Oracle® Fusion Middleware

Metadata Repository Builder's Guide for Oracle Business Intelligence Enterprise Edition

12*c* (12.2.1)

E57378-02

December 2015

Explains how to build an Oracle Business Intelligence metadata repository, how to set up and connect to data sources, and how to build the Physical layer, Business Model and Mapping layer, and Presentation layer. Includes how to use the multiuser development environment, along with a Logical SQL reference.

ORACLE

Oracle Fusion Middleware Metadata Repository Builder's Guide for Oracle Business Intelligence Enterprise Edition, 12*c* (12.2.1)

E57378-02

Primary Author: Stefanie Rhone

Contributors: Oracle Business Intelligence development, product management, and quality assurance team

Contents

2 Before You Begin

3 Setting Up and Using the Multiuser Development Environment

4 Using a Source Control Management System for Repository Development

5 Importing Metadata and Working with Data Sources

6 Working with ADF Data Sources

7 Setting Up Database Objects and Connection Pools

8 Working with Physical Tables, Cubes, and Joins

9 Working with Logical Tables, Joins, and Columns

10 Working with Logical Dimensions

11 Managing Logical Table Sources (Mappings)

12 Creating and Maintaining the Presentation Layer

13 Creating and Persisting Aggregates for Oracle BI Server Queries

14 Applying Data Access Security to Repository Objects

15 Completing Oracle BI Repository Setup

16 Setting Up Data Sources on Linux and UNIX

17 Managing Oracle BI Repository Files

18 Using Expression Builder and Other Utilities

19 Using Variables in the Oracle BI Repository

A Managing the Repository Lifecycle in a Multiuser Development Environment

B MUD Case Study: Eden Corporation

C Logical SQL Reference

D Merge Rules

E Deleting Unwanted Objects from the Repository

F Data Types Supported by Oracle BI Enterprise Edition

G Exchanging Metadata with Databases to Enhance Query Performance

H XML Schema Files for ADF Mapping Customizations

I Administration Tool Keyboard Shortcuts

J Administration Tool User Interface Components

Preface

The Oracle Business Intelligence Foundation Suite is a complete, open, and integrated solution for all enterprise business intelligence needs, including reporting, ad hoc queries, OLAP, dashboards, scorecards, and what-if analysis. The Oracle Business Intelligence Foundation Suite includes Oracle Business Intelligence Enterprise Edition.

Oracle Business Intelligence Enterprise Edition (Oracle BI EE) is a comprehensive set of enterprise business intelligence tools and infrastructure, including a scalable and efficient query and analysis server, an ad-hoc query and analysis tool, interactive dashboards, proactive intelligence and alerts, and an enterprise reporting engine.

The components of Oracle BI EE share a common service-oriented architecture, data access services, analytic and calculation infrastructure, metadata management services, semantic business model, security model and user preferences, and administration tools. Oracle BI EE provides scalability and performance with data-source specific optimized request generation, optimized data access, advanced calculation, intelligent caching services, and clustering.

This guide contains information about building an Oracle Business Intelligence metadata repository and includes topics on setting up and connecting to data sources, building the Physical layer, Business Model and Mapping layer, and Presentation layer, how to use the multiuser development environment, and a Logical SQL reference.

Audience

This document is intended for anyone who intends to design and build a metadata repository using the Oracle Business Intelligence Administration Tool, such as a Business Intelligence strategist, metadata provider, or ETL developer.

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 that have purchased support 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.

Related Documentation and Other Resources

See the Oracle Business Intelligence documentation library for a list of related Oracle Business Intelligence documents.

In addition:

- Go to the Oracle Learning Library for Oracle Business Intelligence-related online training resources.
- Go to the Product Information Center support note (Article ID 1267009.1) on My Oracle Support at `https://support.oracle.com`.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

New Features for Oracle BI Metadata Repository Builders

This preface describes changes to metadata repository features for Oracle Business Intelligence Enterprise Edition 12*c* Release 1 (12.2.1).

This preface contains the following topics:

New Features for Oracle BI EE 12*c* Release (12.2.1)

New metadata repository features in Oracle BI EE 12*c* Release (12.2.1) include:

Logical Level Sequence Numbers for Time Dimensions

The Sequence Numbers tab has been added to the Logical Level dialog. The new tab allows you to add absolute or relative sequence numbers to time dimensions. These mappings provides direct column references in the Time dimension table, which creates a query that is easier for Oracle BI Server to execute against the data source. See "Adding Sequence Numbers to a Time Dimension's Logical Level" for more information.

DISPLAY | SORTKEY Syntax Supported in the SQL ORDER BY Expression

The Oracle BI Server now accepts the `DISPLAY` and `SORTKEY` keywords in the SQL `ORDER BY` expression. You can use the `DISPLAY` keyword to override a logical column's assigned sort order column. For more information, see "ORDER BY Clause Syntax."

Oracle Database Fast Application Notification and Fast Connection Failover Supported by Oracle BI Server

The Oracle BI Server supports the Fast Application Notification (FAN) event and Fast Connection Failover (FCF) Oracle Database configuration. Fast Connection Failover enables quick failover when the data source's Oracle database is not available. See "Oracle Database Fast Application Notification and Fast Connection Failover" for more information.

Generate Fragmented Aggregates in Aggregate Persistence

The aggregate persistence functionality has been enhanced to generate fragmented aggregates from a manually written aggregate specification. You can generate fragmented aggregates by adding a Where clause to the Logical SQL query's Create statement. See "Writing the Create Aggregates Specification" for more information.

New Command Line Utilities

Several command line utilities have been added. See the following topics for more information:

- Download Repository Command
- Upload Repository Command
- List Connection Pool Command
- Update Connection Pool Command
- Rename Application Role Command
- Delete Application Role Command
- Rename Users Command
- Delete Users Command
- List Repository Variables Command
- Update Repository Variables Command

1

Introduction to Building Your Metadata Repository

This chapter explains how to plan and design your Oracle Business Intelligence metadata repository, including how to plan your business model, how to work with the physical content for your business model, and general repository design guidelines.

To effectively plan and build your metadata repository, you need to have experience with SQL queries and be familiar with reporting and analysis. You should also have experience with industry-standard data warehouse modeling practices, and be familiar with general relational entity-relationship modeling.

