another Item folder to create a remote item.

9.1.4.8 Synchronizing Item Folders

Simple Item Folders are defined based on existing tables, views, or external tables. When the definition of the underlying object changes, you can update the Item Folder definition by synchronizing it with the object on which it is based.

To synchronize an Item Folder:

1. Expand the Item Folders node in the Projects Navigator.
2. Right-click the Item Folder and select **Open**.

   The editors for the Item Folder are displayed.
3. Click the Composition editor to view the Item Folder.
4. On the canvas, right-click the Item Folder and select **Synchronize**.

   The Synchronize Item Folder dialog box is displayed.
5. Review the details displayed on this dialog box and click **OK**.

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

9.1.4.9 Synchronize Item Folder Dialog Box

The Synchronize Item Folder dialog box enables you to update the Item Folder with any changes made to the data types used in the database object on which the Item Folder is based. This dialog box displays the details of the changes to be made to the Item Folder.

The Synchronize Item Folder dialog box contains three columns: Object, Reason, and Action. The Object column lists the component in the underlying database object that has changed. The Reason column displays a brief description of the reason for the synchronization. The Action column displays the action that is taken to synchronize the Item Folder. The available actions are Update and None. If you select None for a component, no synchronization is performed for that object. Only definitions that have an Action set to Update are synchronized.

For example, the Item Folder `DEPT_ITMF` is derived from the `DEPT` table. After the Item Folder is created, you modify the `DEPT` table and change the data type of the column `LOCATION` from `VARCHAR2` to `NUMBER`. When you synchronize the Item Folder `DEPT_ITMF`, the Synchronize Item Folder dialog box displays `LOCATION` in the Object column. The Reason column displays "Datatype mismatch". The Action column displays Update.

Click **OK** to perform the actions listed on the Synchronize Item Folder dialog box and update the Item Folder definition. If you do not want to perform the actions listed on this dialog box, click **Cancel**.

9.1.5 Creating a Business Area

Oracle Warehouse Builder enables you to create a Business Area to deploy to a Discoverer EUL. Business Areas contain references to Item Folders stored in your Oracle Discoverer module and are used to group information about a common subject, for example, Sales Analysis, Human Resources, or Stock Control. The Discoverer users use these Business Areas as their starting point for building a query.

Business Areas only contain references to Item Folders and not the actual Item Folder definitions. Thus, a Business Area can contain a collection of unrelated Item Folders and the same Item Folder can appear in multiple Business Areas. It enables you to set up multiple Business Areas with different levels of detail, for example, Sales Analysis area containing one Item Folder, Sales Details area containing six Item Folders, and a Sales Transaction area with 30 Item Folders. When you delete an Item Folder, the reference to it from the Business Area is also deleted.

When you deploy a Business Area using the Control Center, the dependencies of the Business Area are not automatically deployed. For example, if a Business Area BUSN_AREA contains two Item Folders, IF1 and IF2, then when you deploy BUSN_AREA using the Control Center, IF1 and IF2 are not deployed.

You can create a Business Area using the Create Business Area Wizard or from the Graphical Navigator. You also use the editor to edit a business area.

9.1.5.1 Using the Create Business Area Wizard

To create a Business Area using the Create Business Area Wizard:

1. Expand an Oracle Discoverer module.
2. Right-click **Business Areas** and select **New Business Area**.

   Oracle Warehouse Builder opens the Create Business Area Wizard.
3. Follow the wizard steps by clicking **Next.**

9.1.5.1.1 Naming the Business Area In the Name and Description page, enter a name and optional description for the Business Area.

9.1.5.1.2 Selecting the Item Folders In the Source page, all the Item Folders available within the Oracle Discoverer module are displayed. You can multi-select the Item Folders by pressing the Ctrl key and using the right arrow to move them to the list of Selected Item Folders.

9.1.5.1.3 Reviewing the Summary In the summary page, review the Item Folders you selected. Click **Back** to make any changes or click **Finish** to finish creating the Business Area.

After the Business Area is created, you can locate it on the Projects Navigator under the Business Areas node with references to the selected Item Folders stored in it.

To make changes to your Business Area definitions after you create them, use the Edit Business Area dialog box. For details, see "Editing a Business Area" on page 9-14.

9.1.5.2 Using the Graphical Navigator

Alternatively, from the Graphical Navigator you can use the main menu or the canvas to create a business area.

9.1.5.2.1 Using the Menu To create a business area using the menu, click anywhere on the Graphical Navigator. The main menu on the toolbar changes dynamically. From the main menu, select **Graph**, **Add**, **Oracle Discoverer**, **Business Area**. The Add a New or Existing Business Area dialog box is displayed. Select **Create a new Business Area** and specify the name of the business area and the module to which it belongs. Click **OK**. The newly created business area is available on the Project Navigator. Double-click the business area to open the editors for the business area. See "Editing a Business Area" on page 9-14 for details of the editors.

9.1.5.2.2 Using the Canvas To create a business area from the Graphical Navigator, right-click a blank area on the editor canvas and select **Add**, **Oracle Discoverer**, **Business Area**. The Add a New or Existing Business Area dialog box is displayed. Select **Create a new Business Area** and specify the name of the business area and the module to which it belongs. Click **OK**. The newly created business area is now available on the Project Navigator. Double-click the business area to open the editors for the business area. Using the Source editor, move item folders from the Available list to the Selected list. On the Contents editor, you can view existing item folders and add new ones to the business area. See "Editing a Business Area" on page 9-14 for details of the editors.

9.1.6 Editing a Business Area

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

To edit a Business Area:

1. From the Projects Navigator, expand the Business Area node.
2. Right-click a Business Area name and select **Open**.

   Oracle Warehouse Builder opens the Edit Business Area dialog box containing the following editors: Name, Source, and Contents:

9.1.6.1 Name Editor

The Name editor enables you to edit the name and description of a Business Area.

9.1.6.2 Source Editor

The source editor displays the source of those Item Folders that have been included in the Business Area. You can add new item folders to the Business Area or remove existing ones using this editor.

9.1.6.3 Contents Editor

The contents editor enables you to add item folders to a Business Area. Right-click anywhere on the editor and select **Add**, **Item Folder**. The Add a New or Existing Item Folder dialog box is displayed. Follow the steps listed in "Steps to Create an Item Folder" on page 9-11. You can also add an item folder by dragging and dropping the item folder from the Projects Navigator into the Contents editor. This creates a shortcut to the item folder. You can double-click the item folder to access it directly from the Contents editor.

9.1.7 Creating a Drill Path

Oracle Warehouse Builder enables you to create a Drill Path to deploy to a Discoverer EUL. Drill Paths define a hierarchy relationship between the items in your Oracle Discoverer module. For example, Region, Sub-region, Country, and State. Oracle Warehouse Builder creates these drill paths for derived dimensions. You can also create your own customized drill path definitions if you are familiar with your data.

To create a Drill Path:

1. Expand the Oracle Discoverer module.
2. Right-click **Drill Paths** and select **New Drill Path**.

   Oracle Warehouse Builder opens the Create Drill Path Wizard.
3. Follow the wizard steps by clicking **Next.**

9.1.7.1 Naming the Drill Path

In the Name and Description page, enter a name and optional description for the Drill Path.

9.1.7.2 Specifying Drill Levels

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

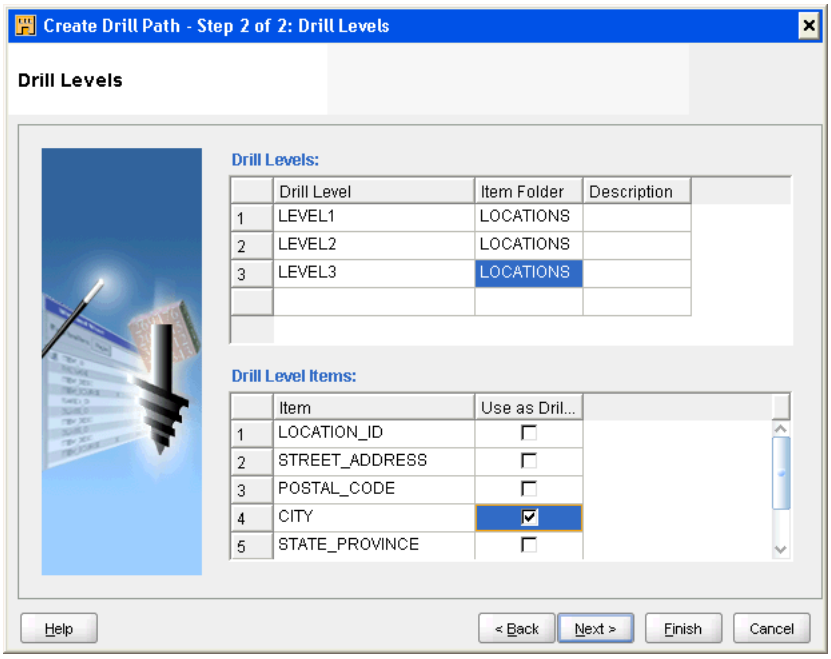
- **Drill Level:** Enter a name for the drill level.
- **Item Folder:** From the field, select the Item Folder it references.
- **Description:** Provide an optional description for the drill level.

When you select a referencing Item Folder for the Drill Level, the wizard lists the available Items within that Item Folder under the Drill Level Items field at the bottom.

In this field, you can specify one or more items to act as drill items. Select the **Use as Drill Item** option for each Item you want to include as a drill item in the level.

Figure 9–2 displays the Drill Levels page of the Create Drill Path Wizard.

Figure 9–2 Creating Drill Levels



9.1.7.3 Specifying the Join

If there are multiple join paths between the Item Folders referenced by the drill levels, then the Join Selection page is displayed. The list displays the existing joins between the selected Item Folder. Select the join to use for the drill path.

9.1.7.4 Reviewing the Summary

In the summary page, review the drill levels you are creating. Click **Back** to make any changes or click **Finish** to create the drill path.

You can locate the drill path on the Projects Navigator under your Oracle Discoverer module. Oracle Warehouse Builder enables you to edit a drill path using the Edit Drill Path dialog box.

9.1.8 Editing a Drill Path

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

To edit a drill path:

1. From the Projects Navigator, expand the Drill Paths node.
2. Right-click the Drill Path and select **Open**.

   Oracle Warehouse Builder displays the Name and Levels editors.

9.1.8.1 Editing the Drill Path Name

The Name editor enables you to edit the name and the description of the drill path.

9.1.8.2 Reviewing the Drill Levels in the Drill Path

Use the Levels editor to edit the drill levels that you defined. The **Drill Levels** section lists the drill levels along with the item folders that they reference. The Item Folder

column displays the item folder that a drill path references. You can modify this by selecting the new item folder from the list.

The **Drill Level Items** section displays the items that act as drill items. You can modify this list by selecting more items that act as drill items.

You use the Structure panel to manipulate hierarchies. For example drill levels can be moved up in the same subtree, moved out of the current tree, or moved to the `root` level. However, multiple roots for Discoverer modules are not enabled and subsequently fail validation. You can also remove a level. In this case, sub levels of deleted levels move up to the `root` level.

9.1.9 Creating Lists of Values

In Discoverer, Lists of Values (LOVs) represents a set of valid values for an item. These are the values in the database column on which the item is based. LOVs enable end users to easily set conditions and parameter values for reports. An example of an LOV can be names of different countries that a user can pick from a list to view a report on the quantities of a product sold in four specific countries.

You can create lists of values for Item Folders using the Create List of Values Wizard as described below.

To create a List of Values:

1. Expand the Oracle Discoverer module.
2. Right-click **Lists of Values** and select **New List of Values**.

   Oracle Warehouse Builder opens the Create List of Values Wizard.
3. Follow the wizard steps by clicking **Next.**

9.1.9.1 Naming the List of Values

In the Name and Description page, enter a name and optional description for this list of values. Select the **Set as Drill to Detail** box if you also want to set this as a Drill to Detail. When you deploy these definitions to Discoverer, an Item Class that you can use both as a List of Values and as a Drill to Detail is created.

9.1.9.2 Defining Items in a List of Values

The Defining Items page enables you to select the item that generates your LOV in Discoverer. This page displays all the Items available in your Discoverer module. Expand the nodes to select an item and click **Next.**

9.1.9.3 Referencing Items in a List of Values

The Referencing Item page enables you to associate your LOV with different items. The Available Items column displays all the Items available in your Discoverer module. Expand the nodes to select the items that references your list of values. Use the right arrow to move your selections to the Selected Items column and click **Next.**

9.1.9.4 Reviewing the Summary

In the summary page, review the defining and referencing items selected for the LOV. Click **Back** to make any changes or click **Finish** to finish creating the LOV.

You can locate the LOV on the Projects Navigator in the Oracle Discoverer module under the Lists of Values node. Oracle Warehouse Builder enables you to edit the name, description, and defining and referencing items associated with an LOV using the Edit List of Values dialog box.

9.1.10 Editing Lists of Values

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

To edit a list of values:

1. From the Projects Navigator, expand the List of Values node.
2. Right-click the List of Values and select **Open**.

   Oracle Warehouse Builder displays the Edit List of Values dialog box, where you can edit the following: Name, Defining Item, Referencing Items, and Options.

9.1.10.1 Editing the Name

Use the Name editor to edit the name and description of the list of values.

9.1.10.2 Editing the Defining Items

Use the Defining Item editor to edit the item that generates the list of values in Discoverer. The item that is the defining item is highlighted. To edit this and specify that another item is necessary to generate the LOV, select the new item.

9.1.10.3 Editing the Referencing Items

Use the Referencing Items editor to edit the items that reference the list of values. The Selected column lists the items that the list of values references. To add more items to which the list of values references, select the item in the Available column and use the right arrow to move it to the Selected column. To remove items that the list of values currently references, select the item from the Selected column and use the left arrow to move it to the Available column.

9.1.10.4 Editing the Options

Use the Advanced editor to specify advanced options for the list of values. The advanced options are as follows:

- **Retrieve Values in groups of**: Use this option to specify the number of values that are retrieved in group. The default value is 100 which means that the values are retrieved in groups of 100.
- **Sort the values and remove duplicates**: Select this option to remove duplicate values from the list of values and to order the values. This ensures that the LOV always shows unique, ordered values.
- **Show values in "Select Items" page of Worksheet Wizard**: Select this option to enable users to expand the List of Values when selecting items to include in a query.
- **Require user to always search for values**: Select this option to display the Search dialog box every time the List of Values is expanded.
- **Cache List of Values during each connection**: Select this option to store the list of values when the List of Values is expanded for the first time. This improves performance because otherwise, every time the List of Values is expanded, the values are fetched from the database.

9.1.11 Creating Alternative Sort Orders

In Discoverer, alternate sorts enable end users to display values in a nonstandard sequence.

For example, by default the values of the Description item are sorted alphabetically. To sort the description according to the values of the Product Key item, you must define an alternate sort item and link the two items. One item defines the sort order and the other defines the item to be sorted.

Define how you want to order the information in your Discoverer EUL using the Create Alternative Sort Order Wizard.

To create an Alternative Sort:

1. Expand the Oracle Discoverer module.
2. Right-click **Alternative Sort Orders** and select **New Alternative Sort Order.**

   Oracle Warehouse Builder opens the Create Alternative Sort Order Wizard.
3. Follow the wizard steps by clicking **Next.**

9.1.11.1 Naming the Alternative Sort Order

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

Select the **Set as Drill to Detail** box if you also want to set this as a Drill to Detail. When you deploy these definitions to Discoverer, an Item Class that can be used both as an Alternative Sort Order and as a Drill to Detail is created.

9.1.11.2 Defining Item for the Alternative Sort Order

The Defining Item page enables you to select the Item that contains the values to be sorted. Expand the nodes to select an item and click **Next.**

9.1.11.3 Defining Order Item for the Alternative Sort Order

Use the Defining Order Item page to select an Item, in the same Item Folder, that defines the order in which the values of the Item you selected on the Defining Item page are displayed. Expand the nodes to select the item and click **Next.**

9.1.11.4 Referencing Items for the Alternative Sort Order

The Referencing Items page enables you to associate your Alternative Sort Order with different items. The Available column lists all the Items in the Discoverer module. Expand the nodes to select the items that references your Alternative Sort Order. Use the right arrow to move your selections to the Selected column and click **Next.**

9.1.11.5 Referencing Selection Panel for the Alternative Sort Order

This panel enables you to shuttle across an item that references an item class. You can either change the reference or decide not to shuttle the item across.

9.1.11.6 Reviewing the Summary

In the summary page, review the alternative sort order definition. Click **Back** if to make any changes or click **Finish** to finish creating the alternative sort order.

You can locate the alternative sort order on the Projects Navigator in the Oracle Discoverer module under the Alternative Sort Order node. Oracle Warehouse Builder enables you to edit the name, description, and the defining and referencing items associated with an alternative sort order using the Edit dialog box.

9.1.12 Editing Alternative Sort Orders

The Edit Alternative Sort Order dialog box enables you to edit an alternative sort order.

To edit an alternative sort order:

1. Expand the Alternative Sort Order node in the Projects Navigator.
2. Right-click the Alternative Sort Order and select **Open**.

   The Edit Alternative Sort Order dialog box containing the following tabs is displayed: Name, Defining Item, Defining Order Item, Referencing Items, and Options.

9.1.12.1 Editing the Alternative Sort Order Name

Use the Name tab to edit the name and description of the alternative sort order.

9.1.12.2 Editing the Defining Item

Use the Defining Item tab to edit the item that contains the values to be sorted. This editor displays the Item that currently defines the alternative sort order highlighted. To change this selection, click the item that you now want to use to define the alterative sort order.

9.1.12.3 Editing the Defining Order Item

The Defining Order Item tab displays the Item Folder with the item that currently defines the order in which the values of the Item selected on the Defining Item editor are displayed. You can change this selection by clicking a new item from the tree.

9.1.12.4 Editing the Referencing Items

The Referencing Items tab lists the items that references your Alternative Sort Order in the **Selected** column. To add more items to this list, select the item in the **Available** column and use the right arrow to move the item to the Selected column. To remove an item that is selected, move the item from the Selected column to the Available column using the left arrow.

9.1.12.5 Editing the Options

Use the Options tab to specify advanced options for the alternative sort order. The options you can set are as follows:

- **Retrieve values in groups of:** Use this option to specify the number of values that are retrieved in group. The default value is 100 which means that the values are retrieved in groups of 100.
- **Sort the values and remove duplicates:** Select this option to remove duplicate values from the alternative sort order and to order the values. This ensures that the alternative sort order always shows unique, ordered values.
- **Show values in "Select Items" page of the Worksheet Wizard:** Select this option to enable users to expand the alternative sort order when selecting items to include in a query.
- **Require user to always search for values:** Select this option to display the Search dialog box every time the Alternative Sort Order is expanded.
- **Cache list of values during each connection:** Select this option to store the Alternative Sort Order when it is expanded for the first time. This improves

performance because otherwise, every time the Alternative Sort Order is expanded, the values are fetched from the database.

9.1.13 Creating Drills to Detail

In Discoverer, drills to detail enable you to analyze your data thoroughly by navigating through your data and performing drill down operations to obtain detailed information. When you define drills to detail, you define relationships between items. These drills enable you to interactively drill up or down through your data to see a different level of detail. For example, you can move from actuals to budgets for the same department, then look at the department employee details, then drill down to their salary and training histories, then drill to their job grades structure, and so on.

You can define a drill to detail using the Create Drill to Detail dialog box.

To create a Drill to Detail:

1. Expand the Oracle Discoverer module.
2. Right-click **Drills to Detail** and select **New Drill to Detail.**

   Oracle Warehouse Builder opens the Create Drill to Detail dialog box.

9.1.13.1 Create Drill to Detail Dialog Box

Name: Enter a name for the drill to detail definition.

Description: Provide an optional description for the drill to detail.

The **Available** column at the bottom of the dialog box lists the Item Folders in the Oracle Discoverer module. Select a referencing item from this set and use the right arrow to move it to the **Selected** column.

9.1.14 Editing Drills to Detail

Use the Edit Drill to Detail dialog box to edit a Drills to Detail.

To edit a Drills to Detail:

1. Expand the Drills to Detail node in the Projects Navigator.
2. Right-click the name of the Drill to Detail and select **Open**.

   The Edit Drill to Detail dialog box is displayed. The contents of this dialog box are the same as the Create Drill to Detail dialog box. In addition to modifying the name and description of the drill to detail, you can edit the referencing items. for more details on the contents of the Drill to Detail dialog box, see "Create Drill to Detail Dialog Box" on page 9-21.

9.1.15 Creating Registered Functions

In Discoverer, you can use custom PL/SQL functions to perform operations or calculations on values in an Item. To access these functions in Discoverer, the user-defined functions are registered in the EUL. To use any of those registered user-defined functions in Discoverer, you must include that information in your object definitions.

You can define a registered function using the Create Registered Function Wizard as described below.

To create a Registered Function:

1. Expand the Oracle Discoverer module.

2. Right-click **Registered Function** and select **New Registered Function**.

   Oracle Warehouse Builder opens the Create Registered Function Wizard.

3. Follow the wizard steps using the guidelines below.

9.1.15.1 Naming the Registered Function

In the Name and Description page, enter a name and optional description for the registered function.

From the **Select the return type of the function** list, select a return type for the function. Select **Available to User** to indicate if a Discoverer end-user can use this registered function in calculations.

9.1.15.2 Specifying the Function Parameters

Specify the function parameters by clicking on a row and entering a name for the parameter. From the **Data Type** list, select the data type for the parameter. Use the **Description** field to enter an optional description.

9.1.15.3 Reviewing the Summary

In the Summary page, review the registered function definition. Click **Back** to make any changes or click **Finish** to finish creating the registered function.

You can locate the registered function on the Projects Navigator in the Oracle Discoverer module under the Registered Functions node. Oracle Warehouse Builder enables you to edit the name, description, and parameters of the registered function using the Edit dialog box.

9.1.16 Editing Registered Functions

Use the Edit Registered Function dialog box to edit a registered function.

To edit a registered function:

1. Expand the Registered Functions node in the Projects Navigator.
2. Right-click the registered function and select **Open**.

   The Edit Registered Function dialog box containing the following tabs is displayed: Name and Parameters.

9.1.16.1 Renaming a Registered Function

Use the Name tab to edit the name and the description of the registered function.

9.1.16.2 Modifying the Parameters of a Registered Function

Use the Parameters tab to edit the parameters of the registered function. You can edit the name, type, and description of a parameter. Add new parameters by clicking on an empty row and specifying the name of the parameter and its data type. You can move the parameters using the arrow keys.To delete a parameter, right-click the gray cell (which displays the number) to the left of the parameter name and select **Delete**.

9.2 Configuring Discoverer Objects

During the design phase, you create definitions for the BI objects. After you design objects, you can assign physical properties to these design objects by setting configuration parameters.

To configure a BI object, right-click the object in the Projects Navigator and select **Configure**. The Configuration Properties dialog box is displayed. Click the object name on the left side of this dialog box to display the configuration parameters on the right.

All BI objects have a configuration parameter called **Deployable**. Select Deployable to generate scripts and deploy the business object. Oracle Warehouse Builder only generates scripts for objects marked deployable.

The following sections describe additional configuration parameters for different types of BI objects.

9.2.1 Configuration Parameters for Oracle Discoverer modules

You can set the following configuration parameters for an Oracle Discoverer module.

Object Matching: Indicates how object matching during deployment to Discoverer should be performed. When you deploy business definitions, an `.eex` file is first created and then this file is imported into the Discoverer EUL.

The options you can select for Object Matching are **By Identifier** or **By Name**. Oracle Warehouse Builder uses this setting to check if an object similar to one that is being deployed exists in the EUL. If a similar object is found, in Create mode the objects are not deployed and in Upgrade mode the objects are refreshed.

MLS Deployment Language: Represents the language used for deployment to Discoverer.

Location: Represents the Discoverer location to which the Oracle Discoverer module is deployed.

9.2.2 Configuration Parameters for Item Folders

You can set the following configuration parameters for item folders.

Optimizer Hint: Represents the optimizer hint to be added when the item folder is used in a query. Click the **Ellipsis** button on this field to specify the optimizer hint.

Location: Represents the location of the database object that the item folder references.

9.2.3 Configuration Parameters for Registered Functions

For registered functions, you can set the following configuration parameters.

Package: Represents the name of the package that contains the registered function.

Location: Represents the location of the database object that the registered function references.

9.3 Deploying Business Definitions to Oracle BI Discoverer

After you create your business definitions, you can deploy them to Oracle BI Discoverer. The method used to deploy business definitions depends on the version of Oracle BI Discoverer to which the business definitions are being deployed and the licensing option used.

Note: The method of deploying business definitions depends on the Oracle Warehouse Builder licensing option and the version of Oracle BI Discoverer to which you want to deploy business definitions.

You can create a direct or indirect connection to the Oracle BI Discoverer EUL. With a direct connection, you must specify the location of the Oracle BI Discoverer and the business definitions are directly deployed to that location. With an indirect connection, you must specify an `.eex` file to store the business definition. You must also specify the mode of transferring the `.eex` to the Discoverer EUL. See "Defining Discoverer Locations" on page 9-3 for information about providing these connection details.

Table 9–1 summarizes the combinations possible when you deploy business definitions to Oracle BI Discoverer using the different licensing options.

Table 9–1 Different Methods of Deploying Business Definitions

Discoverer Version	Oracle Warehouse Builder Core Functionality	Oracle Warehouse Builder Enterprise ETL Option
Versions Lower than Oracle BI Discoverer 10*g* Release 2	Use Indirect mode of connection. Deploy an `.eex` file, and import the file into Oracle Discoverer. See "Indirectly Deploying Business Definitions to Oracle BI Discoverer" on page 9-25.	Use Indirect mode of connection. Deploy an `.eex` file and import the file into Oracle BI Discoverer. See "Deploying Business Definitions to Earlier Versions of Oracle BI Discoverer" on page 9-25.
Oracle BI Discoverer 10*g* Release 2 and later	Use Indirect mode of connection. Deploy the `.eex` file and import it into Oracle BI Discoverer. See "Indirectly Deploying Business Definitions to Oracle BI Discoverer" on page 9-25.	Use the Direct mode of connection. Use the Control Center to directly deploy to Oracle BI Discoverer. Note however that you can make a direct deployment to Oracle BI Discoverer only if you installed a standalone version of Oracle Warehouse Builder 11*g* Release 2 (11.2). If you are using Oracle Warehouse Builder directly from Oracle Database 11g, then you can use only the indirect mode of connection. See *Oracle Warehouse Builder Installation and Administration Guide* for more information about the different installation options. See "Directly Deploying Business Definitions to Oracle BI Discoverer" on page 9-24.

9.3.1 Directly Deploying Business Definitions to Oracle BI Discoverer

You can directly deploy business definitions to Oracle BI Discoverer, just like you deploy other data objects, using the Control Center or Projects Navigator. See Table 9–1 for information about the scenarios when you can directly deploy to Oracle BI Discoverer.

The business definitions are deployed to the Discoverer location associated with the Business Definition module that contains these business definitions. Before you deploy business definitions, ensure that a valid Discoverer location is associated with the Business Definition module. For information about how to associate a Discoverer location with a Business Definition module, see "Setting the Connection Information" on page 9-2.

When you deploy business definitions directly to Oracle BI Discoverer 10*g* Release 2 and later, the following steps are performed:

1. An `.eex` file that contains the definitions of the business definitions is created.
2. A connection is established to the EUL specified in the Discoverer location.

Note: If the EUL is in a different database from your object definitions, then a database link is created.

3. The `.eex` file is imported into Oracle BI Discoverer.

   During the import, new business definitions are appended on top of the existing definitions. You must validate the EUL and remove redundant definitions. For example, if you deployed an item folder with four items. Subsequently, you deleted one item from the item folder. When you redeploy the item folder, it still contains four items. This is because while new definitions are appended, old definitions are not removed.

9.3.2 Deploying Business Definitions to Earlier Versions of Oracle BI Discoverer

You cannot directly deploy business definitions to versions of Oracle BI Discoverer earlier than 10*g* Release 2. However, you can use the indirect mode of connection to deploy business definitions. See Table 9–1 for more information about the different scenarios when you can make an indirect deployment to Oracle BI Discoverer.

While creating the location of the business definition module, use the Indirect mode, and specify an `.eex` file to store the details of the business definition. When you deploy the business definition, the relevant details are captured in the `.eex` file. You can connect to the EUL using Oracle BI Discoverer and import this `.eex` file.

9.3.3 Indirectly Deploying Business Definitions to Oracle BI Discoverer

When you use the core functionality of Oracle Warehouse Builder, you cannot directly deploy business definitions to Oracle BI Discoverer. Instead you must use the indirect method of deployment to deploy the `.eex` file and then manually import it from Oracle BI Discoverer. You can specify the indirect method of deployment by selecting this option in the connection information page while creating a Discoverer location.

Note: Indirect deployment takes place through the run time service running under a user in the database. This database user must have privileges to write to the destination.

9.4 Accessing BI Objects Using Oracle BI Discoverer

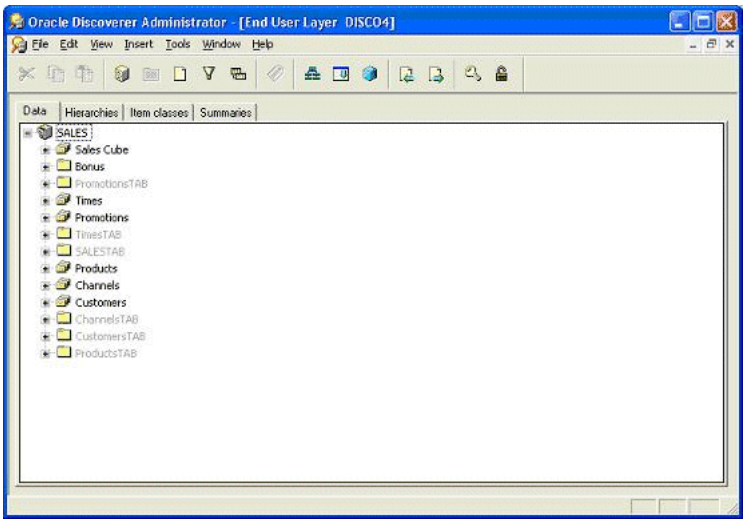
Once you successfully deploy the BI objects that you create, these objects are available in Oracle BI Discoverer. You can use these objects to perform analysis on your warehouse data.

9.4.1 Using Business Definitions in Oracle BI Discoverer

After you deploy the business definitions that you create, these objects are available in the EUL to which they were deployed. Log in to Oracle BI Discoverer Administrator using the user name that you used to deploy the business definitions.

Figure 9–3 displays the Discoverer Administrator interface with the business definitions that you deployed.

Figure 9–3 Discoverer Administrator Showing BI Objects



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

9.5 Creating Business Definitions for OBIEE

You can create business definitions from various database objects and integrate these business definitions with OBIEE. The various database objects that can be derived include tables, dimensions, cubes, views, and MVs. These objects can be derived from Oracle and other non-Oracle databases. The data exported as business definitions into OBIEE facilitates data analysis and report generation, which is then used in decision making by businesses.

9.5.1 Creating an Oracle Business Intelligence Module

You must create an Oracle Business Intelligence (OBI) module to store business definitions before you can deploy them to OBIEE.

To create an Oracle Business Intelligence module:

1. In the Projects Navigator, under Business Intelligence, right-click **Oracle Business Intelligence** and select **New Oracle Business Intelligence**.

   The Create Module wizard is displayed.

2. In the Name and Description page, provide a name and description (optional) for the module. Also specify the type of module.

3. In the Connection Information page, select a location from the Location list. To edit the connection information for the location, click **Edit**. The Edit Oracle BI location dialog box is displayed. See "Defining Oracle Business Intelligence Location" on page 9-27 for details of connection information to be provided.

Once you complete the steps in the wizard, the new Oracle Business Intelligence module is available under the Business Intelligence node. The module consists of the following objects:

- Logical Tables
- Catalog Folders
- Dimension Drill Paths

9.5.1.1 Defining Oracle Business Intelligence Location

The OBI module location refers to a UDML file on the OBIEE server. At the time of specifying the location, specify a UDML file in the OBIEE server as the location. You must also specify the mode of file transfer from the local system to the OBIEE server. Provide the following details in the Edit Oracle BI Location dialog box:

- **Name**: Provide a location name.
- **Description**: Provide an optional description.
- **Transport Type**: Select from FILE, FTP, HTTP, and HTTPS.
- **Version**: Select the OBIEE version.

Depending on the mode of file transfer, you must provide the following details:

FILE

- **Root Path**: Directory of the UDML file.
- **File Name**: The name of the UDML file.

FTP

- **Host Name**: The system credentials where the OBIEE server resides.
- **Host Login Port**: Login port number, which is initially set to 0. You must change this according to your local configuration.
- **Transfer Format**: Select from ASCII and IMAGE.
- **Host Login User**: User name to run FTP.
- **Host Login Password**: User password to run the FTP command.
- **File Name**: The name of the UDML file along with the complete path.

HTTP and HTTPS

- **Host Name**: The system credentials where the OBIEE server resides.
- **Host Login Port**: Login port number, which is initially set to 0. You must change this according to your local configuration.
- **Host Login User**: User name for the HTTP/HTTPS command.
- **Host Login Password**: User password for the HTTP/HTTPS command.
- **File Name**: The name of the UDML file along with the complete path.

9.5.2 About Logical Tables

Logical Tables are equivalent to objects in OBIEE that map to database tables, external tables, or views. They represent a result set of data, similar to a database view. Logical Tables also store information just like tables. A logical table contains items that map to columns in a table. Each item has a name and contains specific type of information. For example, the logical table containing details about employees may include items such as employee name, start date, and department.

Oracle Warehouse Builder creates Logical Tables when you derive business definitions from warehouse design objects in the database modules, as described in "Deriving BI Objects" on page 9-41. You can also manually create a customized Logical Table using the Create Logical Table Wizard or the Graphical Navigator. The Graphical Editor is used to edit Logical Tables.

The following sections contain more information related to Logical Tables:

- "Editing a Logical Table"
- "Creating a Logical Table"

9.5.3 Editing a Logical Table

After you derive your design object definitions, a Logical Table is created as part of the derived business definitions.

Oracle Warehouse Builder provides the document editors that enable you to edit the name and description of a Logical Table, view its source design objects, edit the Items it contains, and specify or edit any joins or conditions.