This chapter contains the following topics:

About Oracle BI Server and Oracle BI Repository Architecture

The architecture of the Oracle BI Server and the Oracle BI repository provides a layer of abstraction that lets users send simple Logical SQL queries against complex federated data sources.

This section contains the following topics:

About Oracle BI Server Architecture

The Oracle BI Server is an Oracle Business Intelligence component that processes user requests and queries underlying data sources. The Oracle BI Server maintains the logical data model and provides client access to this model through ODBC.

The Oracle BI Server uses the metadata in the Oracle BI repository to perform the following two tasks:

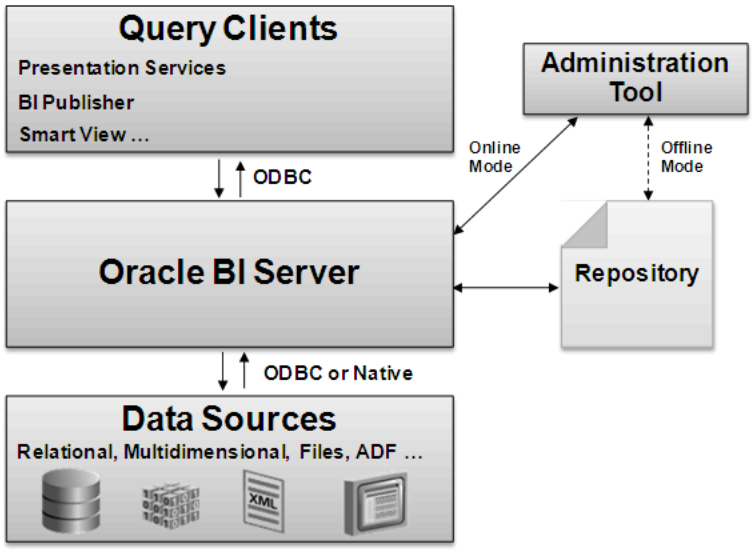
- Interpret Logical SQL queries and write corresponding physical queries against the appropriate data sources
- Transform and combine the physical result sets and perform final calculations

The Oracle BI Server connects to the underlying data sources through either ODBC or through native APIs, such as OCI for Oracle Database.

The Administration Tool client is a Windows application that you can use to create and edit your Oracle BI repository. The Administration Tool can connect directly to the repository in offline mode, or it can connect to the repository through the Oracle BI Server. Some options are only available in online mode. See "Using Online and Offline Repository Modes" for more information.

Figure 1–1 shows how the Oracle BI Server interacts with query clients, data sources, the Oracle BI repository, and the Administration Tool.

Figure 1–1 Oracle BI Server Architecture



Example 1–1 shows how the Oracle BI Server interprets and converts Logical SQL queries.

Example 1–1 Logical Requests Are Transformed Into Complex Physical Queries

Assume the Oracle BI Server receives the following simple client request:

```
SELECT
"D0 Time"."T02 Per Name Month" saw_0,
"D4 Product"."P01 Product" saw_1,
"F2 Units"."2-01 Billed Qty (Sum All)" saw_2
FROM "Sample Sales"
ORDER BY saw_0, saw_1
```

The Oracle BI Server can then convert the Logical SQL query into a sophisticated physical query, as follows:

```
WITH SAWITH0 AS (
select T986.Per_Name_Month as c1, T879.Prod_Dsc as c2,
   sum(T835.Units) as c3, T879.Prod_Key as c4
from
   Product T879 /* A05 Product */ ,
```

```
   Time_Mth T986 /* A08 Time Mth */ ,
   FactsRev T835 /* A11 Revenue (Billed Time Join) */
where ( T835.Prod_Key = T879.Prod_Key and T835.Bill_Mth = T986.Row_Wid)
group by T879.Prod_Dsc, T879.Prod_Key, T986.Per_Name_Month
)
select SAWITH0.c1 as c1, SAWITH0.c2 as c2, SAWITH0.c3 as c3
from SAWITH0
order by c1, c2
```

About Layers in the Oracle BI Repository

An Oracle BI repository has the following layers:

- **Physical layer.** This layer defines the objects and relationships that the Oracle BI Server needs to write native queries against each physical data source. You create this layer by importing tables, cubes, and flat files from your data sources.

  Separating the logical behavior of the application from the physical model provides the ability to federate multiple physical sources to the same logical object, enabling aggregate navigation and partitioning, as well as dimension conformance and isolation from changes in the physical sources. This separation also enables the creation of portable BI Applications.

- **Business Model and Mapping layer.** This layer defines the business or logical model of the data and specifies the mapping between the business model and the physical schemas. This layer determines the analytic behavior seen by users, and defines the superset of objects and relationships available to users. It also hides the complexity of the source data models.

  Each column in the business model maps to one or more columns in the Physical layer. At run time, the Oracle BI Server evaluates Logical SQL requests against the business model, and then uses the mappings to determine the best set of physical tables, files, and cubes for generating the necessary physical queries. The mappings often contain calculations and transformations, and might combine multiple physical tables.

- **Presentation layer.** This layer provides a way to present customized, secure, role-based views of a business model to users. It adds a level of abstraction over the Business Model and Mapping layer and provides the view of the data seen by users building requests in Presentation Services and other clients.