To edit a Logical Table:

1. From the Projects Navigator, expand the OBI module node, then expand the Logical Tables node.
2. Right-click the Logical Table and select **Open**. Or double-click the Logical Table. This displays the Logical Table editors.
3. Click each of the editors to edit the Logical Table using the guidelines below.

9.5.3.1 Name Editor

The Name editor enables you to edit the name and description for the Logical Table.

9.5.3.2 Source Items Editor

The Source Items editor displays the available source items for the Logical Table.

The Available column displays database tables in the current project and logical tables in the OBI module.

When you are editing an existing logical table, the Selected column displays the source items that were selected at the time of creating the logical table. To select different items as the source, use the left arrow to return the items from the Selected column to the Available column. Then use the right arrow to move the new source item from the Available column to the Selected column.

Your selected objects can contain items from multiple Logical Tables.

To change the selected items, then use the left arrow to return the previously selected items. Now select an initial folder item from any of the available Logical Tables within the same OBI module. You can then select additional folder items with the previously selected item.

9.5.3.2.1 Adding Items Through the Source Item To add an item through the Source Items editor:

1. Double-click the Logical Table in the Projects Navigator. The logical table editor is displayed. Click **Source Items**.

   The Selected section displays the items that are currently included in the logical table. If you drill down the Available section, it displays the items that can be added.
2. Select and move the required items from the Available to the Selected section.

9.5.3.2.2 Deleting Items To delete an item using the Source Items editor, select and move the item from the Selected section to the Available section.

9.5.3.3 Items Editor

Items Editor displays the details and properties of all Items in a Logical Table. You can view, create, and edit the following for an Item:

9.5.3.3.1 Item Details

- **Name**: Represents the name of an Item. To change the current Item, double-click the name and retype the new name.
- **Description**: Optionally enter a description for this Item.

If you select an item name, the property inspector displays the following properties for that item:

- **Business Name**: The business name of the item.
- **Created By**: Created by.
- **Creation Time**: Time of creation.
- **Database Column**: The database column that the item maps to.
- **Datatype**: The data type of the item.
- **Default Aggregate**: The aggregation function for the items.
- **Description**: Description of the item.
- **Formula**: This is the expression for a calculated item.
- **Last Update Time**: Last updated time.
- **Physical Name**: Physical name of the item.
- **Updated By**: Updated by.
- **Visible**: Select this option if the item is to be made visible to OBIEE Report Builder.

9.5.3.3.2 Adding Items Through Items Editor

To add an item through Items Editor:

1. Double-click the Logical Table in the Projects Navigator. The logical table editor is displayed. Click **Items** to open the Item Details section.
2. In the Item Details, add items by entering the names of the items.
3. Click **Composition** to open the composition editor.

   Use the composition editor to define the referencing items or use the formula property to enter the expression for this item.

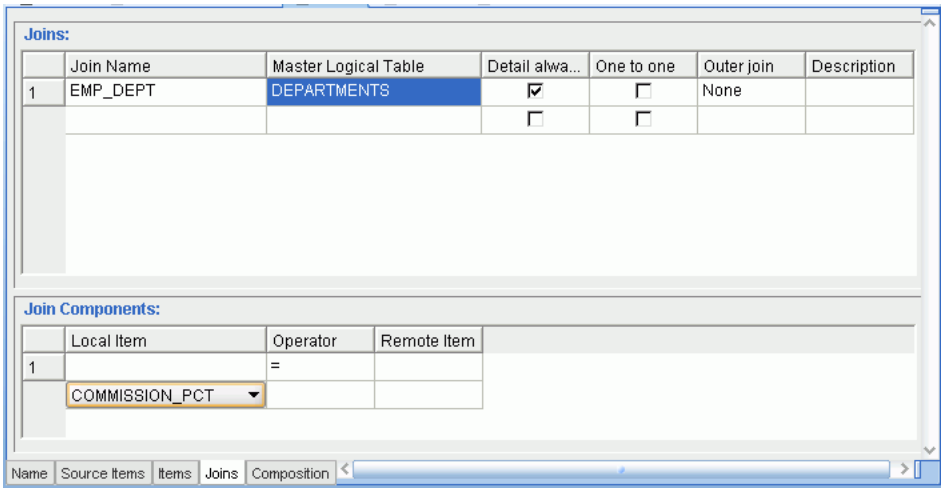
9.5.3.4 Joins Editor

Joins enable you to associate data between two logical tables. During data analysis, you must retrieve information that resides in multiple logical tables. Joins enable end users to perform business analysis and run reports across multiple logical tables. After you create joins between logical tables and deploy them to your OBIEE Repository, they are available for analysis in OBIEE Answers or Dashboards.

The Joins editor displays the relationships or joins between two logical tables. You can define new joins by clicking on a new row and providing the required information. You can delete a join by right-clicking the box on the left of each join row and selecting **Delete.**

Figure 9–4 shows the contents of the Joins editor for the logical table EMP.

Figure 9–4 Creating and Editing Joins



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

- **Join Name**: Enter a name for the join you are creating.
- **Master Logical Table**: Select the Logical Table that is the Master. In the above example, you select the logical table DEPARTMENTS as your master. This implies that you select an item from the DEPARTMENT Logical Table to join with the two items you selected from the Logical Table EMP.
- **Detail always has Master**: Select this box to indicate if your detail Logical Table always have this master.
- **One to one**: Select this box to indicate a one-to-one relationship between the two Logical Tables.
- **Outer join**: Indicate from the list whether there is an outer join in this relationship and its type.
- **Description**: Optionally describe the join.

For each join, you can specify the Join Components by clicking in the field below and providing the following information:

- **Local Item**: This list is populated with the Items contained in the current Item Folder. Select an Item from this list.
- **Operator**: Select the relationship between the Local Item you selected and the Remote Item you select from the Master Logical Table.
- **Remote Item**: Select an Item from your Master Logical Table to join with the Local Item from your local Logical Table.

Also see "Adding Joins Using the Graphical Navigator" on page 9-33 for an alternate method to add joins.

9.5.3.4.1 Properties of a Join Double-click a Join name to view the following properties of the join:

- **Business Name**: The business name of the item.
- **Created By**: Created by.
- **Creation Time**: Time of creation.
- **Description**: Description provided at the time of creation.

- **External Foreign Key**: If the join was derived from a foreign key, then that foreign key is listed here.
- **Last Update Time**: Last updated time.
- **Physical Name**: Physical name of the join.
- **Updated By**: Updated by.

9.5.3.5 Composition Editor

The Composition editor enables you to view the components of a Logical Table. It displays the Logical Table and the objects from which it is referencing.

9.5.3.5.1 Adding Items Through the Composition Editor To add an item through the Composition editor:

1. Double-click the Logical Table in the Projects Navigator. The logical table editor is displayed. Click **Composition**.

   The editor canvas displays the logical table and the source objects from which the logical table was created. If no source object was selected while creating the logical table, then the canvas displays only the logical table.

2. Right-click anywhere on the canvas, and select **Add**, and then select the type of object (Table, View, Materialized View, Logical Table).
3. The Add a New or Existing Object dialog box is displayed. Select the object from which you want to add the items and click **OK**.

   The selected object is now visible on the canvas.

4. Map the required columns or items from the object to the logical table.

9.5.3.5.2 Deleting Items To delete an item using the Composition editor, right-click the item and select **Delete**.

9.5.4 Creating a Logical Table

When you derive intelligence objects, Logical Tables are created as part of the derived business definitions. However, to define a customized Logical Table, you can create a Logical Table using the Create Logical Table Wizard.

Each logical table contains items that you can delete or edit, as described in "Editing a Logical Table" on page 9-28.

9.5.4.1 Using the Create Logical Table Wizard

To create a Logical Table using the Create Logical Table Wizard:

1. Expand the OBI module in which you want to create a logical table.
2. Right-click **Logical Tables** and select **New Logical Table**.

   Oracle Warehouse Builder opens the Create Logical Table Wizard.

3. Follow the wizard steps by clicking **Next.**

9.5.4.2 Naming and Describing the Logical Table

In the Name and Description page, enter a name and optional description for the Logical Table.

9.5.4.3 Selecting Source Items

You can select from tables, views, or materialized views from any of the database modules, to be referenced by the Logical Table. If you select multiple tables, then these must be joined by a foreign key. Expand the selected object and proceed to select columns within the selected object, to your selected items. You can multi-select these referenced items by pressing the **Ctrl** key and using the right arrow to move them to the list of selected Items.

You can also select from other Logical tables within the same OBI module. You can select the initial items from Logical Table A within an OBI module. You can then select additional folder items from another Item Folder B within the same module.

9.5.4.4 Selecting the Foreign Key

If there are multiple foreign keys between the tables selected as the item sources, the Foreign Key Selection page is displayed. The list on this page displays all the joins between the tables. Select the join to be used for the table being created.

9.5.4.5 Reviewing the Summary

In the Summary page, review the details you have provided for the Logical Table including the items to be included in the table. Click **Back** to make any changes or click **Finish** to create the Logical Table.

You can locate the Logical Table on the Projects Navigator under the Logical Tables node in your OBI module. This Logical Table contains all the selected items. You can edit the Logical Table properties, create joins, and edit item properties. The Graphical Navigator can also be used to edit a Logical Table.

After creating the Logical Table, you can use the following editors:

- "Name Editor" on page 9-28
- "Source Items Editor" on page 9-28
- "Items Editor" on page 9-29
- "Joins Editor" on page 9-29
- "Composition Editor" on page 9-31

For more information about how you specify the details on each editor, refer to the description of these editors in the "Editing a Logical Table" on page 9-28.

9.5.4.6 Using the Graphical Navigator to Create Logical Tables

An alternate way of creating logical tables is to use the Graphical Navigator. You can use the editor menu or the editor canvas of the Graphical Navigator to create a logical table.

9.5.4.6.1 Using the Menu To create a logical table using the menu, click anywhere on the Graphical Navigator. The main menu on the toolbar changes dynamically. From the main menu, select **Graph**, **Add**, **Oracle Business Intelligence**, **Logical Table**. The Add a New or Existing Logical Table dialog box is displayed. Follow the steps listed in "Steps to Create a Logical Table" on page 9-33.

9.5.4.6.2 Using the Canvas To create a Logical Table using the canvas, drag and drop a Logical Table icon from the Component Palette onto the canvas.

Or right-click a blank area on the canvas and select **Add**, **Oracle Business Intelligence**, **Logical Table**. The Add a New or Existing Logical Table dialog box is displayed. Follow the steps listed in "Steps to Create a Logical Table" on page 9-33.

9.5.4.6.3 Steps to Create a Logical Table Use the following steps to create a Logical Table:

1. Select the **Create a New Logical Table** option.
2. In the **New Logical Table Name** field, specify a name for the Logical Table.
3. In the OBI Module list, select the name of the OBI module to which the Logical Table should belong.
4. Click **OK**.

The Logical Table is added to the editor canvas. Double-click the logical table to open the editors for the logical table. For more information about the contents of these editors, see "Editing a Logical Table" on page 9-28.

9.5.4.6.4 Adding Items to a Logical Table After you add a Logical Table to the Graphical Navigator, you can add individual items to it. On the Graphical Navigator, right-click Items and select **Add an Item**. On the Add Item dialog box provide a name for the new item.

9.5.4.6.5 Properties of a Logical Table At all times, for any Logical Table, the property inspector displays a list of properties for that Logical Table. You can configure the following:

- **Bridge Table**: Select this option to create a bridge table. A bridge table is an intermediate table between a fact table and a dimension table, and is used to resolve a many-to-many association.
- **Distinct Values**: Select this option if you want only distinct values to be selected from the source physical table.
- **Visible**: Select this option to make the logical visible to OBIEE users.

9.5.4.7 Adding Joins Using the Graphical Navigator

An alternate way to create a Join is to use the Graphical Navigator. To add a Join:

1. Ensure that the logical table is available on the Graphical Navigator.
2. Right-click Joins, and select **Add a Join**.

   The Add Join dialog box is displayed.
3. Specify a name for the join, and click **OK**.
4. Open the Joins editor for the logical table and specify the conditions for the join. See "Joins Editor" on page 9-29 for more details.

Using the Graphical Navigator, you can also create a join between items in two Logical Tables. Ensure that the Logical Tables are visible on the Graphical Navigator. To create a join between two Items, drag an Item from one Logical Table to the required Item in the other Logical Table. Similarly, you can also drag an Item from the Items node to the Joins node to create a local item. You can also drag an Item to the Joins node of another Item folder to create a remote item.

9.5.4.8 Synchronizing Logical Tables

Logical Tables are defined based on existing tables, views, or materialized views. When the definition of the underlying object changes, you can update the Logical Table definition by synchronizing it with the object on which it is based.

To synchronize a Logical Table:

1. Expand the Logical Tables node in the Projects Navigator.
2. Right-click the Logical Table and select **Open**.

   The editors for the Logical Table are displayed.
3. Click the Composition editor to view the Logical Table.
4. On the canvas, right-click the Logical Table and select **Synchronize**.

   The Synchronize Item Folder dialog box is displayed.
5. Review the details displayed on this dialog box and click **OK**.

   Oracle Warehouse Builder synchronizes the Logical Table with the data object on which the item is based.

9.5.4.9 Using the Synchronize Logical Table Dialog Box

The Synchronize Logical Table dialog box enables you to update the Logical Table with any changes made to the data types used in the database object on which the Logical Table is based. This dialog box displays the details of the changes to be made to the Logical Table.

The Synchronize Logical Table dialog box contains three columns: Object, Reason, and Action. The Object column lists the component in the underlying database object that has changed. The Reason column displays a brief description of the reason for the synchronization. The Action column displays the action that is taken to synchronize the Logical Table. The available actions are Update and None. If you select None for a component, no synchronization is performed for that object. Only definitions that have an Action set to Update are synchronized.

For example, the Logical Table `DEPT_ITMF` is derived from the `DEPT` table. After the Logical Table is created, you modify the `DEPT` table and change the data type of the column `LOCATION` from `VARCHAR2` to `NUMBER`. When you synchronize the Logical Table `DEPT_ITMF`, the Synchronize Logical Table dialog box displays `LOCATION` in the Object column. The Reason column displays "Datatype mismatch". The Action column displays Update.

Click **OK** to perform the actions listed on the Synchronize Logical Table dialog box and update the Logical Table definition. If you do not want to perform the actions listed on this dialog box, click **Cancel**.

9.5.5 Creating a Dimension Drill Path

Oracle Warehouse Builder enables you to create a Dimension Drill Path to deploy to an OBIEE repository. Dimension Drill Paths define a hierarchy relationship between the items in your OBI module. For example, Region, Sub-region, Country, State, and so on. Oracle Warehouse Builder creates these dimension drill paths for derived dimensions. You can also create your own customized dimension drill path definitions if you are familiar with your data.

To create a Dimension Drill Path:

1. Expand the Oracle Business Intelligence module.

2. Right-click **Dimension Drill Paths** and select **New Dimension Drill Path**.

   Oracle Warehouse Builder opens the Create Dimension Drill Path Wizard.

3. Follow the wizard steps by clicking **Next.**

9.5.5.1 Naming the Dimension Drill Path

In the Name and Description page, enter a name and optional description for the Dimension Drill Path.

9.5.5.2 Specifying Drill Levels

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

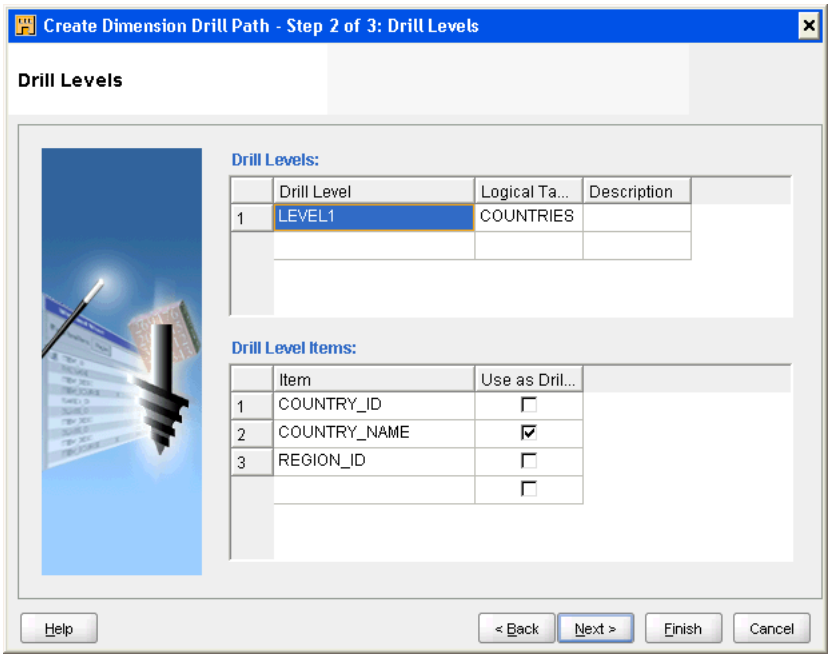
- **Drill Level:** Enter a name for the drill level.
- **Logical Table:** From the field, select the Logical Table it references.
- **Description:** Provide an optional description for the drill level.

When you select a referencing Logical Table for the Drill Level, the wizard lists the available Items within that Logical Table under the Drill Level Items field at the bottom.