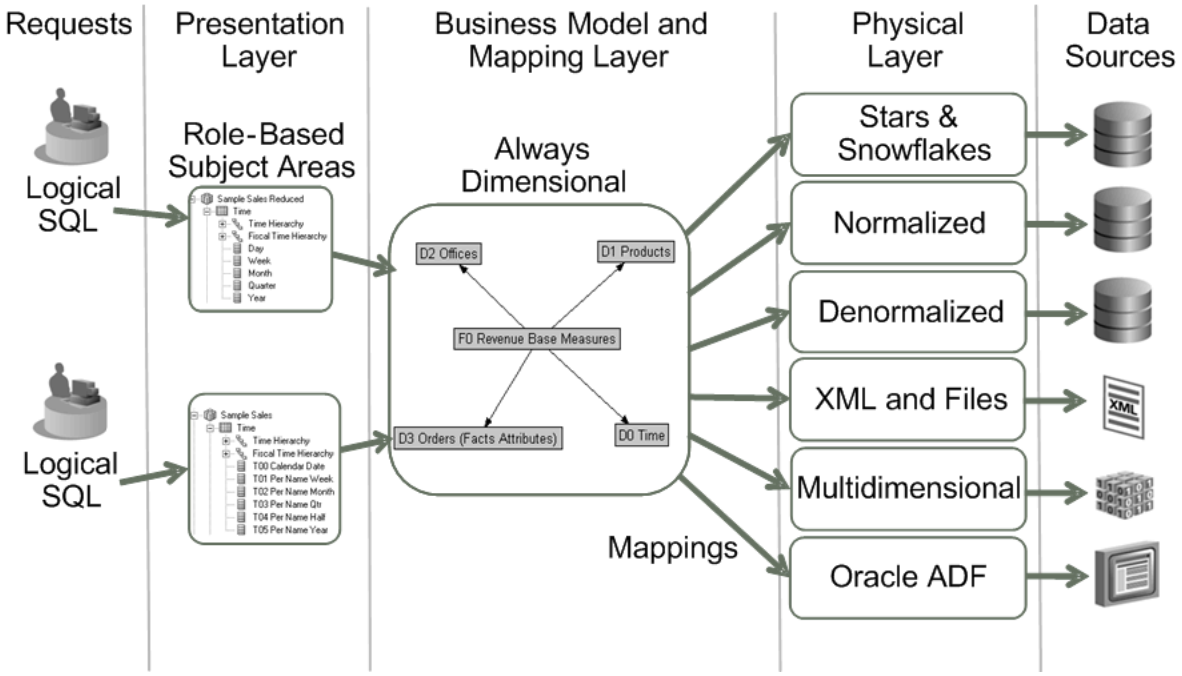
  You can create multiple subject areas in the Presentation layer that map to a single business model, effectively breaking up the business model into manageable pieces.

Before you build any repository layers in the Administration Tool, it is important to create a high-level design of the Business Model and Mapping layer based on the analytic requirements of your users. After you have a conceptual design to work toward, you can then build your metadata objects.

The typical order is to create the Physical layer objects first, the Business Model and Mapping layer objects next, and the Presentation layer objects last. However, you can work on each layer at any stage. After you complete all three layers, you can set up security when you are ready to begin testing the repository.

Figure 1–2 shows how a Logical SQL query traverses the layers of an Oracle BI repository.

Figure 1–2 Logical SQL Query Traversing the Layers in an Oracle BI Repository



Note that a single Oracle BI repository can contain two or more independent semantic models, rather than a single, integrated, enterprise-wide model. A semantic model consists of one business model, its related objects in the Presentation and Physical layers, and additional related objects like variables, initialization blocks, and application roles. A semantic model is also known as a Common Enterprise Information Model.

See also Figure A–4 for a visual representation of multiple semantic models.

Planning Your Business Model

Planning your business model is the first step in developing a usable data model for decision support. After you have followed the planning guidelines in this section, you can begin to create your repository.

Analyzing Your Business Model Requirements

Your first task is to thoroughly understand your business model requirements. You must first understand what business model you want to build before you can determine what the Physical layer needs to have in it.

In a decision support environment, the objective of data modeling is to design a model that presents business information in a manner that parallels business analysts' understanding of the business structure. A successful model allows the query process to become a natural process by enabling analysts to structure queries in the same intuitive fashion as they would ask business questions. This model must be one that business analysts inherently understand and that answers meaningful questions correctly.

Unlike visual SQL tools such as Oracle BI Publisher, the business model defines the analytic behavior of your BI application. In contrast, the Physical layer only provides the components used to assemble a physical query mapped from business model logic.

This requires breaking down your business into several components to answer the following questions:

- What kinds of business questions are analysts trying to answer?
- What are the measures required to understand business performance?
- What are all the dimensions under which the business operates? Or, in other words, what are the dimensions used to break down the measurements and provide headers for the reports?
- Are there hierarchical elements in each dimension, and what types of relationships define each hierarchy?

After you have answered these questions, you can identify and define the elements of your business model.

Identifying the Content of the Business Model

To determine what content to include in your business model, you must first identify the logical columns on which users need to query. Then, you must identify each column's role as either a measure column or a dimensional attribute. Finally, you need to arrange the logical columns in a dimensional model based on the relevant roles, relationships between columns, and logic.

Businesses are analyzed by relevant dimensional criteria, and the business model is developed from these relevant dimensions. These dimensional models form the basis of the valid business models to use with the Oracle BI Server.

Although not all dimensional models are built around a star schema, it is a best practice to use a simple star schema in the business model layer. In other words, the dimensional model should represent some measurable facts that are viewed in terms of various dimensional attributes.

After you analyze your business model requirements, you need to identify the specific logical tables and hierarchies that you need to include in your business model.

This section contains the following topics:

- Identifying Logical Fact Tables
- Identifying Logical Dimension Tables
- Identifying Dimensions
- Identifying Lookup Tables

Identifying Logical Fact Tables

Logical fact tables in the Business Model and Mapping layer contain measures that have aggregations built into their definitions. Logical fact tables are different from physical fact tables in relational models, which instead define facts at the lowest grain of the table.

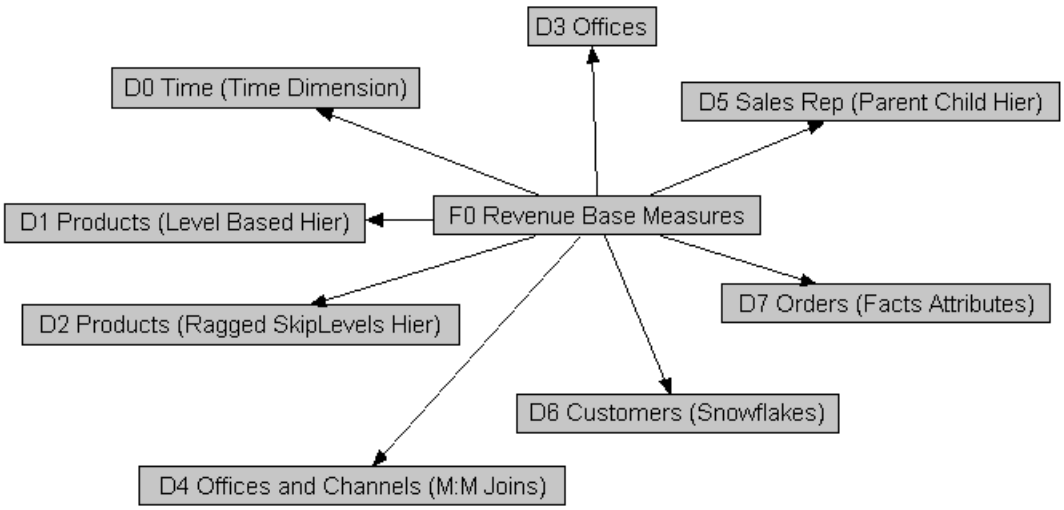
Measures aggregated from facts must be defined in a logical fact table. Measures are typically calculated data such as dollar value or quantity sold, and they can be specified in terms of dimensions. For example, you might want to determine the sum of dollars for a given product in a given market over a given time period.