In this field, you can specify one or more items to act as drill items. Select the **Use as Drill Item** option for each Item you want to include as a drill item in the level. An Item can be a drill item in a single level only.

Figure 9–2 displays the Drill Levels page of the Create Dimension Drill Path Wizard.

Figure 9–5 Creating Drill Levels



9.5.5.3 Specifying the Drill Level Key

In the Drill Level Keys section, select a Drill Level and specify a name for Drill Level Key and optionally provide a description for the key. In the Drill Level Key Items, select the Item that must used as the key.

9.5.5.4 Reviewing the Summary

In the summary page, review the drill levels you are creating. Click **Back** to make any changes or click **Finish** to create the dimension drill path.

You can locate the dimension drill path on the Projects Navigator under your OBI module. Oracle Warehouse Builder enables you to edit a dimension drill path using the Edit Dimension Drill Path dialog box.

9.5.5.5 Properties of a Dimensional Drill Path

For dimension drill paths, you can configure the following property from the Property Inspector:

Time Dimension: Select this option if it represents a time dimension.

9.5.6 Editing a Dimension Drill Path

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

To edit a dimension drill path:

1. From the Projects Navigator, expand the Dimension Drill Paths node.
2. Right-click the Dimension Drill Path and select **Open**.

   Oracle Warehouse Builder displays the Edit Dimension Drill Path dialog box containing the following editors: Name, Levels, and Level Keys.

9.5.6.1 Editing the Dimension Drill Path Name

The Name editor enables you to edit the name and the description of the dimension drill path.

9.5.6.2 Reviewing the Drill Levels in the Dimension Drill Path

Use the Drill Levels editor to edit the drill levels that you defined. The **Drill Levels** section lists the drill levels along with the logical tables that they reference. The Logical Table column displays the logical table that a dimension drill path references. You can modify this by selecting the new logical table from the list.

The **Drill Level Items** section displays the items that act as drill items. You can modify this list by selecting more items that act as drill items. You can also move a level up in same subtree, move a level out of the current tree, move a level to the `root` level, copy a level out of the current tree, remove a level, remove a level cascade, and remove a level instance, for example when there are multiple instances of the level in the network.

Use the Level Keys editor to select the Drill Level, and edit the Drill Level Keys and Drill Level Key Items.

9.5.7 Creating a Catalog Folder

Oracle Warehouse Builder enables you to create a Catalog Folder to deploy to an OBIEE repository. Catalog folders contain references to Logical Tables and Dimension

Drill Paths stored in your OBI module and are used to group information about a common subject, for example, Sales Analysis, Human Resources, or Stock Control. The OBIEE end users use these Catalog Folders as the starting point for building a query.

Catalog folders only contain references to an object and not the actual object definition. Thus, a Catalog Folder can contain a collection of unrelated Logical Tables and the same Logical Table can appear in multiple Catalog Folders. It enables you to set up multiple Catalog Folders with different levels of detail: Sales Analysis area containing one Logical Table, Sales Details area containing six Logical Tables, and a Sales Transaction area with 30 Logical Tables. When you delete a Logical Table, the reference to it from the Catalog Folder is also deleted.

You can create a Catalog Folder using the Create Catalog Folder Wizard or from the Graphical Navigator. You can also use the editor to edit a catalog folder.

9.5.7.1 Using the Create Catalog Folder Wizard

To create a Catalog Folder using the Create Catalog Folder Wizard:

1. Expand an OBI module.
2. Right-click **Catalog Folders** and select **New Catalog Folder**.

   Oracle Warehouse Builder opens the Create Catalog Folder Wizard.
3. Follow the wizard steps by clicking **Next.**

9.5.7.2 Naming the Catalog Folder

In the Name and Description page, enter a name and optional description for the Catalog Folder.

9.5.7.3 Selecting the Source

In the Source page, all the Logical Tables and Dimension Drill Paths available within the OBI module are displayed. You can multi-select the objects by pressing the Ctrl key and using the right arrow to move them to the list of Selected objects.

9.5.7.4 Reviewing the Summary

In the summary page, review the Logical Tables or the Dimension Drill Paths that you selected. Click **Back** to make any changes or click **Finish** to finish creating the Catalog Folder.

After the Catalog Folder is created, you can locate it on the Projects Navigator under the Catalog Folders node with references to the selected Logical Tables stored in it.

To make changes to your Catalog Folder definitions after you create them, use the Edit Catalog Folder dialog box. For details, see "Editing a Catalog Folder" on page 9-38.

9.5.7.5 Using the Graphical Navigator

Alternatively, you can use the main menu or the canvas to create a catalog folder.

9.5.7.5.1 Using the Menu To create a catalog folder using the menu:

1. Click anywhere on the Graphical Navigator. The main menu on the toolbar changes dynamically.
2. From the main menu, select **Graph**, **Add**, **Oracle Business Intelligence**, **Catalog Folder**.

   The Add a New or Existing Catalog Folder dialog box is displayed.

3. Select **Create a new Catalog Folder** and specify the name of the catalog folder and the module to which it belongs. Click **OK**.

The newly created catalog folder is available on the Project Navigator. Double-click the catalog folder to open the editors for the catalog folder. See "Editing a Catalog Folder" on page 9-38 for details of the editors.

9.5.7.5.2 Using the Canvas To create a catalog folder

1. From the Graphical Navigator, right-click a blank area on the editor canvas and select **Add**, **Oracle Business Intelligence**, **Catalog Folder**.

   The Add a New or Existing Catalog Folder dialog box is displayed.

2. Select **Create a new Catalog Folder** and specify the name of the catalog folder and the module to which it belongs. Click **OK**.

The newly created catalog folder is now available on the Project Navigator. Double-click the catalog folder to open the editors for the catalog folder. See "Editing a Catalog Folder" on page 9-38 for details of the editors.

9.5.8 Editing a Catalog Folder

Oracle Warehouse Builder enables you to edit the definitions for a Catalog Folder using the Edit Catalog Folder dialog box.

To edit a Catalog Folder:

1. From the Projects Navigator, expand the Catalog Folder node.
2. Right-click a Catalog Folder name and select **Open**.

   Oracle Warehouse Builder opens the Edit Catalog Folder dialog box, which contains the following editors: "Name Editor", "Source Editor", and "Contents Editor".

9.5.8.1 Name Editor

Name Editor enables you to edit the name and description of a Catalog Folder.

9.5.8.2 Source Editor

Source Editor displays the source of those Logical Tables and Dimension Drill Paths that have been included in the Catalog Folder. You can add new Logical Tables and Dimension Drill Paths to the Catalog Folder or remove existing ones using this editor.

9.5.8.3 Contents Editor

Contents Editor displays the contents of the Catalog Folder. You can add Logical Tables and Dimension Drill Paths using the Content editor. For example, to add a Logical Table, right-click anywhere on the editor and select **Add**, **Logical Table**. The Add a New or Existing Logical Table dialog box is displayed. Follow the steps listed in "Steps to Create a Logical Table" on page 9-33. You can also add a Logical Table or a Dimension Drill Path by dragging and dropping it from the Projects Navigator into the Contents editor. This creates a shortcut to the object. You can double-click the object to access it directly from the Contents editor.

9.6 Configuring Oracle Business Intelligence Objects

During the design phase, you create definitions for the OBIEE objects. After you design objects, you can assign physical properties to these design objects by setting configuration parameters.

To configure a BI object, right-click the object in the Projects Navigator and select **Configure**. The Configuration Properties dialog box is displayed. Click the object name on the left side of this dialog box to display the configuration parameters on the right.

All BI objects have a configuration parameter called **Deployable**. Select Deployable to generate scripts and deploy the business object. Oracle Warehouse Builder only generates scripts for objects marked deployable.

The following sections describe additional configuration parameters for different types of BI objects.

9.6.1 Configuration Parameters for Oracle Business Intelligence Modules

You can set the following configuration parameters for an Oracle Business Intelligence module:

Location: Represents the location to which the module is deployed.

MLS Deployment Language: Represents the language used for deployment to OBIEE.

9.7 Accessing BI Objects Using OBIEE

Once you deploy objects within Oracle Business Intelligence modules, OBIEE can then use this data to generate reports. OBIEE repositories are represented in the repository data (RPD) format. Oracle Warehouse Builder cannot directly store files in the RPD format. Instead, the file is stored in the UDML format, which is later converted to the RPD format. These are the steps involved in moving data from Oracle Warehouse Builder to OBIEE:

1. Create an OBI module and derive Oracle Warehouse Builder objects into this module.
2. Define the location of the OBI module such that it points to a UDML file on the OBIEE server.

   When the object is deployed, the UDML file is transferred to the OBIEE server using ftp or any other method of transfer, as specified while defining the location.
3. At the OBIEE server side, convert the UDML file to an RPD file. Navigate to the following path: *`OBIEE_HOME`*`\server\Bin` and run the command:

   ```
   nQUDMLExec -I x.udml -O x.rpd
   ```

   where `x.udml` is the file name specified while defining the location of the OBI module, and `x.rpd` is the target RPD file.
4. The RPD file can now be accessed by the OBIEE server.

9.7.1 Merging Updated RPD Files

After you transfer an RPD file to the OBIEE server, there might be changes made to the OBIEE module within Oracle Warehouse Builder. Similarly changes could also be made to the RPD file at the OBIEE server side. In such scenarios, you can merge the changes made to the files and create a single updated RPD file. OBIEE provides a

merge mechanism to merge an existing RPD file with a newly modified file. Let us look at the following scenarios and the possible solutions:

Example Scenario

You create an OBIEE module, for example `sales`, within Oracle Warehouse Builder and associate it with a location that points to the UDML file `sales.udml`. You then derive the warehouse definitions from a cube into this module and deploy it. On the OBIEE server, you create an RPD file from the UDML file using the following command:

```
nQUDMLExec -I  sales.udml -O  sales_original.rpd
```

The `sales_original.rpd` can now be used by Answers/Dashboard to generate the required reports.

Suppose you now rename the cube in Oracle Warehouse Builder. To update this change at the OBIEE server side, you must rederive the cube and redeploy the OBIEE module `sales`. Ensure that you select the create action for all objects. After you deploy this object, create a new repository file in OBIEE server:

```
nQUDMLExec -I  sales.udml -O  sales_modified.rpd
```

If the original file `sales_original.rpd` has not yet been used to generate reports, then you can overwrite this file with the newly created file `sales_modified.rpd`. If however, the file has been used to generate reports, then the repositories have to be merged. OBIEE provides a three-way repository merge of the following repository files:

- Original repository (`sales_original.rpd` in this case)
- The current repository (the original RPD file with modifications, if any, made at the OBIEE server side)
- The modified repository (`sales_modified.rpd` in this case)

You can use the OBIEE admin tool to merge the three files into a single updated file. If no changes have been made to the original file, then the current repository equals the original repository. In such a scenario, create a copy of the original file, `sales_original.rpd`, and name it as `sales_current.rpd`. Now from the admin tool, perform the following steps:

1. Open the current repository file `sales_current.rpd`.
2. Click **File/Merge**.

   The Merge repositories dialog box is displayed.
3. Select the original repository file `sales_original.rpd`. Also select a file to save the merged repositories.
4. Click **Merge** to merge the current and original files.
5. Now select the modified repository file `sales_modified.rpd`. Also select a file to save the merged repositories.
6. Click **Merge** to merge the modified file.

This creates a new RPD file which contains the merges from all the three files.

9.8 Deriving BI Objects

Oracle Warehouse Builder enables you to directly derive BI objects from your data warehouse design definitions. You can derive these objects into an Oracle Discoverer module or an OBI module. When you run the Perform Derivation Wizard on a warehouse module, it generates objects for business intelligence tools such as item folders from tables and drill paths from dimension hierarchies enabling you to quickly build reports from an integrated metadata platform. For example, the Perform Derivation Wizard organizes the metadata into Item Folders and Drill Paths ready to be integrated with a Discoverer EUL.

A Discoverer module can hold only those objects that are derived from an Oracle data source. However, OBI modules can also hold objects derived from non-Oracle data sources as well.

To derive BI objects:

1. From the Projects Navigator, select a module to derive. This indicates that you are deriving all the objects contained in that module. Alternatively, you can also choose to derive one object definition at a time. For example, you can select an individual table or dimension to derive.
2. Right-click the name of the warehouse module or object and select **Derive**.

   Oracle Warehouse Builder opens the Perform Derivation Wizard.
3. Follow the wizard steps using the guidelines below.

You can also start the Perform Derivation Wizard from the Graphical Navigator using the following steps:

1. Drop the source object into the navigator.
2. Right-click the source object and select **Derive**.
3. Follow the wizard steps using the guidelines below.

9.8.1 Selecting Source Objects

The Source Objects page enables you to select additional objects for derivation. The **Available** column displays all the derivable objects available in your project for deployment. These objects can belong to different warehouse modules. You can also select a collection for derivation. The Oracle module or object you selected before starting the wizard displays in the Selected Objects column.

Expand the nodes in the Available column and use the right arrow to select the objects you want to derive. Select the **Automatically add the Dimensions** option to derive the dimension objects that are associated with the selected cube objects.

9.8.2 Selecting a Target for the Derived Objects

In the Target page, indicate the Oracle Discoverer module or the OBI module in which you want to store the definitions for the derived objects. For example, if you created an Oracle Discoverer module called DISCOVERER_OBJECTS, then the name of that module displays on this page. Select DISCOVERER_OBJECTS and click **Next.**

For Discoverer modules, you can also select a Business Area as the target. In this case, shortcuts are created to the item folders in the business areas. It is recommended that you deploy to a business area. Otherwise, when you deploy objects, the objects do not belong to any Business Area and thus is not shown to end-users of BI tools. Similarly, for OBI modules, it is recommended that you deploy to a Catalog Folder.

When you select a collection for derivation, if the target is a business area, the individual objects contained in the collection are derived. Shortcuts are created to these item folders from the business area. If the target is an Oracle Discoverer module, Oracle Warehouse Builder creates a business area with the same name as the collection, stores the objects in the collection as item folders in the Oracle Discoverer module, and creates shortcuts to these item folders from the business area. This is applicable to Catalog Folders as well, when the target is an OBI module.

9.8.3 Specifying Derivation Rules

In the Rules page, specify the derivation rules and parameters. Oracle Warehouse Builder loads, configures, and executes these rules to derive the BI definitions from the selected design object definitions. You can set parameters for different rule types by selecting the type of objects from the **Rules** list. For example, you can set global rules, rules for relational objects, rules for dimension objects, or rules for cube objects. The rules and parameters that you can set are displayed on the page.

Select **Show advanced parameters** to display certain advanced rules for an object. You can also set parameters for multiple rule types.

9.8.3.1 Setting Global Rules

You can specify the following parameters:

- **Preserve user changes:** Select to preserve any manual changes to the display properties name and description.
- **Log level:** Specify the level of detail you want to see in the message log by selecting one of the options from the list. You can choose to record only errors, warnings, information, or trace debug information.
- **Validate before derive:** Select the box to validate the selected objects before deriving them.
- **Abort on error:** Select the box to stop the derivation if it encounters an error.
- **Capitalize:** To capitalize the names of the derived objects, select from the list based on your requirements.
- **Replace underscores with spaces:** Select the box to replace the underscores in the names with spaces after derivation.

You can specify the following rule for Relational objects:

- **Bound Table Suffix:** Specify a suffix for the bound tables you want to derive.
- **Default Aggregate (Oracle Discoverer only):** Specify the default aggregate function to be applied to numeric measures.
- **Remove Column name prefixes:** Select the option to remove the text immediately before an underscore in the column name. The prefix is removed provided the same prefix is used for all columns.
- **Sort items by name:** Select this option to sort the items alphabetically.

You can specify the following rules for Dimensions:

- **Always build Item Folders/Logical Tables for the dimension:** Select this option to force the Perform Derivation Wizard to create Item Folders for the derived dimension definitions.

- **Build Item Folders/Logical Tables for the levels:** Select this option to force the Perform Derivation Wizard to create Item Folders for the derived dimension levels.
- **Drill Paths on Item Folders/Logical Tables for the levels:** Select this option if you want the Perform Derivation Wizard to create Drill Paths on Item Folders being created for each dimension level. This option applies only if item folders are created for each level.
- **Prefix Items with Level Name:** Select this option to prefix the item names with the dimension level names.
- **Prefix separator:** If you choose to prefix the item names with the dimension level names, then indicate a prefix separator. The default is an underscore.
- **Sort Items by name:** Select this option to sort the items alphabetically.
- **Derive Dimension Roles:** Select this option to Perform Derivation Wizard to derive additional item folders for each role.

You can specify the following rules for Cubes:

- **Sort items by name:** Select this option to sort the items alphabetically.

9.8.4 Reviewing the Pre Derivation Rules

The Pre Derivation page displays the objects to be derived and the target or Oracle Discoverer module for storing the derived definitions.

Review this information and click **Next** to perform the derivation.

9.8.5 Reviewing Derivation Progress

The Derivation page displays a progress bar indicating the status of the derivation. When the progress bar displays 100%, the Message Log field displays any errors or warnings. At the end, the log indicates if the derivation was completed successfully.

Click **Next** to view the list of derived objects.

9.8.6 Finishing the Derivation

The Finish page displays the list of derived objects. Click **Finish** to accept the derivation. If you are not satisfied and you want to perform the derivation again, click **Back** to repeat the process.