Each measure has its own aggregation rule such as `SUM`, `AVG`, `MIN`, or `MAX`. A business might want to compare values of a measure and need a calculation to express the comparison. Also, aggregation rules can be specific to particular dimensions. The Oracle BI Server lets you define complex, dimension-specific aggregation rules.

You do not explicitly label tables in the Business Model and Mapping layer as fact tables or dimension tables. Rather, the Oracle BI Server treats tables at the "one" end of a join as dimension tables, and tables at the "many" end of a join as fact tables.

Figure 1–3 illustrates the many-to-one joins to a fact table in a Business Model Diagram. In the diagram, all joins have an arrow (indicating the one side) pointing away from the Fact-Pipeline table; no joins are pointing to it. For an example of this in a business model, open a repository in the Administration Tool, right-click a fact table, and select **Business Model Diagram > Whole Diagram**.

Figure 1–3 Diagram of Fact Table Joins



Identifying Logical Dimension Tables

A business uses facts to measure performance by well-established dimensions, for example, by time, product, and market. Every dimension has a set of descriptive attributes. Dimension tables contain attributes that describe business entities (such as Customer Name, Region, Address, Country and so on). Dimension tables also contain primary keys that identify each member. Unlike logical fact tables, which are different from physical fact tables in relational models, logical dimension tables behave very much like relational dimension tables.

Dimension table attributes provide context to numeric data, such as being able to categorize Service Requests. Attributes stored in this dimension might include Service Request Owner, Area, Account, Priority, and so on.

Dimensions in the business model are conformed dimensions. In other words, even if a particular data source has five different instances of a particular Customer table, the business model should only have one table. To achieve this, all five physical source instances of Customer are mapped to a single Customer logical table, with transformations in the logical table source as necessary. Conformed dimensions hide the complexity of the Physical layer from users and enable you to combine data from multiple fact sources at different grains. They also enable you to federate multiple data sources.

Also note that dimension and level keys in the business model should be business keys rather than generated surrogate keys. In other words, use "Customer Name" with values like "Oracle" instead of "Customer Key" with values like "1076823." Using business keys in the business model ensures that all sources for that dimension can be conformed to the same logical dimension table with the same logical key and level key.

Although generated surrogate keys can still exist in the Physical layer, where they are useful for their query performance advantages on joins, you typically do not have surrogate key columns in the Business Model and Mapping layer at all.

Identifying Dimensions

Dimensions are categories of attributes by which the business is defined. Common dimensions are time periods, products, markets, customers, suppliers, promotion conditions, raw materials, manufacturing plants, transportation methods, media types, and time of day. Within a given dimension, there may be many attributes. For example, the time period dimension can contain the attributes day, week, month, quarter, and year. Exactly what attributes a dimension contains depends on the way the business is analyzed.

Dimensions typically contain hierarchies, which are sets of top-down relationships between members within a dimension. There are two types of hierarchies: level-based hierarchies (structure hierarchies), and parent-child hierarchies (value hierarchies). Level-based hierarchies are those in which members of the same type occur only at a single level, while members in parent-child hierarchies all have the same type. Oracle Business Intelligence also supports a special type of level-based dimension, called a time dimension, that provides special functionality for modeling time series data.

In level-based hierarchies, levels roll up from lower level to higher level; for example, months can roll up into a year. These rollups occur over the hierarchy elements and span natural business relationships.

In parent-child hierarchies, the business relationships occur between different members of the same real-world type, such as the manager-employee relationship in an organizational hierarchy tree. Parent-child hierarchies do not have explicitly named levels. There is no limit to the number of implicit levels in a parent-child hierarchy.

To define your hierarchies, you define the "contains" relationships in your business (geographical, product, time, and so on) to drive rollup aggregations in all calculations, as well as drill-down navigation in reports and dashboards. For example, if month rolls up into year and an aggregate table exists at the month level, that table can be used to answer questions at the year level by adding up all of the month-level data for a year.

It is important to use the right type of hierarchy for your needs. To determine which type to use, consider the following:

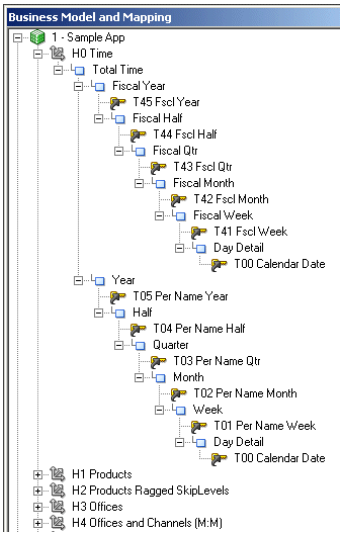
- Are all the members of the same type (such as employee, assembly, or account), or are they of different types that naturally fall into levels (such as year-quarter-month, continent-country-state/province, or brand-line-product)?
- Do members have the same set of attributes? For example, in a parent-child hierarchy like Employees, all members might have a Hire Date attribute. In a level-based hierarchy like Time, however, the Day type might have a Holiday attribute, while Month does not.
- Are the levels fixed at design time (year-quarter-month), or can run-time business transactions add or subtract levels? For example, a level could be added when the current lowest-level employee hires a subordinate, who then becomes the new lowest level.
- Are there constraints in your primary data source that require a certain hierarchy type? If your primary data source is modeled in one way or the other, you might need to use the same hierarchy type in your business model, regardless of other factors.

See Chapter 10, "Working with Logical Dimensions" for more information.

About Dimensions with Multiple Hierarchies Sometimes, dimensions can contain multiple hierarchies. For example, time dimensions often have one hierarchy for the calendar year, and another hierarchy for the fiscal year. Note that dimensions with multiple hierarchies must always end with the same leaf table.

Figure 1–4 shows a dimension with multiple hierarchies in the Business Model and Mapping layer of the Administration Tool.

Figure 1–4 Dimension with Multiple Hierarchies



Identifying Lookup Tables

When you need to display translated field information from multilingual schemas, you create a logical lookup table that corresponds to a lookup table in the Physical layer. A lookup table stores multilingual data corresponding to rows in the base tables. Before you can use a particular logical lookup table, you must designate it as a lookup table in the General tab of the Logical Table dialog. See "Localizing Oracle Business Intelligence" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about localization and lookup tables.

In addition to localization, lookup tables can be used any time you need to display one set of values to users, while using a different, corresponding set of values in the physical query. If necessary, the human-readable value can be looked up in a completely different data source.

Identifying the Data Source Content for the Physical Layer

After you have determined the requirements for your business model, you can look at what data source content you need in the Physical layer. Unlike the Business Model and Mapping layer, which is always dimensional, each physical model mirrors the shape of the source (for example, normalized, cube, and so on).

This section contains the following topics:

- About Types of Physical Schemas in Relational Data Sources
- About Cubes in Multidimensional Data Sources
- Identifying the Data Source Table Structure

About Types of Physical Schemas in Relational Data Sources

You can successfully model any physical schema in the Oracle BI repository, regardless of its type, because the model of any physical source can be broken down into overlapping subsets that are dimensional.

There are four types of physical schemas (models):

- **Star Schemas.** A star schema is a set of dimensional schemas (stars) that each have a single fact table with foreign key join relationships to several dimension tables. When you map a star to the business model, you first map the physical fact columns to one or more logical fact tables. Then, for each physical dimension table that joins to the physical fact table for that star, you map the physical dimension columns to the appropriate conformed logical dimension tables.
- **Snowflake Schemas.** A snowflake schema is similar to a star schema, except that each dimension is made up of multiple tables joined together. Like star schemas, you first map the physical fact columns to one or more logical tables. Then, for each dimension, you map the snowflaked physical dimension tables to a single logical table. You can achieve this by either having multiple logical table sources, or by using a single logical table source with joins.
- **Normalized Schemas.** Normalized schemas distribute data entities into multiple tables to minimize data storage redundancy and optimize data updates. Before mapping a normalized schema to the business model, you need to understand how the distributed structure can be understood in terms of facts and dimensions.

  After analyzing the structure, you pick a table that has fact columns and then map the physical fact columns to one or more logical fact tables. Then, for each dimension associated with that set of physical fact columns, you map the distributed physical tables containing dimensional columns to a single logical table. Like with snowflake schemas, you can achieve this by having multiple logical table sources, or by using a single logical table source with joins. Mapping normalized schemas is an iterative process because you first map a certain set of facts, then the associated dimensions, and then you move on to the next set of facts.

  Note that when a single physical table has both fact and dimension columns, you may need to create a physical alias table to handle the multiple roles played by that table.
- **Fully Denormalized Schemas.** This type of dimensional schema combines the facts and dimensions as columns in one table (or flat file), and is mapped differently than other types of schemas. When you map a fully denormalized schema to the star-shaped business model, you map the physical fact columns from the single physical fact table to multiple logical fact tables in the business model. Then, you map the physical dimension columns to the appropriate conformed logical dimension tables.

About Cubes in Multidimensional Data Sources

Cubes are made up of measures and organized by dimensions. Because they are already dimensional, each cube maps easily to the logical fact and dimension tables in the business model.

Note the following about measures and dimensions:

- Measures in multidimensional cubes and relational fact columns both map to logical measures in the Business Model and Mapping layer. However, measures in multidimensional cubes already include calculations and aggregations, unlike

relational fact columns, which require the calculations and aggregations to be applied in the business model. Rather than treating cubes like relational sources, the Oracle BI Server can take advantage of the pre-aggregated data and powerful calculations in the cube.

- Multidimensional physical objects and relational physical objects both map to logical dimensions in the Business Model and Mapping layer. However, dimensional and hierarchical semantics are already built into multidimensional data sources, unlike relational sources. The Oracle BI Server can take advantage of the more complete hierarchy and dimensional support in the cube, both during import and at query time.

Identifying the Data Source Table Structure

The Administration Tool provides an interface to map logical tables to the underlying physical tables in your data sources. Before you can map the tables, you need to identify the contents of the physical data sources as it relates to your business model. To do this correctly, you need to identify the following contents of the physical data source:

- Identify the contents of each table
- Identify the detail level for each table
- Identify the table definition for each aggregate table. This lets you set up aggregate navigation. The following detail is required by the Oracle BI Server:
  - The columns by which the table is grouped (the aggregation level)
  - The type of aggregation (`SUM`, `AVG`, `MIN`, `MAX`, or `COUNT`)

  For information on how to set up aggregate navigation in a repository, see Chapter 11.
- Identify the contents of each column
- Identify how each measure is calculated
- Identify the joins defined in the database

To acquire this information about the data, you could refer to any available documentation that describes the data elements created when the data source was implemented, or you might need to spend some time with the DBA for each data source to get this information. To fully understand all the data elements, you might also need to ask people in the organization who are users of the data, owners of the data, or the application developers of applications that create the data.

Guidelines for Designing a Repository

After analyzing your business model needs and identifying the database content that your business requires, you can complete your repository design. This section contains some design best practices that can help you implement a more efficient repository.

Typically, you should not make performance tuning changes until you have imported and tested your databases. These tasks are performed during the final steps in completing the setup of your repository. For more information about these final steps, see Chapter 15.

This section contains the following topics:

- General Tips for Working on the Repository

- Design Tips for the Physical Layer
- Design Tips for the Business Model and Mapping Layer
- Design Tips for the Presentation Layer

General Tips for Working on the Repository

Follow these recommended design strategies for structuring your Oracle BI repository:

- If you work in online mode, save backups of the online repository before and after every completed unit of work. If needed, use **Copy As** on the **File** menu to make an offline copy containing the changes.
- Use the Physical Diagrams in the Administration Tool to verify sources and joins.
- Decide whether you want to set up row-level security controls in the database, or in the repository. This decision determines whether you share connection pools and cache, and may limit the number of separate source databases you want to include in your deployment. See Chapter 14, "Applying Data Access Security to Repository Objects" for more information.

Most dialogs in the Administration Tool have Help that provides information about how to complete a task. To access a help topic, click the **Help** button in a dialog, or select **Help** from some menus.