Oracle Warehouse Builder displays the derived definitions in the appropriate BI module (Oracle Discoverer or Oracle Business Intelligence). You can edit the definitions of the derived objects or create additional definitions for deployment to Discoverer or OBIEE.

10

Importing Design Definitions from Oracle Designer

Oracle Designer is shipped with Oracle Developer Suite. Designer incorporates support for business process modeling, systems analysis, software design and system generation.

Oracle Designer provides a multiuser repository based on Oracle SCM, and is closely integrated with Oracle Forms Developer, Oracle's declarative database application development tool. In this way, Designer allows organizations to design and rapidly deliver scalable, client/server systems that can adapt to changing business needs.This chapter shows you how to import design definitions from Oracle Designer.

This chapter contains the following:

10.1 Using Oracle Designer Sources

You can create a source module that connects to an Oracle Designer repository. When the definitions for an application are stored and managed in an Oracle Designer repository, the time required to connect to the application is reduced.

Designer repositories use workareas to control versions of an object. By selecting a workarea, you can specify a version of a repository object. With Oracle Designer, you can also group objects into container elements within workareas. Container Elements contain definitions for namespace and ownership of objects, and enable you to view objects even if they are owned by a different user. Because Designer container elements are controlled by workareas, they are version controlled. See the Oracle Designer documentation for more information about workareas and container elements.

All visible objects of a workarea or a container element in Designer are available for use as data sources. To select Designer objects as a source, you must:

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

The list of repository objects available for import is determined by the following criteria:

- The object type must be supported by Oracle Warehouse Builder (Table, View, Sequence, and Synonyms).
- The object must be accessible in the specified workarea. This determines the version of objects accessed.

- The object must be visible in the specified container element. The list displays objects owned by the specified container element and other objects shared by the specified container element, but not owned by it.

To import definitions from a Designer source, you must create an Oracle database module.

10.1.1 Using Oracle Designer as a Metadata Source

To create a Designer source module:

1. Create a database source module that points to a database containing a Designer object.

   Follow the steps outlined in "Importing Metadata Definitions from Oracle Database" on page 2-9 to create the module.

2. From the Projects Navigator, double-click the name of the newly created module to open the Edit Module dialog box.

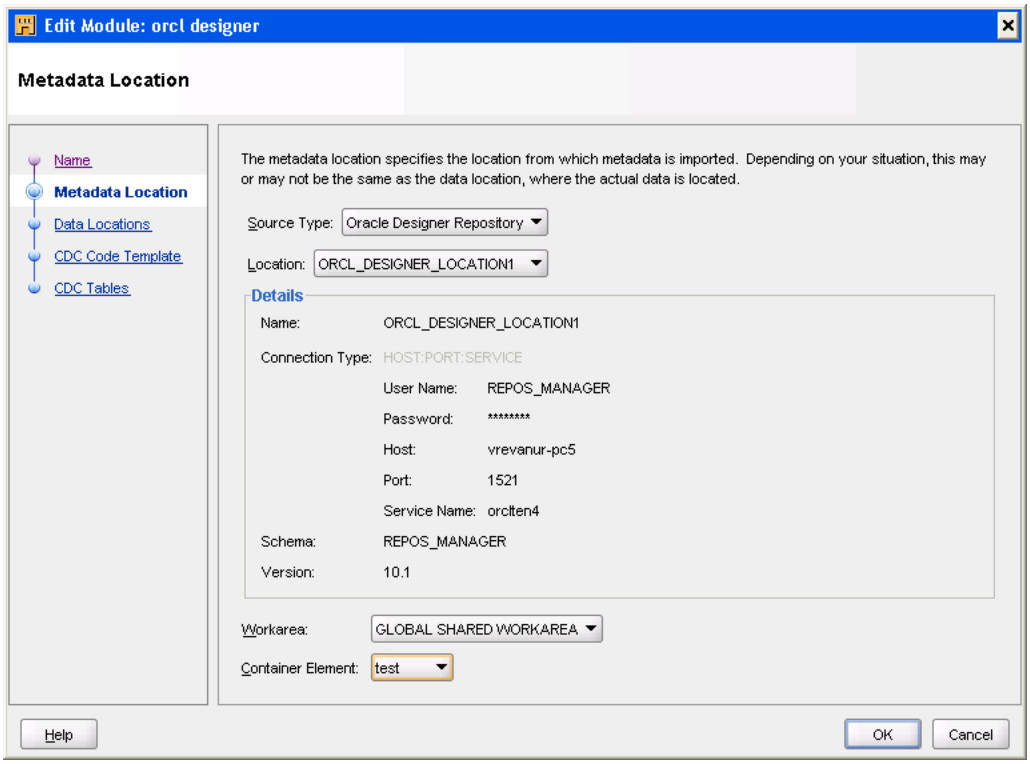
3. In the Metadata Location tab, select the source type as **Oracle Designer Repository**. Also select the database location containing the Designer object.

   When you select the source type as **Oracle Designer Repository**, two new lists, **Workarea** and **Container Element**, are visible in the Metadata Location tab.

   Figure 10–1 displays the Metadata Locations tab of the Edit Module dialog box.

4. Select the Designer object from the workarea and select the specific container element.

Figure 10–1 The Metadata Location Tab



****

> **Note:** The database you specify as source must contain a Designer object. If not, then the **Workarea** and **Element Container** lists do not display any values.

5. Click **OK**.

For related information, see the following sections:

- "Importing Metadata Definitions from Oracle Database" on page 2-9
- "Reimporting Definitions" on page 2-14
- "Updating Source Module Definitions" on page 2-16

11

Creating New Platforms

Starting with Oracle Warehouse Builder 11*g* Release 2 (11.2), you can create new platforms to integrate with other systems and construct integration capabilities using code templates based on your requirements. A platform refers to a data source or target. By creating new platforms, you can thus connect to new databases in addition to those that are supported by default in Oracle Warehouse Builder.

This chapter contains the following topics:

11.1 Creating a New Platform

When you create a new platform, you must define properties including connection information, code generation options, data types supported by the platform, and how these data types map to the generic data types.

To create the platform and define its characteristics, you must use OMB*Plus scripting commands.

For more information about OMB*Plus scripting, see *Oracle Warehouse Builder API and Scripting Reference*.

> **See Also:** *Oracle Warehouse Builder OMB*Plus Command Reference* for a list of all OMB*Plus commands.

The `OMBCREATE` command creates a new platform:

```
OMBCREATE PLATFORM 'MICROSOFT_EXCEL' SET PROPERTIES (BUSINESS_NAME) VALUES
('Microsoft Excel')
```

The business name is displayed in the Projects Navigator.

To define the properties of this platform, use the `OMBALTER` command:

```
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD PLATFORM_TYPE 'CHAR'
```

For a complete listing of the commands to create and define a new platform, see "Creating a Microsoft Excel Platform" on page 11-13. You can add to this depending on your specific requirements.

When you create a platform from the OMB*Plus interface, the platform gets added under the Databases node in the Projects Navigator. The location of the platform gets added under the Databases node in the Locations Navigator. You can now create a module under this new platform.

11.2 Defining the Properties of a New Platform

While creating a new platform, you must define certain basic properties for the platform. These include:

- Designer and run-time properties
- Platform data types
- CMI/MIV to be used for custom import
- Data type definitions for the mappings
- JDBC driver for the platform

Table 11–1 lists the properties that must be specified for a platform.

Table 11–1 Properties of a Platform

Property	Data Type	Description
NAME	STRING	The physical name of the platform.
BUSINESS_NAME	STRING	Business name of the platform.
DESCRIPTION	STRING	Description of the platform.
DRIVERCLASS	STRING	The default JDBC driver class. For example `com.sunopsis.jdbc.driver.xml.SnpsXmlDriver` for XML.
URL_TEMPLATE	STRING	The default URL for the driver. This is used as a template.
COL_ALIAS_WORD	STRING	The separator for a column and its alias. It is not mandatory to specify this property.
TAB_ALIAS_WORD	STRING	The separator for a table and its alias. It is not mandatory to specify this property.
DATE_FCT	STRING	The function that returns the date and time. For example `sysdate` for Oracle.
DDL_NULL	STRING	A column that can hold NULL value.
DEFAULT_MAX_NAME_LEN	INTEGER	The maximum length for a table name. If you specify a name longer than this limit, the name is truncated to this length.
DEFAULT_NAME_LEN_SEMANTIC	STRING	Whether the name length is specified in characters or bytes.
SPECIAL_MAX_NAME_LEN	STRING	Name length for second class objects (SCOs). (For example INDEX=18 or COLUMN=30).
SPECIAL_NAME_LEN_SEMANTICS	STRING	Whether SPECIAL_MAX_NAME_LEN is specified in terms of characters or bytes.

Table 11–1 (Cont.) Properties of a Platform

Property	Data Type	Description
ESCAPE_CHAR	STRING	The escape character. For example, it is the double quotation mark (") for Oracle Database.
ENCLOSURE_CHAR	STRING	The enclosure character. For example, it is the backslash (\) for Oracle Database.
RESERVED_WORDS	STRING	Reserved words for a platform.
ILLEGAL_CHARS	STRING	Characters that cannot be used while naming objects in a platform.
ILLEGAL_LEADING_CHARS	STRING	Characters that cannot be used as the first character in the name of an object.
CUSTOM_IMPORTERS		The custom metadata definitions that are associated to a platform.
Data Types		The data types supported by a platform. You can specify the syntax to define a data type and any parameter that the data type uses. For example the length, precision, and scale.
Data Type map to Generic platform		The mapping of a platform's data types to the generic data types.
Data Type map from Generic platform		The mapping of generic data types to a platform's data types.
VARCHAR_MASK	STRING	Run-time properties used by the platform while executing Code Templates.
DATE_MASK	STRING	
NUMERIC_MASK	STRING	The syntax to be used to describe the numeric data type in DDL. The tags %L (data length) and %P (precision) can be used.

11.2.1 Defining the Platform Type

Based on your requirements, you can provide support for various data types in a platform. The properties of the Data Type define the scope of a data type. These properties are listed in Table 11–2.

Table 11–2 Properties of Data Type

Property	Type	Description
NAME	STRING	The physical name of the data type.
BUSINESS_NAME	STRING	The business name of the data type.
DESCRIPTION	STRING	Description of the data type.
SYNTAX	STRING	The syntax to be used during code generation. For example `CHAR[(%size)]`.
P1	STRING	Either size, precision, or scale.
P1TYPE	STRING	Specifies the range.

Table 11–2 (Cont.) Properties of Data Type

Property	Type	Description
P1DEFAULT	STRING	Default value for P1.
P1MIN	STRING	Minimum value for P1.
P1MAX	STRING	Maximum value for P1.
P2	STRING	Specifies either Size, Precision, or Scale.
P2TYPE	STRING	Specifies the range.
P2DEFAULT	STRING	Default value for P2.
P2MIN	STRING	Minimum value for P2.
P2MAX	STRING	Maximum value for P2.

11.2.2 Defining the Data Type Map

After you define data types for a platform, you must also define how these data types map to other generic data types. You can define these data type mappings with the properties listed in Table 11–3.

Table 11–3 Data Type Mapping Properties

Property	Description
NAME	The physical name of the map.
BUSINESS_NAME	Business name of the map.
DESCRIPTION	Description of the map.
FROM_DATATYPE	The data type to map from.
TO_DATATYPE	Data type to be mapped to.
CONDITION1	Defines the condition where the data type mapping is valid. For example, mapping from a generic CHAR data type to a DB2UDB CHAR data type is dependent on the size of the characters, and this size is specified in the CONDITION1 parameter.
CONDITION2	Used for setting additional condition.
CONDITION3	Used for setting additional condition.

11.2.3 Generic Data Types Supported in Oracle Warehouse Builder

The Generic data types supported in Oracle Warehouse Builder are listed in Table 11–4.

Table 11–4 Generic Data Types

Category	Generic Data Type	Parameters	Description
Numbers	BIGINT		Range is -2^63 to 2^63-1

Table 11–4 (Cont.) Generic Data Types

Category	Generic Data Type	Parameters	Description
	BINARY_FLOAT		Single precision floating point numbers Minimum positive finite value = 1.17549E-38F Maximum positive finite value = 3.40282E+38F
	BINARY_DOUBLE		Double precision floating point numbers Minimum positive finite value = 2.22507485850720E-308 Maximum positive finite value = 31.79769313486231E+308
	BIT		An integer data type that can take a value of 1, 0, or NULL
	FLOAT	FLOAT (precision)	-1.79769E+308 to -2.23E-308, 0 and 2.23E-308 to 1.79769E+308
	DOUBLE	DOUBLE (precision)	-1.79769E+308 to -2.23E-308, 0 and 2.23E-308 to 1.79769E+308
	INTEGER		NUMERIC(38)
	INT10		The range of large integers is -2 147 483 648 to +2 147 483 647
	MONEY		Ranges from -922,337,203,685,477.5808 to 922,337,203,685,477.5807. (Used only by MS SQL Server)
	NUMERIC	NUMERIC [(precision [, scale])]	The precision p can range from 1 to 38. The scale s can range from -84 to 127
	DECIMAL	DECIMAL [(precision [, scale])]	The precision p can range from 1 to 38. The scale s can range from -84 to 127
	REAL		Ranges from - 3.40E + 38 to -1.18E - 38, 0 and 1.18E - 38 to 3.40E + 38
	SMALLINT		Ranges from -2^15 (-32,768) to 2^15-1 (32,767)
	SMALLMONEY		Ranges from - 214,748.3648 to 214,748.3647
	TINYINT		Ranges from 0 to 255
Large Objects	BLOB	BLOB	A binary large object, with no limit on the maximum size
	VARLOB	VARBLOB [(size[K|M|G])]	A binary large object is a varying-length binary string that can be up to 2 GB (2 147 483 647 bytes) long. The default value is 1 MB (1,048,576)
	CLOB		Character large object. No limit on the maximum size

Table 11–4 (Cont.) Generic Data Types

Category	Generic Data Type	Parameters	Description
	VARCLOB	VARCLOB [(size[K | M | G])]	A CLOB (character large object) value can be up to 2 GB (2 147 483 647 bytes) long. The default value is 1 MB (1,048,576)
	DBCLOB	DBCLOB [(size[K | M | G])]	A DBCLOB (double-byte character large object) value can be up to 1 073 741 823 double-byte characters long. (Only used by IBM DB2 UDB)
	NCLOB		Character large object in Unicode or double-byte. No limit on the maximum size
Character Strings	CHAR	CHAR[(size)]	Fixed length character data with size between 1 and 8000
	GRAPHIC	GRAPHIC (size)	The size attribute must be between 1 and 127, inclusive. (Only used by IBM DB2 UDB)
	LONGVARGRAPHIC	LONGVARGRAPHIC (size)	VARGRAPHIC value can be up to 16 350 double-byte characters long. (Only used by IBM DB2 UDB)
	NCHAR	NCHAR (size)	Fixed length (1 to 2000) Unicode or double-byte character data Size: 1-4000
	NVARCHAR	NVARCHAR (size)	Variable length (1 to 4000) Unicode or double-byte character data
	NVARCHARMAX	NVARCHARMAX	Variable length (1 to 2^31-1) Unicode or double-byte character data
	VARCHAR	VARCHAR (size)	Variable length (1 to 8000) character data
	VARCHARMAX	VARCHARMAX	Variable length (1 to 2^31) character data
	VARGRAPHIC	VARGRAPHIC (size)	A VARGRAPHIC value can be up to 16 336 double-byte characters long. (Only used by IBM DB2 UDB)
Date and Time	DATE		Date in year, month, day, hour, minute, and second without fractional seconds precision or time zone
	DATETIME		Range is January 1, 1753, through December 31, 9999, accuracy is 3.33 milliseconds. (MS SQL Server only)

Table 11–4 (Cont.) Generic Data Types

Category	Generic Data Type	Parameters	Description
	INTERVAL YEAR TO MONTH	INTERVAL DAY [(day_precision)] TO SECOND [(fractional_ seconds_precision)]	Stores a period in days, hours, minutes, and seconds, where day_precision is the maximum number of digits in the DAY datetime field. Accepted values are 0 to 9. The default is 2. fractional_seconds_precision is the number of digits in the fractional part of the SECOND field. Accepted values are 0 to 9. The default is 6. The size is fixed at 11 bytes.
	SMALLDATETIME		Range is January 1, 1900, through June 6, 2079; accuracy is 1 minute
	TIME		A TIME is a three-part value (hour, minute, and second) designating a time of day under a 24-hour clock. The range of the hour part is 0 to 24. The range of the other two parts is 0 to 59. If the hour is 24, the minute and second specifications are zero.
	TIMESTAMP	TIMESTAMP [(fractional_ seconds_precision)]	DATE with factional seconds precision
	TIMESTAMP WITH TIME ZONE	TIMESTAMP [(fractional_ seconds_precision)] WITH TIME ZONE	TIMESTAMP plus time zone displacement value
	TIMESTAMP WITH LOCAL TIME ZONE	TIMESTAMP [(fractional_ seconds_precision)] WITH LOCAL TIME ZONE	TIMESTAMP normalized to the database time zone
Binary Strings	BINARY	BINARY [(size)]	Binary data up to 8000 bytes
	VARBINARY	VARBINARY [(size)]	Binary data of variable length up to 8000
	VARBINARYMAX	VARBINARY [(size)]	Binary data of variable length up to 2 GB
	LONGVARBINARY	LONGVARBINARY	Raw binary data of variable length up to 2 gigabytes
	IMAGE	IMAGE	Variable-length binary data from 0 through 2^31-1 (2,147,483,647) bytes
Others	UNDEFINED	UNDEFINED	Represent all non-supported data types (catch-all)
	XMLTYPE	XMLTYPE	Only mapped between Oracle/Oracle Work Flow and Generic
	BOOLEAN	BOOLEAN	Only mapped between Oracle/Oracle Work Flow and Generic

11.2.4 DB2 Data Types Mapping

Table 11–5 lists the mapping of DB2 data types to generic data types.

Table 11–5 DB2 Data Types to Generic Data Types

DB2 Data Type	Generic Data Type
CHARACTER, CHAR	CHAR
VARCHAR, CHARACTER VARYING, CHAR VARYING	VARCHAR
LONG VARCHAR	LONG VARCHAR
GRAPHIC	GRAPHIC
VARGRAPHIC	VARGRAPHIC
LONG VARGRAPHIC	LONG VARGRAPHIC
DBCLOB	DBCLOB
SMALLINT	SMALLINT
INTEGER, INT	INT10
BIGINT	BIGINT
NUMERIC, NUM	NUMERIC
DECIMAL, DEC	DECIMAL
REAL	REAL
FLOAT	FLOAT
DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP
TIME	TIME
BLOB, BINARY LARGE OBJECT	VARLOB
CLOB, CHARACTER LARGE OBJECT, CHAR LARGE OBJECT	VARCLOB

Table 11–6 lists the mapping of generic data types to DB2 data types.

Table 11–6 Generic Data Types to DB2 Data Types

Generic Data Type	DB2 Data Type
BIGINT	BIGINT
BINARY_FLOAT	REAL
BINARY_DOUBLE	FLOAT(53)
BIT	NUMERIC(1)
FLOAT [(precision)]	FLOAT [(precision)]
DOUBLE [(precision)]	DOUBLE [(precision)]
INTEGER	NUMERIC(31)
INT10	INTEGER
MONEY	REAL

Table 11–6 (Cont.) Generic Data Types to DB2 Data Types

Generic Data Type	DB2 Data Type
NUMERIC [(precision [, scale])]	NUMERIC [(precision [, scale])]
DECIMAL [(precision [, scale])]	DECIMAL [(precision [, scale])]
REAL	REAL
SMALLINT	SMALLINT
SMALLMONEY	REAL
TINYINT	SMALLINT
BLOB	BLOB
VARLOB	BLOB
CLOB	CLOB (2147483647)
VARCLOB	CLOB
DBCLOB [(size [K|M|G])]	DBCLOB [(size [K|M|G])]
NCLOB	DBCLOB
CHAR [(size)]	CHAR [(size)]
GRAPHIC (size)	GRAPHIC (size)
LONGVARCHAR	LONGVARCHAR
LONGVARGRAPHIC	LONGVARGRAPHIC
NCHAR [(size)]	VARGRAPHIC [(size)]
NVARCHAR (size)	VARGRAPHIC (size)
NVARCHARMAX	DBCLOB
VARCHAR (size)	size<=32,672: VARCHAR 32672<size<=32700:LONG VARCHAR size>32,700: CLOB
VARCHARMAX	CLOB
VARGRAPHIC (n)	VARGRAPHIC (n)
DATE	DATE
DATETIME	TIMESTAMP
INTERVAL YEAR [(year_ precision)] TO MONTH	VARCHAR
INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds_ precision)]	VARCHAR
SMALLDATETIME	TIMESTAMP
TIME	TIME
TIMESTAMP [(fractional_seconds_ precision)]	TIMESTAMP
TIMESTAMP [(fractional_seconds_ precision)] WITH TIME ZONE	VARCHAR
TIMESTAMP [(fractional_seconds_ precision)] WITH LOCAL TIME ZONE	VARCHAR

Table 11–6 (Cont.) Generic Data Types to DB2 Data Types

Generic Data Type	DB2 Data Type
BINARY [(size)]	size<=254: CHAR (size) FOR BIT DATA size>254:VARCHAR(size) FOR BIT DATA
VARBINARY [(size)]	size<=32,672:VARCHAR(size) FOR BIT DATA size>32,672: BLOB
VARBINARYMAX	BLOB
LONGVARBINARY	BLOB
IMAGE	BLOB
UNDEFINED	VARCHAR(32672)
XMLTYPE	VARCHAR(32672)
BOOLEAN	VARCHAR(10)

There might be precision loss in the following cases:

- Mapping a generic NUMERIC data type (up to 38) to DB2 NUMERIC data type with a maximum precision of 31
- Mapping a generic DECIMAL data type (up to 38) to DB2 DECIMAL data type with a maximum precision of 31
- Mapping a generic INTEGER data type (up to 38) to DB2 NUMERIC(31) data type with a maximum precision of 31

11.2.5 MS SQL Server Data Types Mapping

Table 11–7 lists the mapping of MS SQL Server data types to generic data types.

Table 11–7 MS SQL Server Data Types to Generic Data Types

MS SQL Server Data Type	Generic Data Type
CHAR	CHAR
VARCHAR	VARCHAR
VARCHAR(MAX)	VARCHARMAX
TEXT	VARCHAR(2147483647)
NCHAR	NCHAR
NVARCHAR	NVARCHAR
NVARCHAR(MAX)	NVARCHARMAX
NTEXT	NVARCHAR
BINARY	BINARY
VARBINARY	VARBINARY
VARBINARY(MAX)	VARBINARYMAX
IMAGE	IMAGE
SMALLINT	SMALLINT
INT	INT10
BIGINT	BIGINT

Table 11–7 (Cont.) MS SQL Server Data Types to Generic Data Types

MS SQL Server Data Type	Generic Data Type
TINYINT	TINYINT
BIT	BIT
MONEY	MONEY
SMALLMONEY	SMALLMONEY
NUMERIC	NUMERIC
DECIMAL	DECIMAL
REAL	REAL
FLOAT	FLOAT
DATETIME	DATETIME
SMALLDATETIME	SMALLDATETIME
UNIQUEIDENTIFIER	UNDEFINED
XML	UNDEFINED
TIMESTAMP	UNDEFINED
SQL_VARIANT	UNDEFINED

Table 11–8 lists the mapping of generic data types to MS SQL Server data types.

Table 11–8 Generic Data Types to MS SQL Server Data Types

Generic Data Type	MS SQL Server Data Type
BIGINT	BIGINT
BINARY_FLOAT	REAL
BINARY_DOUBLE	FLOAT
BIT	BIT
FLOAT	FLOAT
DOUBLE	FLOAT
INTEGER	NUMERIC(38)
INT10	INT
MONEY	MONEY
NUMERIC	NUMERIC
DECIMAL	DECIMAL
REAL	REAL
SMALLINT	SMALLINT
SMALLMONEY	SMALLMONEY
TINYINT	TINYINT
BLOB	VARBINARY(MAX)
VARBLOB(N)	VARCHAR(MAX)
CLOB	VARCHAR(MAX)
VARCLOB(N)	VARCHAR(MAX)

Table 11–8 (Cont.) Generic Data Types to MS SQL Server Data Types

Generic Data Type	MS SQL Server Data Type
DBCLOB	NVARCHAR(MAX)
NCLOB	NVARCHAR(MAX)
CHAR(N)	CHAR(N)
GRAPHIC(N)	NCHAR(254)
LONGVARCHAR	VARCHAR(MAX)
LONGVARGRAPHIC(N)	NVARCHAR(MAX)
NCHAR(N)	NCHAR(N)
NVARCHAR(N)	NVARCHAR(N)
NVARCHARMAX	NVARCHAR(MAX)
VARCHAR(N)	1<=n<=8000: varchar (n) n>8000: varchar (max)
VARCHARMAX	VARCHAR(MAX)
VARGRAPHIC(N)	NVARCHAR(MAX)
DATE	DATETIME
DATETIME	DATETIME
INTERVAL DAY TO SECOND	VARCHAR
INTERVAL YEAR TO MONTH	VARCHAR
SMALLDATETIME	SMALLDATETIME
TIME	VARCHAR
TIMESTAMP	DATETIME
TIMESTAMP WITH TIME ZONE	DATETIME
TIMESTAMP WITH LOCAL TIME ZONE	DATETIME
BINARY (size)	BINARY (size)
VARBINARY (size)	1<=size<=8000: varbinary (size) n>8000: varbinary (max)
VARBINARYMAX	VARBINARY(MAX)
LONGVARBINARY	VARBINARY(MAX)
IMAGE	IMAGE
UNDEFINED	VARCHAR(MAX)
XMLTYPE	VARCHAR(MAX)
BOOLEAN	VARCHAR(10)

11.2.6 Creating Modules Based on a Platform

When you run the `OMBCREATE` command to create a platform, the corresponding platform node is added under the `Databases` node in the Projects Navigator. You can now create a module under this new platform.

For example, to create a module under **Microsoft_Excel**, right-click **Microsoft_Excel** and select **New Microsoft_Excel Module**. The Create Module Wizard guides you through the steps to create a module. This includes:

1. "Providing a Name and Access Method"
2. "Providing Connection Information"

Providing a Name and Access Method

On the Name and Description page, provide a name for the module. By default, this is a Generic Access module. Select the access method as well. It can be either **Native Database Connection** or **Gateway**. For native database connectivity, Oracle Warehouse Builder supports importing metadata based on JDBC. So if a JDBC or ODBC driver is installed on the system, you can use this driver for the data import. Click **Next** to open the Connection Information page.

Providing Connection Information

Native Database Connection implies a generic JDBC connection. Click **Edit** to open the Edit Generic JDBC Location dialog box and provide the location information. You must provide a JDBC URL in the UrL field and a JDBC driver in the Driver Class field.

If you select Gateway connection, then provide the database connection details.

A platform module supports the following objects:

- Transformations
- Tables
- Views

The data you can store in these objects depends on the data types you defined for the platform.

You can also import metadata by defining a Custom Metadata Interface (CMI) that is based on a custom API.

11.3 Creating a Microsoft Excel Platform

The following example creates a new platform to extract data from Microsoft Excel worksheets. This example lists only the most basic options to create the platform, including the connection information, and the supported data types. You can build on the platform by inserting additional features.

First, define the platform on the Projects Navigator:

```
OMBCREATE PLATFORM 'MICROSOFT_EXCEL' SET PROPERTIES (BUSINESS_NAME) VALUES
('Microsoft Excel')
```

Next, define the connection information for the platform. This includes specifying the ODBC:JDBC driver and the URL template:

```
OMBALTER PLATFORM 'MICROSOFT_EXCEL' SET PROPERTIES (DRIVER_CLASS,URI_TEMPLATE)
VALUES
('sun.jdbc.odbc.JdbcOdbcDriver','jdbc:odbc:Driver={Microsoft Excel Driver
(*.xls)}\;
DBQ=<filename>\;DriverID=22\;READONLY=true')
```

We then define the properties for the platform:

```
OMBALTER PLATFORM 'MICROSOFT_EXCEL' SET PROPERTIES (LOCAL_OBJECT_MASK) VALUES
('%SCHEMA.%OBJECT')
```

```
OMBALTER PLATFORM 'MICROSOFT_EXCEL' SET PROPERTIES (DATE_MASK) VALUES ('DATETIME')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' SET PROPERTIES (DDLNULL) VALUES ('null')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' SET PROPERTIES (NUMERIC_MASK) VALUES
('NUMBER')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' SET PROPERTIES (VARCHAR_MASK) VALUES
('VARCHAR(%L)')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' SET PROPERTIES (LOCAL_OBJECT_MASK) VALUES
('%OBJECT')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' SET PROPERTIES (DEFAULT_MAX_NAME_LEN) VALUES
('30')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' SET PROPERTIES (REMOTE_OBJECT_MASK) VALUES
('%OBJECT')
```

We finally define the data types supported by the platform:

```
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD PLATFORM_TYPE 'LOGICAL'
OMBALTER PLATFORM 'MICROSOFT_EXCEL' MODIFY PLATFORM_TYPE 'LOGICAL' SET
PROPERTIES(SYNTAX) VALUES ('LOGICAL')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD FROM_PLATFORM_TYPEMAP 'LOGICAL_TOG' SET
PROPERTIES (FROM_DATATYPE, TO_DATATYPE) VALUES ('LOGICAL', 'CHAR')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD TO_PLATFORM_TYPEMAP 'LOGICAL_FROMG' SET
PROPERTIES (FROM_DATATYPE, TO_DATATYPE) VALUES ('CHAR', 'LOGICAL')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD PLATFORM_TYPE 'CURRENCY'
OMBALTER PLATFORM 'MICROSOFT_EXCEL' MODIFY PLATFORM_TYPE 'CURRENCY' SET
PROPERTIES(P1,P1MIN, P1MAX,P1DEFAULT,P1TYPE) VALUES ('precision','1', '64000',
'1','range')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' MODIFY PLATFORM_TYPE 'CURRENCY' SET
PROPERTIES(P2,P2MIN, P2MAX,P2DEFAULT,P2TYPE) VALUES ('scale','1', '18',
'1','range')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' MODIFY PLATFORM_TYPE 'CURRENCY' SET
PROPERTIES(SYNTAX) VALUES ('CURRENCY(%precision,%scale)')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD FROM_PLATFORM_TYPEMAP 'CURRENCY_TOG' SET
PROPERTIES (FROM_DATATYPE, TO_DATATYPE) VALUES ('CURRENCY', 'NUMERIC')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD PLATFORM_TYPE 'NUMBER'
OMBALTER PLATFORM 'MICROSOFT_EXCEL' MODIFY PLATFORM_TYPE 'NUMBER' SET
PROPERTIES(P1,P1MIN, P1MAX,P1DEFAULT,P1TYPE) VALUES ('precision','1', '64000',
'1','range')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' MODIFY PLATFORM_TYPE 'NUMBER' SET
PROPERTIES(P2,P2MIN, P2MAX,P2DEFAULT,P2TYPE) VALUES ('scale','1', '18',
'1','range')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' MODIFY PLATFORM_TYPE 'NUMBER' SET
PROPERTIES(SYNTAX) VALUES ('NUMBER(%precision,%scale)')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD FROM_PLATFORM_TYPEMAP 'NUMBER_TOG' SET
PROPERTIES (FROM_DATATYPE, TO_DATATYPE) VALUES ('NUMBER', 'NUMERIC')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD TO_PLATFORM_TYPEMAP 'NUMBER_FROMG' SET
PROPERTIES (FROM_DATATYPE, TO_DATATYPE) VALUES ('NUMERIC', 'NUMBER')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD PLATFORM_TYPE 'DATETIME'
OMBALTER PLATFORM 'MICROSOFT_EXCEL' MODIFY PLATFORM_TYPE 'DATETIME' SET
PROPERTIES(SYNTAX) VALUES ('DATETIME')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD FROM_PLATFORM_TYPEMAP 'DATETIME_TOG' SET
PROPERTIES (FROM_DATATYPE, TO_DATATYPE) VALUES ('DATETIME', 'DATETIME')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD TO_PLATFORM_TYPEMAP 'DATETIME_FROMG' SET
PROPERTIES (FROM_DATATYPE, TO_DATATYPE) VALUES ('DATETIME', 'DATETIME')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD PLATFORM_TYPE 'VARCHAR'
OMBALTER PLATFORM 'MICROSOFT_EXCEL' MODIFY PLATFORM_TYPE 'VARCHAR' SET
PROPERTIES(P1,P1MAX,P1DEFAULT,P1TYPE) VALUES ('size','64000', '1','range')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' MODIFY PLATFORM_TYPE 'VARCHAR' SET
PROPERTIES(SYNTAX) VALUES ('VARCHAR(%size)')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD FROM_PLATFORM_TYPEMAP 'VARCHAR_TOG' SET
PROPERTIES (FROM_DATATYPE, TO_DATATYPE) VALUES ('VARCHAR', 'VARCHAR')
OMBALTER PLATFORM 'MICROSOFT_EXCEL' ADD TO_PLATFORM_TYPEMAP 'VARCHAR_FROMG' SET
```

```
PROPERTIES (FROM_DATATYPE, TO_DATATYPE) VALUES ('VARCHAR', 'VARCHAR')
```

Run this script from the OMB*Plus console in Oracle Warehouse Builder. The newly created platform is now visible as a node in the Projects Navigator.

11.3.1 Importing an Excel Worksheet

To import an Excel worksheet:

1. Create a new Excel module from the MICROSOFT EXCEL platform node in the Projects Navigator.
2. On the Name and Descriptions page of the Create Module wizard, specify **Native Database Connection** as the access method.
3. On the Connection Information page, click **Edit** to open the Edit Generic JDBC Location dialog box.

   Provide a dummy user name and password as shown in Figure 11–1. You can provide any value for the user name and password, but cannot leave the fields blank. Values for Driver Class and URL fields are set, based on the values provided in the script. However, you must edit the UrL field to point to the location of the Excel file.