Design Tips for the Physical Layer

The Physical layer contains information about the physical data sources. The most common way to create the schema in the Physical layer is by importing metadata from databases and other data sources. If you import metadata, many of the properties are configured automatically based on the information gathered during the import process. You can also define other attributes of the physical data source, such as join relationships, that might not exist in the data source metadata.

The Physical layer can contain data sources of many different types, including multidimensional, relational, and XML sources. See "System Requirements and Certification" for information about supported databases.

For each data source, there is at least one corresponding connection pool. The connection pool contains data source name (DSN) information used to connect to a data source, the number of connections allowed, timeout information, and other connectivity-related administrative details. See "About Connection Pools" for more information.

The following is a list of tips to use when designing the Physical layer:

- It is recommended that you use table aliases frequently in the Physical layer to eliminate extraneous joins, including the following:
  - Eliminate all physical joins that cross dimensions (inter-dimensional circular joins) by using aliases.
  - Eliminate all circular joins (intra-dimensional circular joins) in a logical table source in the Physical Model by creating physical table aliases.

    For example, suppose you have a Customer table that can be used to look up ship-to addresses, and using a different join, to look up bill-to addresses. You can avoid the circular joins by creating an alias table in the Physical layer so that there is one table instance for each purpose, with separate joins.

If you do not eliminate circular joins, you might get erroneous report results. In addition, query performance might be negatively impacted.

- It is recommended that you use alias tables to create separate physical joins when you need the join to be an inner join in one logical table source and an outer join in another logical table source.
- You might import some tables into the Physical layer that you might not use right away, but that you do not want to delete. To identify tables that you do want to use right away in the Business Model and Mapping layer, you can assign aliases to physical tables before mapping them to the business model layer.

> **Note:** To have the name of a table to which you assigned an alias appear, select **Tools**, then select **Options**, then select **General**, and then select **Display original name for alias in diagrams**.

- Use an opaque view (a Physical layer table that consists of a `SELECT` statement) only if there is no other solution to your modeling problem. Ideally, you should create a physical table, or alternatively a materialized view. Opaque views prevent the Oracle BI Server from generating its own optimized SQL, because they contain fixed SQL statements that are sent to the underlying data source.

Design Tips for the Business Model and Mapping Layer

The Business Model and Mapping layer organizes information by business model. In this layer, each business model is effectively a separate application.

The logical schema defined in each business model must contain at least two logical tables. Relationships need to be defined between all the logical tables. See "About Layers in the Oracle BI Repository" for more information about business model schemas. See Chapter 9 for more information about setting up the Business Model and Mapping layer.

The following is a list of tips to use when designing the Business Model and Mapping layer:

- Create the business model with one-to-many logical joins between logical dimension tables and the fact tables wherever possible. The business model should ideally resemble a simple star schema in which each fact table is joined directly to its dimensions.
- Every logical fact table must join to at least one logical dimension table. Note that when the source is a fully denormalized table or flat file, you must map its physical fact columns to one or more logical fact tables, and its physical dimension columns to logical dimension tables.
- Every logical dimension table should have a dimensional hierarchy associated with it. This rule holds true even if the hierarchy has only one level, such as a scenario dimension {actual, forecast, plan}.
- When creating level-based measures, make sure that all appropriate fact sources map to the appropriate level in the hierarchy using aggregation content. You set up aggregation content in the Levels tab of the Logical Column dialog for the measure. Note that this is different from the Content tab of the Logical Table Source dialog, which is used to specify the grain of the source tables to which it maps.

You only need to set up aggregation content in the Levels tab of the Logical Column dialog for level-based measures. For measures that are not level based, leave the Logical Level field blank.

- Typically, logical fact tables should not contain any keys. The only exception is when you need to send Logical SQL queries against the Oracle BI Server from a client that requires keys. In this case, you need to expose those keys in both the logical fact tables, and in the Presentation layer.
- Normally, all columns in logical fact tables are aggregated measures, except for keys required by external clients, or dummy columns used as a divider. Other non-aggregated columns should instead exist in a logical dimension table.
- In some situations, you might want to have multiple logical fact tables in a single business model. For Logical SQL queries, the multiple logical fact tables behave as if they are one table.

  Reasons to have multiple logical fact tables include:

  – To assign projects. See "Setting Up Projects" for more information.

  – To automatically create small subject areas in the Presentation layer. See "Automatically Creating Subject Areas Based on Logical Stars and Snowflakes" for more information.

  – For organization and simplicity of understanding.

  Unlike relational fact tables, logical fact tables can contain measures of different grains. Because of this, grain is not a reason to split up logical fact tables.
- You can define calculations in either of the following ways:

  – Before the aggregation, in the logical table source. For example:

  ```
  sum(col_A *( col_B))
  ```

  – After the aggregation, in a logical column derived from two other logical columns. For example:

  ```
  sum(col A) * sum(col B)
  ```

  You can also define post-aggregation calculations in Answers or in Logical SQL queries.
- If you plan to use Oracle Scorecard and Strategy Management, it is a best practice to implement at least one time dimension in the Oracle BI repository you are using for your KPIs. This action is necessary because you use KPIs in scorecards to measure progress and performance over time. Note that an individual scorecard automatically includes any dimension used by KPIs in that scorecard.
- Aggregate sources should be created as separate logical table sources. For fact aggregates, use the Content tab of the Logical Table Source dialog to assign the correct logical level to each dimension.
- Each dimension level in a hierarchy must have a unique level key. Also, each logical dimension table must have a unique primary key. Normally, this key is also used as the level key for the lowest hierarchy level.
- Renaming columns in the Business Model and Mapping layer automatically creates aliases (synonyms) for Presentation layer columns that have the property **Use Logical Column Name** selected. This occurs because Presentation layer columns with this option selected are automatically renamed so that the logical column and presentation column names are in sync. Note that renaming

Presentation layer columns directly (when **Use Logical Column Name** is not selected) also causes an alias to be created.

- To prevent problems with aggregate navigation, ensure that each logical level of a dimension hierarchy contains the correct value in the field named **Number of elements at this level**. Fact sources are selected on a combination of the fields selected as well as the levels in the dimensions to which they map. By adjusting these values, you can alter the fact source selected by the Oracle BI Server. See "Creating Logical Levels in a Dimension" for more information about setting this value.