   The URL field contains the value:

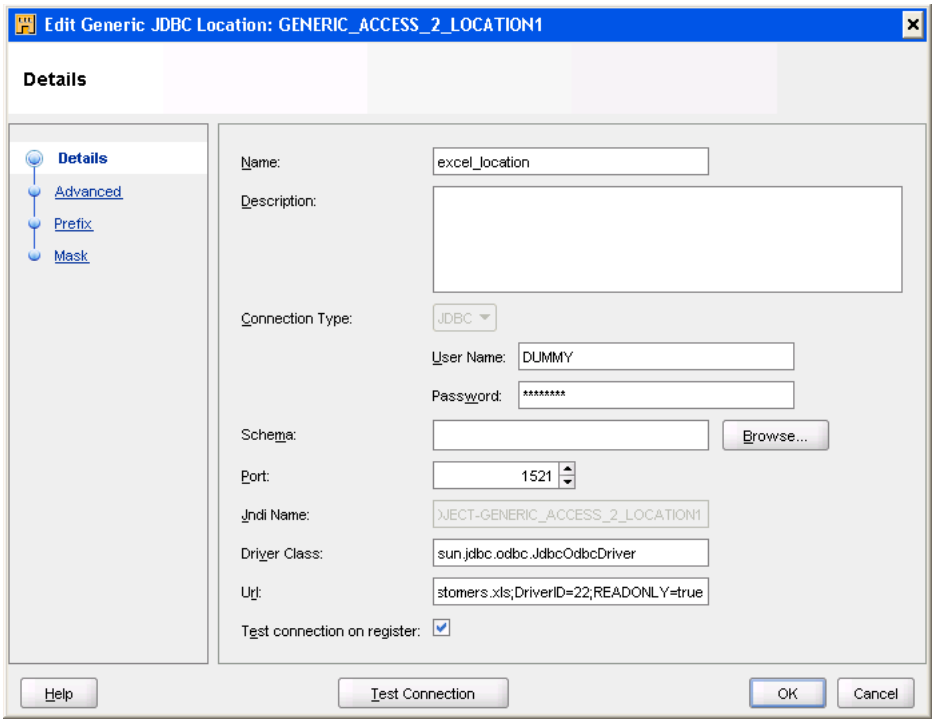
   ```
   jdbc:odbc:Driver={Microsoft Excel Driver
   (*.xls)};DBQ=<filename>;DriverID=22;READONLY=true
   ```

   The file name field must be altered to point to the location of the Excel file. For example:

   ```
   jdbc:odbc:Driver={Microsoft Excel Driver (*.xls)};DBQ=c:/my_
   projects/excel/employees.xls;DriverID=22;READONLY=true
   ```

   In this example, the location points to the file `employees.xls`.

Figure 11–1 Connection Information for the Excel File



After you create the Excel module and provide the location details for the excel file, you can import the table definitions from the file.

To import the definitions:

1. Right-click the module and select **Database Objects**.

   The Import Metadata Wizard is displayed.

2. Import the table from the Excel file in the same way you import tables from a database.

3. To view the data in the table, right-click the imported table and select **Data**.

11.4 Using Custom Metadata Import in Platforms

Using a CMI mechanism, you can define how metadata from a database is to be imported into Oracle Warehouse Builder. You can define a CMI that leverages the SQL definitions or the API definitions of the database from which you want to import metadata.

11.4.1 Example: Implementing SQL-Based CMI Import for DB2 UDB

You can define a CMI that uses SQL to retrieve metadata from the SQL-based data dictionary of DB2 UDB and import tables from a DB2 platform. You can implement a similar mechanism to import metadata from any database that uses an SQL-based data dictionary.

To leverage on a CMI mechanism, you must define a CMI DEFINITION for the platform. CMI definitions can only be created from the `root` context. You can switch to the `root` context only from the OMB Plus console. You cannot switch to the `root` context using the OMB*Plus view from within the Oracle Warehouse Builder UI.

To use the OMB Plus console on a Windows system, select **Start**, then **All Programs**, **<OWB>**, **Warehouse Builder**, and then **OMB Plus**.

To switch to the `root` context, use the following command:

```
OMBCONNECT <repository user>/<password>@<host>:<port number>:<service name>
```

For example,

```
OMBCONNECT rep_user/password@localhost:1521:orcl
```

Where `rep_user/password` is the user name/password to connect to the repository, `localhost` indicates a local installation, `1521` is the port number, and `orcl` is the service name of the database.

Example 11–1 lists a platform definition for DB2 UDB. This lists only the basic definition and does not include the data types that can be added to the platform.

Example 11–1 Platform Definition for IBM DB2 UDB

```
set platformname IBM_DB2_UDB
set platformdisplay "IBM DB2 CMI Api"

puts "Creating Platform $platformdisplay"

OMBCREATE PLATFORM '$platformname' SET PROPERTIES (BUSINESS_NAME) VALUES
('$platformdisplay')
OMBALTER PLATFORM '$platformname' SET PROPERTIES (DRIVER_CLASS,URI_TEMPLATE)
VALUES ('com.ibm.db2.jcc.DB2Driver','jdbc:db2://Host:Port/Database Name')
OMBALTER PLATFORM '$platformname' SET PROPERTIES (LOCAL_OBJECT_MASK) VALUES
('%SCHEMA.%OBJECT')
OMBALTER PLATFORM '$platformname' SET PROPERTIES (DATE_MASK) VALUES ('TIMESTAMP')
OMBALTER PLATFORM '$platformname' SET PROPERTIES (DDLNULL) VALUES ('null')
OMBALTER PLATFORM '$platformname' SET PROPERTIES (NUMERIC_MASK) VALUES
('NUMERIC(%L,%P)')
OMBALTER PLATFORM '$platformname' SET PROPERTIES (VARCHAR_MASK) VALUES
('VARCHAR(%L)')
OMBALTER PLATFORM '$platformname' SET PROPERTIES (LOCAL_OBJECT_MASK) VALUES
('%SCHEMA.%OBJECT')
OMBALTER PLATFORM '$platformname' SET PROPERTIES (DEFAULT_MAX_NAME_LEN) VALUES
('30')
OMBALTER PLATFORM '$platformname' SET PROPERTIES (REMOTE_OBJECT_MASK) VALUES
('%SCHEMA.%OBJECT')
```

Next, add a CMI definition to the platform.

```
OMBCREATE CMI_DEFINITION 'DB2_IMPORT_SQL' USING DEFINITION_FILE 'c:\\platformdef2_
miv.xml'
OMBALTER CMI_DEFINITION 'DB2_IMPORT_SQL' SET PROPERTIES (MIV_TYPE) VALUES
('Databases')
OMBALTER PLATFORM '$platformname' SET REF CMI_DEFINITION 'DB2_IMPORT_SQL'
```

Save the changes using the command `OMBCOMMIT`.

> **Note:** You can store the entire script as a `.tcl` file and run the file from the OMB Plus console.

The file `platformdef2_miv.xml` contains the custom import definition to import metadata using SQL. Example 11–2 lists the CMI definition file.

Example 11–2 CMI Definition File

```
<?xml version="1.0"?>
<miv>

<miv_tables type="SQLStatement" default="true">
        SELECT rtrim(tabname) table_name
        FROM syscat.tables
        WHERE tabschema = <Parameter name="owner"/>
                AND type = 'T'
                AND status = 'N'
        ORDER BY table_name
</miv_tables>

<miv_columns type="SQLStatement" default="true">
        SELECT rtrim(col.tabname) entity_name,
               col.colno position,
               rtrim(col.colname),
               col.typename,
               col.length,
               col.length,
               col.scale,
               col.remarks,
               col.nulls,
               '' datatypeowner,
               col.default,
               CHAR(col.codepage) charset,
               1 bytes_per_char,
               'N' use_char_semantics
        FROM syscat.columns col
        WHERE col.tabschema = <Parameter name="owner"/>
</miv_columns>

<miv_capabilities type="ResultSet">
<table_supported>true</table_supported>
<view_supported>false</view_supported>
<sequence_supported>false</sequence_supported>
<table_name_filter_supported>true</table_name_filter_supported>
<view_name_filter_supported>false</view_name_filter_supported>
<sequence_name_filter_supported>false</sequence_name_filter_supported>
<business_area_supported>false</business_area_supported>
<business_area_table_supported>false</business_area_table_supported>
<business_area_view_supported>false</business_area_view_supported>
<business_area_sequence_supported>false</business_area_sequence_supported>
<application_owner_supported>true</application_owner_supported>
<table_fklevel_supported>false</table_fklevel_supported>
<reimport_supported>true</reimport_supported>
<data_object_at_leaf_levels>false</data_object_at_leaf_levels>
<multiple_tree_supported>false</multiple_tree_supported>
</miv_capabilities>

</miv>
```

The definition file contains queries to retrieve tables and columns. This MIV file is created using elements defined in an XML schema definition (XSD) file. This file is called `cmi.xsd` and is stored in the *`OWB_HOME`*`/owb/misc` folder.

After you create the platform, it is available under the Databases node in Projects Navigator. You can now connect to a DB2UDB database and import metadata. To connect to DB2, you require the appropriate driver files. For information on the driver file requirements, see "JDBC Connection Drivers for DB2" on page 6-1.

Similarly, you can create a new platform for any database that uses an SQL-based data dictionary and then create a SQL-based CMI mechanism to import metadata from the database.

11.5 Defining Other Platforms

Table 11–9 lists the URL template specification and the driver class for other common platforms.

Table 11–9 JDBC Requirements for Different Platforms

Platform	URL Template	Driver Class
Teradata	`jdbc:teradata://<host>:<port>/<server>`	`com.ncr.teradata.TeraDriver`
Informix	`jdbc:informix-sqli://<host>:<port>:informixserver=<servername>\;database=<dbname>\[\;<property>=<value>...\]`	`com.informix.jdbc.IfxDriver`
Sybase	`jdbc:sybase:Tds:<host>:<port>\[/<database>\]\[?<property>=<value>&<property>=<value>...\]`	`com.sybase.jdbc2.jdbc.SybDriver`
MySQL	`jdbc:mysql://<host>\[:<port>\]/\[<database>\]\[?<property>=<value>\[&<property>=<value>...\]\]`	`com.mysql.jdbc.Driver`

For each database, you must also install the required JDBC driver. The JDBC driver for a database may be shipped with the product, or may require a separate download or purchase.

12

Using Code Templates to Load and Transfer Data

As Oracle Warehouse Builder loads and transforms data from many different database platforms and systems, the technology used to access and load these different data sources must be flexible, extensible, and efficient. Oracle Warehouse Builder solves this problem using code templates (CTs).

CTs are extensible components that enable efficient loading, transformation, or integration of data for a specific data source or target. They can be used in a mapping to perform specific tasks. For example, the task can be extracting data from a source and loading it into a target table.

This chapter contains the following topics:

12.1 About Code Templates

CTs are components of Oracle Warehouse Builder's open connector technology. CTs contain the knowledge required by Oracle Warehouse Builder to perform a specific set of tasks against a specific system or set of systems. Combined with a connectivity layer such as JDBC, CTs define an open connector that performs defined tasks against a system, such as connecting to this system, extracting data from the system, transforming the data, or checking and integrating the data.

Open connectors provide a combination of:

- Connection options such as JDBC
- Correct syntax, such as SQL, for the specific platform
- Control over the creation and deletion of objects such as the temporary and work tables, views, and triggers
- Data processing and transformation strategies
- Data movement options (create target table, insert, delete, update)

There are different types of CTs available for performing different tasks.Oracle Warehouse Builder contains the following types of CTs:

- "Load Code Template": To load data from a source database into a staging area.
- "Integration Code Template": To efficiently transform data from staging area to the target tables, generating optimized native SQL for the given database.

- "Control Code Template": To check for errors in source data.
- "Change Data Capture Code Template": To capture changed data in source objects.
- "Oracle Target Code Template": To deploy and execute a map within a CT framework. It supports the rich set of transformation operators available in mappings including match/merge, name and address and so on.
- "Function Code Template": To deploy functions, packages, and procedures.

With CTs, you thus get templates to perform specific tasks in an efficient manner. For example, to load data from a DB2 database and move it to a SQL Server database using CTs:

1. Create a mapping with a source table and a target table
2. Use a Load CT (LCT) to load data from the DB2 source table
3. Use an Integration CT (ICT) to move data to the SQL Server target table

When you execute this mapping, the loading and transfer of data from DB2 to SQL Server is managed by the LCT and the ICT.

You cannot use CTs in customary mappings. A CT can be used only in a special type of mapping called code template mappings. You can create code template mappings from the Template Mappings node in the Projects Navigator.

To use CT in a mapping, you must encapsulate the objects in a code template mapping into units called *execution unit*. See "Using Code Templates" on page 12-5 for more details.

You can either import predefined CTs from Oracle Data Integrator 10*g* (Knowledge Modules) or create new CTs based on your specific requirements.

12.1.1 Load Code Template

LCTs enable you to load data from a remote data source to the staging area. They support a variety of data sources. These include:

- Oracle
- DB2
- SQL Server
- File

You can also add new data sources. By default, an LCT must be associated with an execution unit that contains a source table object.

12.1.2 Integration Code Template

ICTs are used to integrate data from staging tables into a target database. Based on the loading method and the type of target database, you can use one of the existing set of ICTs. The following rules are applicable when using ICTs.

- The input to an ICT must come from an LCT
- An ICT must be associated with a target database (through an execution unit).

12.1.3 Control Code Template

Control CTs provide data quality checks to ensure data integrity. These include checking for key constraints and other user-defined data loading rules. CCTs are used

to check for data constraints in existing tables and while loading new data into tables. CCTs are associated with execution units in maps. For details about implementing data quality rules in ETL, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

12.1.4 Change Data Capture Code Template

The CDC CTs enable you to capture only those changes that have been made to a data object since the last import. CDC CTs control the change capture data processing and are associated with tables and modules.

12.1.5 Oracle Target Code Template

Oracle Target CTs can contain all of the regular mapping operators such as match/merge, name/address, dimension, and so on.

12.1.6 Function Code Template

A Function CT is used to deploy functions, packages, and procedures. Function CTs are associated with functions, packages, and procedures.

12.1.7 Predefined Code Templates

Oracle Warehouse Builder contains certain predefined CTs that you can directly use in Oracle Warehouse Builder.

These predefined CTs are also installed in the Oracle Warehouse Builder repository and are listed in the Globals Navigator. To view the predefined CTs from the Globals Navigator, expand the Public Code Templates node and then expand the BUILT_IN_CT node. You can directly use these predefined code templates without having to import them.

For a list of all the predefined code templates shipped with Oracle Warehouse Builder, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

Note: The predefined code templates available in Oracle Warehouse Builder were imported from the Knowledge Modules available in Oracle Data Integrator 10g. These code template files are available at *OWB_HOME*/owb/misc/CodeTemplates.

12.2 Working with Code Templates

To use CTs in Oracle Warehouse Builder, perform the following steps:

- Create a CT folder to store a CT, as described in "Creating a Code Template Folder" on page 12-3.
- Create or import a CT, as described in "Creating a Code Template" on page 12-4.
- Create a mapping under Template Mappings, define execution units within the map, and associate CTs to those execution units, as described in "Using Code Templates" on page 12-5.

12.2.1 Creating a Code Template Folder

You must have a container in Oracle Warehouse Builder to hold CTs. This is known as a CT folder.

To create a CT folder:

1. From the Projects Navigator, right-click **Code Templates** and click **New Code Template Folder** to open the Create Code Template Folder dialog box.
2. Provide a name for the new folder and click **OK**.

The newly created CT folder is available in the Projects Navigator. You can also view all the available types of CTs under the newly created folder.

12.2.2 Creating a Code Template

You can create any of the available types of CTs under a CT folder. For some types of CTs such as Load, Integrate, and Change Data Capture, you can either import an existing CT from Oracle Data Integrator or create a new one. For Oracle Target CTs and Function CTs, you must create a new CT. The process to create any type of CT equals as described for LCT.

To create a new LCT:

1. Right-click **Load** and click **New Load CT**.
2. The Create Load Code Template dialog box is displayed.
3. Enter a name and an optional description for the CT and click **OK**.

   The newly created CT is displayed under Load.

   You can now specify the tasks between the Start and End tasks.

> **See Also:** For more information about creating code templates, refer to the section on Knowledge Modules in the *Oracle Data Integrator* documentation set.

12.2.3 Importing a Code Template

To import CTs:

1. Select a Code Template Folder into which you want to import the CTs.
2. Select **File**, then **Import**, and then **Code Template**.

   The Import Code Template Wizard is displayed.
3. Click **Next** to open the Select Files page.
4. Click **Browse** and select the source directory containing the CTs to be imported.

   The available CTs are listed under Directory Files in the Select Files page.
5. Select the CTs and move them to the Files to Import list.
6. Click **Next** to open the Name Code Templates page.

   Here, you can either provide new names for the CTs or retain the current name.
7. Click **Next** to open the Summary page, and then click **Finish**.

Oracle Warehouse Builder automatically inserts imported CTs under the appropriate type. For example, if you import an LCT, it gets added under Load in the Projects Navigator.

Note: You do not have to import predefined CTs, as these are available in the repository and can be used directly.

12.2.4 Tasks in a Code Template

A CT consists of a series of predefined tasks that are broadly classified as:

- **JDBC**: To process JDBC statements.
- **Jython**: To process Jython statements.
- **Runtime API**: To provide access to Oracle Data Integrator 10*g* tools such as OdiOSCommand.
- **Operating System**: To provide access to the operating system.
- **Jacl**: To provide access to native `tcl` script that is generated for a CT mapping.