Modeling Outer Joins

The following guidelines provide tips on how to model outer joins:

- Due to the nature of outer joins, queries that use them are usually slower. Because of this, define outer joins only when necessary. Where possible, use ETL techniques to eliminate the need for outer joins in the reporting SQL.
- Outer joins are always defined in the Business Model and Mapping layer. Physical layer joins do not specify inner or outer.
- You can define outer joins by using logical table joins, or in logical table sources. Which type of outer join you use is determined by whether the physical join maps to a business model join, or to a logical table source join.
- If you must define an outer join, try to create two separate dimensions, one that uses the outer join and one that does not. Make sure to name the dimension with the outer join in a way that clearly identifies it, so that client users can use it as little as possible.
- Avoid using more than one outer join. Instead, to achieve the same effect as a logical outer join, Oracle recommends that the logical join be an inner join and that the analysis designer at design time selects the **Include Null Value** option in the corresponding analysis. For more about the **Include Null Value** option in the Analysis Editor, see "Understanding Null Suppression" in *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition*.

Design Tips for the Presentation Layer

You set up the user view of a business model in the Presentation layer. The names of folders and columns in the Presentation layer can appear in localized language translations. The Presentation layer is the appropriate layer in which to set user permissions. For complete information about working in the Presentation layer, see Chapter 12.

In this layer, you can do the following:

- You can show fewer columns than exist in the Business Model and Mapping layer. For example, you can exclude the key columns because they have no business meaning.
- You can organize columns using a different structure from the table structure in the Business Model and Mapping layer.
- You can display column names that are different from the column names in the Business Model and Mapping layer.
- You can set permissions to grant or deny users access to individual subject areas, tables, and columns.
- You can export logical keys to ODBC-based query and reporting tools.

- You can create multiple subject areas for a single business model.
- You can create a list of aliases (synonyms) for presentation objects that can be used in Logical SQL queries. This feature lets you change presentation column names without breaking existing reports.

The following is a list of tips to use when designing the Presentation layer:

- Because there is no automatic way to synchronize all changes between the Business Model and Mapping layer and the Presentation layer, it is best to wait until the Business Model and Mapping layer is relatively stable before adding customizations in the Presentation layer.
- There are many ways to create subject areas, such as dragging and dropping the entire business model, dragging and dropping incremental pieces of the model, or automatically creating subject areas based on logical stars or snowflakes. See "About Creating Subject Areas" for information about each of these methods. Dragging and dropping incrementally works well if certain parts of your business model are still changing.
- It is a best practice to rename objects in the Business Model and Mapping layer rather than the Presentation layer, for better maintainability. Giving user-friendly names to logical objects rather than presentation objects ensures that the names can be reused in multiple subject areas. Also, it ensures that the names persist even when you need to delete and re-create subject areas to incorporate changes to your business model.
- Be aware that members in a presentation hierarchy are not visible in the Presentation layer. Instead, you can see hierarchy members in Answers.
- You can use the Administration Tool to update Presentation layer metadata to give the appearance of nested folders in Answers. See "Nesting Folders in Answers and BI Composer" for more information.
- When setting up data access security for a large number of objects, consider setting object permissions by role rather than setting permissions for individual columns. See Chapter 14, "Applying Data Access Security to Repository Objects" for details.
- When setting permissions on presentation objects, you can change the default permission by setting the `DEFAULT_PRIVILEGES` configuration setting in the NQSConfig.INI file. See "Appendix A: NQSConfig.INI File Configuration Settings" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

Topics of Interest in Other Guides

Some topics that may be of interest to metadata repository builders are covered in other guides. Table 1–1 lists these topics, and indicates where to go for more information.

Table 1–1 Topics Covered in Other Guides

Topic	Where to Go for More Information
Starting and stopping Oracle Business Intelligence processes	*Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*
Using the Oracle BI Server XML API to work with your repository	*Oracle Fusion Middleware XML Schema Reference for Oracle Business Intelligence Enterprise Edition*

Table 1–1 (Cont.) Topics Covered in Other Guides

Topic	Where to Go for More Information
Using the Oracle BI Server web services	*Oracle Fusion Middleware Integrator's Guide for Oracle Business Intelligence Enterprise Edition*
Setting up and managing query caching	*Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*
Managing configuration settings that affect repository development in Fusion Middleware Control and NQSConfig.INI	*Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*
Managing users, groups, and application roles	*Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*
Moving from test to production environments	*Oracle Fusion Middleware Administrator's Guide*
Managing Oracle Business Intelligence Application Module metadata in BI Archive (BAR)	*Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*
Setting up DSNs for the Oracle BI Server	*Oracle Fusion Middleware Integrator's Guide for Oracle Business Intelligence Enterprise Edition*
Localizing Oracle Business Intelligence deployments	*Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*
Information about the SA System subject area	*Oracle Fusion Middleware Scheduling Jobs Guide for Oracle Business Intelligence Enterprise Edition*
Managing logging	*Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*
Managing usage tracking	*Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*
General information about managing Oracle WebLogic Server	*Oracle Fusion Middleware Administrator's Guide*

System Requirements and Certification

Refer to the system requirements and certification documentation for information about hardware and software requirements, platforms, databases, and other information. Both of these documents are available on Oracle Technology Network (OTN).

The system requirements document covers information such as hardware and software requirements, minimum disk space and memory requirements, and required system libraries, packages, or patches:

http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-requirements-100147.html

The certification document covers supported installation types, platforms, operating systems, databases, JDKs, and third-party products:

http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html

2

Before You Begin

This chapter provides an overview of the Oracle BI Administration Tool, and explains other concepts that you must know before beginning to build a metadata repository.

This chapter contains the following topics:

About the Oracle BI Administration Tool

The Oracle BI Administration Tool is a Windows application that you can use to create and edit repositories.

This section describes the Administration Tool main window, how to set preferences, Administration Tool menus, and other related information.

This section contains the following topics:

Opening the Administration Tool

To open the Administration Tool, choose **Start > Programs > Oracle Business Intelligence > BI Administration**.

> **Note:** Do not open the Administration Tool by double-clicking a repository file. The resulting Administration Tool window is not initialized to your Oracle instance, and errors will result.