For example, an LCT that is used to load data from SQL to Oracle consists of the following tasks:

- Start_Task
  1. Validate KM options: Jython task
  2. Drop work table: JDBC task
  3. Create work table: JDBC task
  4. Lock journalized table: JDBC task
  5. Load data: JDBC task
  6. Analyze work table: JDBC task
  7. Cleanup journalized table: JDBC task
  8. Drop work table: JDBC task
- End_Task

12.2.5 Using Code Templates

CTs are used in a code template mapping. To use CTs:

- Create a template mapping module.
- Create a map under the module.
- Use the CT in the mapping.

12.2.5.1 Creating a Template Mappings Module

To create a Template Mappings module:

1. From the Projects Navigator, right-click **Template Mappings** and select **New Mapping Module**.

   The Create Module Wizard is displayed.
2. In the Name and Description page, provide a name and description (optional).
3. In the Connection Details page, either select an existing location or click **Edit** to open the Edit Agent Location dialog box and provide the connection details for the agent. See "Specifying an Agent for Template Mappings" on page 12-6 for more details.
4. In the Summary page, verify the information you provided and click **Finish**.

The newly created template mapping module is now available under the Template Mappings node.

12.2.5.2 Specifying an Agent for Template Mappings

Template mappings get executed in a control center agent. By default, Oracle Warehouse Builder contains a default agent location, which like the default control center, refers to the control center agent on the same host as the control center service.

12.2.5.2.1 Starting and Stopping the Control Center Agent To use the default agent, ensure that the control center agent (CCA) is running.

Windows On Windows, start the CCA instance by navigating to *OWB_HOME*/owb/bin/win32 and running the file ccastart.bat. When you run this file the first time, you are prompted to enter a password for the oc4jadmin user.

To stop the CCA, run the ccashut.bat script located in the *OWB_HOME*/owb/bin/win32 directory. The script prompts you to enter the oc4jadmin user password.

UNIX On UNIX, start the CCA instance by running the file ccastart located in the *OWB_HOME*/owb/bin/unix directory.

To stop the CCA, run the ccashut file located in the *OWB_HOME*/owb/bin/unix directory. The script prompts you to enter the oc4jadmin user password.

12.2.5.3 Edit Agent Location Dialog Box

You must specify the following details in the Edit Agent Location:DEFAULT AGENT dialog box:

- **User**: oc4jadmin
- **Password**: The password set while invoking the ccastart file.
- **Host**: Host where the agent resides
- **Port**: 23791
- **Port Type**: RMI
- **Instance**: Can be left blank
- **Application Name**: Name of the application where the mapping gets deployed. Enter the value jrt to deploy it to the default CCA that is installed with Oracle Warehouse Builder.
- **HTTP Port**: 8888

> **Note:** Ensure that the CCA instance is running before you set the agent location.

12.2.5.4 Creating a Mapping Under the Code Template Mapping

To create a mapping under the code template mapping module:

1. Right-click the Template Mapping module and click **New Mapping**.
2. In the Create Mapping dialog box, provide a name for the mapping.

The newly created mapping is available under Template Mapping module.

12.2.5.5 Using Code Templates in the Mapping

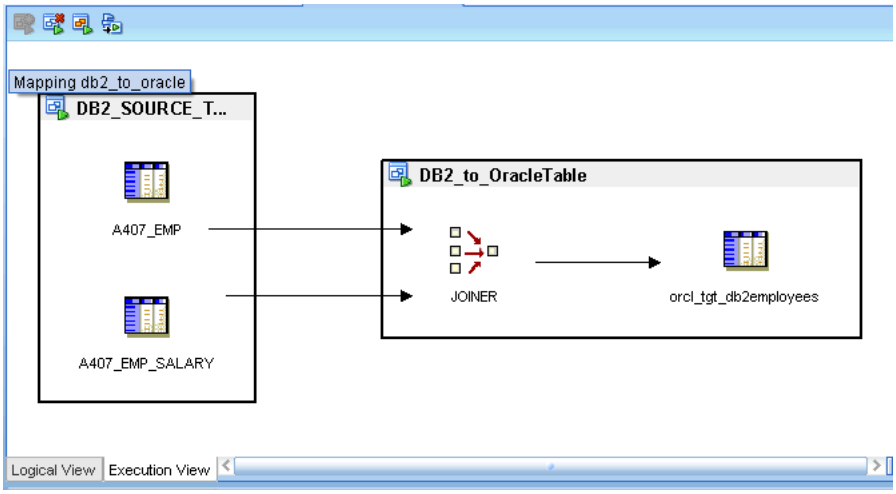
For mappings under a code template mapping, Oracle Warehouse Builder provides two different views to work with, Logical View and Execution View.

In the logical view, you can insert the various components of the mapping. This includes the source and target operators (tables), and the transformation operators. You can also define the data flow from the source to the target, along with the transformation operators that define how the data is transformed before it is loaded into the target.

In the execution view, you must define *execution units* for the components within the mapping. An execution unit is a module that holds the various mapping operators.

For example, if your mapping has a DB2 source table, an expression operator, and an Oracle target table, then you can bind the source table to an execution unit `DB2_SOURCE_TABLE`, and the expression and the target table to another execution unit `DB2_to_OracleTable`, as shown in Figure 12–1.

Figure 12–1 Execution Units in a Map



After you bind the components of the mapping to execution units, you must associate a CT with each of the execution units. In the current example, you must associate `DB2_SOURCE_TABLE` to an LCT that loads data from a DB2 source (`LCT_SQL_TO_ORACLE`). `DB2_to_OracleTable` must be associated with an ICT that retrieves and loads data into an Oracle table (`ICT_ORACLE_INCR_UPD`). For more information about how to bind the components of a mapping to execution units, see *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

Execution units containing certain transformation operators cannot be bound to LCTs and ICTs. See *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for more information about operators not supported in LCTs and ICTs.

For more information about different types of CTs, and about creating and using CTs in mappings, see the *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

13

Importing and Publishing Web Services

In Oracle Warehouse Builder, you can publish core functions such as mappings as a Web service. The Web service can then be imported by other applications. Similarly, you can import Web services published by other applications into Oracle Warehouse Builder. Publishing Oracle Warehouse Builder functions as Web services enhances their visibility and makes them accessible to a larger number of applications because these Web services are created using widely accepted open standards based on XML.

This chapter contains the following topics:

13.1 Defining Web Services

Web services are application components that use XML-based open standards to define how data in an application can be accessed by different clients. These clients may be working on different platforms and residing on different networks. When seen as a Client-Server model, a Web service resides on a server and clients use transport protocols to retrieve the Web service from the server.

Web service data is packaged as messages whose format is defined by the *Simple Object Access Protocol* (SOAP) framework. SOAP is a transport protocol for transmitting XML-based messages over a network. It generally uses HTTP for transmission. Any client that can read and interpret SOAP- based messages can thus access data from a Web service. Oracle Warehouse Builder supports SOAP versions 1.1 and 1.2.

A Web service is built using *Web Services Description Language* (WSDL). A WSDL file is an XML file that describes a Web service and provides details of how to communicate using the Web service. It provides details of the transport protocols and the message format that the Web service uses. The file also contains details of the port, which is a metatag in the WSDL file, that defines the operations or functions of a Web service. Oracle Warehouse Builder supports WSDL version 1.1.

An example of a WSDL file that defines a Web service is as shown:

```
<definitions
     name="PfWebServiceImplService"
     targetNamespace="http://oracle.wh.jrt.sdk.packaging.dbWebService/"
     ............................
    >
    <types>
    </types>
    <message name="runInput"/>
```

```
    <message name="runOutput">
        <part name="return" type="xsd:int"/>
    </message>
    ......................
    <portType name="PfWebServiceImpl">
        <operation name="run">
            <input message="tns:runInput"
xmlns:ns1="http://www.w3.org/2006/05/addressing/wsdl"
                 ns1:Action=""/>
            <output message="tns:runOutput"
xmlns:ns1="http://www.w3.org/2006/05/addressing/wsdl"
                 ns1:Action=""/>
        </operation>
        ............................
    </portType>
    <binding name="PfWebServiceImplSoapHttp" type="tns:PfWebServiceImpl">
        <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="run">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="literal"
namespace="http://oracle.wh.jrt.sdk.packaging.dbWebService/"/>
            </input>
            <output>
                <soap:body use="literal"
namespace="http://oracle.wh.jrt.sdk.packaging.dbWebService/"
                     parts="return"/>
            </output>
        </operation>
        ........................
    </binding>
    <service name="PfWebServiceImplService">
        <port name="HttpSoap11" binding="tns:PfWebServiceImplSoapHttp">
            <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
        </port>
    </service>
</definitions>
```

The WSDL file is automatically generated by Oracle Warehouse Builder when you publish a Web service. The function of the Web service is defined within the `portType` metatag. The `soap:address location` metatag defines the location where the Web service is deployed.

13.2 Publishing Functions as Web Services

Publishing Oracle Warehouse Builder functions as Web services ensures that these functionalities are widely available to clients that use the same XML-based open standards. The following functionalities can be published as Web service:

- Mappings (Code template-based mappings and non code template-based mappings)
- Process Flows
- Transformations (Functions, Procedures, Packages)
- Data Auditors
- Modules and Tables

Note: You can publish only those tables and modules that are imported through native database connection. The modules must have a Change Data Capture code template associated with them. The tables must be identified as being tracked for CDC.

13.2.1 Operations and Parameters Generated in a Web Service

There are three types of parameters that can be passed on to a Web service when it is executed.

- **Custom parameters (custom_params)**: These are input parameters that you might have to enter while executing Web services that are based on PL/SQL mappings, process flows, and data auditors. For example, if you create a Web service based on a mapping that requires you to enter input parameters, then you must provide the input through custom parameters while executing the Web service.
- **System parameters (system_params)**: These are parameters that you can configure for an object in Oracle Warehouse Builder. For example, for a mapping, you can configure its runtime parameters, such as Bulk Size, Audit Level, and so on. These values are then passed on as system parameters while executing a Web service based on the mapping. System parameters are applicable to Web services based on PLS/SQL mappings, process flows, and data auditors.
- **OWB parameters (owb_params)**: These parameters are used in Web services that are based on code template mappings and CDC objects, as these objects are executed in a control center agent.

See *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for more information about the different parameters.

For each of the objects that can be published as a Web service, the following operations and parameters are generated in a Web service.

- **Non Code Template Mappings**, **Process Flows**, **and Data Auditors**: You can use the following operations.
  - `run()`: Run the Web service without any parameters.
  - `runWithSystemParam(string system_params)`: Run the Web service with system parameters.
  - `runWithCustomParam(string custom_params)`: Run the Web service with custom parameters.
  - `runWithParams(string system_params, string custom_params, boolean isAsync)`: Run the Web service with system and custom parameters. If the Async value is set to `true`, then the process is run in the background.
- **Code Template Mappings**: You can use the following operations.
  - `runWithoutParams()`: Run the Web service without any parameters.
  - `runWithOwbParams(string owb_params)`: Run the Web service with Oracle Warehouse Builder parameters.

  The format to specify a parameter is `PARAM=x`. Multiple parameters are separated by comma. For example, the system parameters for a mapping can be specified as:

  ```
  OPERATING_MODE=SET_BASED,AUDIT_LEVEL=NONE
  ```

> **Note:** When you publish a code template-based mapping as a Web service, the code for the mapping is embedded within the Web service. Therefore, when you make changes to such mappings, you must redeploy the Web service.
>
> However, when you publish non code template-based mappings, such as PL/SQL mappings, as a Web service, the code for the mapping is not embedded within the Web service. Instead, the Web service contains only a reference to the code. Therefore, when you make changes to the mapping, you must redeploy only the mapping. You do not have to redeploy the Web service.

- **Transformations**: For Web services based on transformation operators, the operations are the same as those for the function, which means that they share the same function name and parameters.
- **Modules and Tables (CDC based)**: Consistent set CDC uses the following module commands:
  - `start_cdc()`: Set up the CDC infrastructure.
  - `drop_cdc()`: Remove the CDC infrastructure.
  - `subscribe(string subscriber_name)`: Add a subscriber to this CDC.
  - `unsubscribe(string subscriber_name)`: Remove a subscriber from the CDC.
  - `extend_window()`: The Consistency Window is a range of available changes in all the tables of the consistency set for which `insert`, `update`, `delete` are possible without violating referential integrity. The extend window operation computes this window to consider new changes captured since the latest `Extend Window` operation.
  - `lock_subscriber(string subscriber_names)`: Although the extend window is applied to the entire consistency set, subscribers consume the changes separately. This operation performs a subscriber(s) specific snapshot of the changes in the consistency window. This snapshot includes all the changes within the consistency window that have not been consumed yet by the subscriber(s).
  - `unlock_subscriber(string subscriber_names)`: This operation commits the use of the changes that were locked during the `lock subscriber` operations for the subscribers. It should be processed only after all the changes for the subscribers have been processed.
  - `purge_cdc()`: After all subscribers have consumed the changes they have subscribed to, extra entries may still remain in the Capture tables and must be deleted. This is performed by the Purge Journal operation.

For non-consistent set CDC, `Start`, `Drop`, `Subscribe`, and `Unsubscribe` commands are available for both tables and modules. `Lock Subscriber` and `Unlock Subscriber` are available but not used.

13.2.2 Creating a Web Service

Web services are stored in Application Server modules. Creating a Web service involves the following tasks:

- "Creating an Application Server Module"

- "Providing an Application Server Location"
- "Creating the Web Service"

13.2.2.1 Creating an Application Server Module

Web services are created under Application Server modules. To create an Application Server Module:

1. In the Projects Navigator, right-click **Application Servers** and click **New Application Server**.

   The Create Module Wizard is displayed.

2. Click **Next** to proceed to the Name and Description page. Provide a name, and description (optional).
3. Click **Next** to proceed to the Connection Information page. Provide the location details, as described in "Providing an Application Server Location" on page 13-5.
4. Click **Next**, and on the Summary page, click **Finish** after verifying the details.

The newly created module is now available under the Application Servers node. It consists of two sub-nodes: Web Services and Web Service Packages.

Application Server Modules can also be created under Public Application Servers from the Globals Navigator.

13.2.2.2 Providing an Application Server Location

You can provide the location information while creating an Application Server module or while deploying a Web service.

A Web service can be deployed to the following locations:

- OC4J Standalone
- Oracle Application Server

Before providing the location information, ensure that the control center agent (CCA) is running. See "Starting and Stopping the Control Center Agent" on page 12-6 for more information about starting and stopping a control center agent on Windows and UNIX systems.

To create a location:

1. On the Connection Information page, click **Edit** to open the Edit Agent Location dialog box and provide the following details:
   - **Username**: The `oc4jadmin` user or application server user with sufficient privileges.
   - **Password**: Password set for `oc4jadmin` user while running `ccastart` file.
   - **Host**: The host where OC4J or the application server resides.
   - **Port**: The port number to access the server. The default value is 23791.
   - **Port Type**: RMI Port for OC4J Standalone, OPMN for Oracle Application Server.
   - **Instance**: OC4J instance name. To be provided only for Oracle Application Server.
   - **Application Name**: The default value is set to `jrt`.

- **HTTP Port**: This port is used if the Web service is accessed through a browser. The default value is set to 8888.

2. Click **Test Connection** to verify the connection details.

You cannot deploy a Web service without providing a valid deployment location. If you try to deploy a Web service without providing the deployment location, Oracle Warehouse Builder displays the Edit Agent Location dialog box. Provide the connection details before attempting to deploy the Web service.

13.2.2.3 Creating the Web Service

Within the Application Server module, you can create a standalone Web service or create a Web service under a Web Service Package.

To create a Web Service Package:

1. Right-click **Web Services Packages** and click **New Web Service Package**.

   The Create Web Service Package dialog box is displayed.

2. Provide a name and a description (optional), and click **OK**.

   The newly created Web service package is now available under the Web Services Package node.

   If you selected the **Proceed to Web Service Wizard** option, the Create Web Service Wizard is displayed and you can create individual Web services.

To create a Web service:

1. Right-click **Web Services** and click **New Web Service**.

   The Create Web Service Wizard is displayed.

2. Click **Next** to proceed to the Name and Description page. Provide a name and description (optional).

3. Click **Next** to proceed to the Define Implementation page. Select the object is to be published as a Web service.

4. Click **Next** to proceed to the Review Specification page.

   Click **View Source** to view the WSDL file that is generated for the Web service.

5. Click **Finish** to create the Web service.

The newly created Web service is now available under the Web Services node.

13.3 Importing External Web Services

You can import Web services using the Public Application Servers node, which is available in the Globals Navigator.

To import a Web service, you must first create an application server module and then import the Web service into this module.

To create an Application Server module:

1. On the Globals Navigator, right-click **Public Application Servers** and select **New Application Server**.

   The Create Module dialog box is displayed.

2. Provide a name for the module.

The newly created module is now available under the Public Application Servers node.

To import a Web service:

1. Right-click the Application Server node and select **New Web Service**.

   The Create Web Service wizard is displayed.

2. On the Name and Description page, provide the location details of the Web service.

   You can import a Web service that is deployed to a local system. You can provide the location details using the URL format or browse for the location.

After importing a Web service, you can execute it to run its function. See *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for more information about executing Web services.

See *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for more information about creating, publishing, and executing Web services.

Index

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

W