You can also launch the Administration Tool from the `admintool` utility. The location of the `admintool` utility is:

ORACLE_HOME/user_projects/domains/bi/bitools/bin

About the Administration Tool Main Window

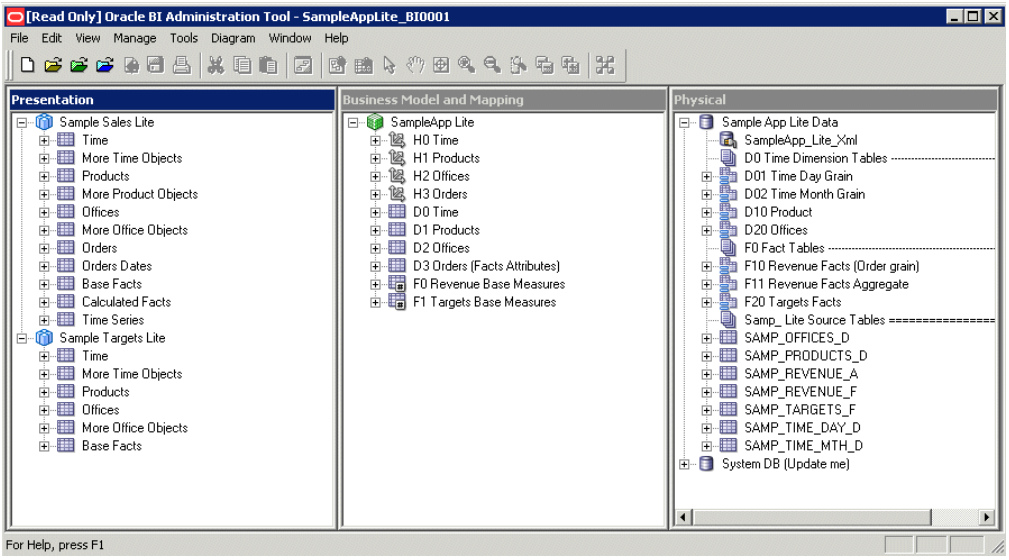
The main window of the Administration Tool shows a graphical representation of the three layers of a repository (the Physical layer, Business Model and Mapping layer, and Presentation layer). See "About Layers in the Oracle BI Repository" for more information.

The Administration Tool main window also contains the following:

- **Menus.** See "About Administration Tool Menus" for more information.
- **Toolbar.** Provides access to global functionality such as Open and Save, and also includes functions for the Physical Diagram and Business Model Diagram.
- **Status bar.** Provides contextual information about the current dialog or selected object, as well as other useful information.
- **Title bar.** In online mode, displays the DSN for the Oracle BI Server to which you are connected. In offline mode, displays one of the following:
  - RPD files: The name of the open repository (for example, SampleAppLite).
  - MDS XML files: The format and root folder location (for example, MDS XML C:\Root_Folder).

Figure 2–1 shows the Administration Tool main window.

Figure 2–1 Example Administration Tool Main Window



Setting Administration Tool Options

You can use the Options dialog to set preferences and options for the Administration Tool.

To set Administration Tool options:

1. In the Administration Tool, select **Tools**, then select **Options** to display the Options dialog.
2. On the General tab, select the options you want to choose.

   Table 2–1 describes the options on the General tab.

Table 2–1 Options on the General Tab

Option	Action When Selected
Tile when resizing	Automatically tiles the layer panes of the repository when you resize the Administration Tool. When this option is selected, the **Cascade** and **Tile** options are not available in the **Windows** menu of the Administration Tool.
Display qualified names in diagrams	Displays fully qualified names in the Physical Diagram and Business Model Diagram. For example, selecting this option displays **"B - Sample Fcst Data"..."B02 Market"** rather than **B02 Market** in the Physical Diagram. Selecting this option can help identify objects by including the name of the parent database or business model, but it can also make the diagram harder to read because the fully qualified names are longer. **Note:** If you choose not to select this option, you can still see fully qualified names by moving the cursor over an object in the diagram, or by selecting an object in the diagram and then viewing the text in the status bar.
Display original names for alias in diagrams	Displays the names of original physical tables rather than the names of alias tables in the Physical diagram. Select this option when you want to identify the original table rather than the alias table name.

Table 2–1 (Cont.) Options on the General Tab

Option	Action When Selected
Show Calculation Wizard introduction page	Displays the Calculation Wizard introduction page. The introduction page also contains an option to suppress its display in the future. Use the Calculation Wizard to create new calculation columns that compare two existing columns and to create metrics in bulk (aggregated), including existing error trapping for NULL and divide by zero logic. See "Using the Calculation Wizard" for more information.
Check out objects automatically	Automatically checks out an object when you double-click it. If you do not select this option, you are prompted to check out objects before you can edit them. This option only applies when the Administration Tool is open in online mode. See "Editing Repositories in Online Mode" for more information.
Show row count in physical view	Displays row counts for physical tables and columns in the Physical layer. Row counts are not initially displayed until they are updated. To update the counts, select **Tools > Update All Row Counts**. You can also right-click a table or column in the Physical layer and select the option **Update Row Count**. **Note:** Row counts are not shown for items that are stored procedure calls (from the **Table Type** list in the General tab of the Physical Table dialog). Row counts are not available for XML, XML Server, or multidimensional data sources. When you are working in online mode, you cannot update row counts on any new objects until you check them in.
Show toolbar	When selected, displays the Administration Tool toolbar.
Show statusbar	When selected, displays the Administration Tool status bar.
Prompt when moving logical columns	Lets you ignore, specify an existing, or create a new logical table source for a moved column.
Remove unused physical tables after Merge	Executes a utility to clean the repository of unused physical objects. It might make the resulting repository smaller.
Allow import from repository	When selected, the **Import from Repository** option on the **File** menu becomes available. **Note:** By default, the **Import from Repository** option on the **File** menu is disabled and this option will not be supported in the future. It is recommended that you create projects in the repository that contain the objects that you want to import, and then use repository merge to bring the projects into your current repository. See "Merging Repositories" for more information.
Allow logical foreign key join creation	When selected, provides the capability to create logical foreign key joins with the Joins Manager. This option is provided for compatibility with previous releases and is generally not recommended.
Skip Gen 1 levels in Essbase drag and drop actions	When selected, excludes Gen 1 levels when you drag and drop Essbase cubes or dimensions from the Physical layer to the Business Model and Mapping layer. Often, Gen 1 levels are not needed for analysis, so they can be excluded from the business model. See "Working with Essbase Data Sources" for more information.

Table 2–1 (Cont.) Options on the General Tab

Option	Action When Selected
Hide unusable logical table sources in Replace wizard	By default, the Replace Wizard shows all logical table sources, even ones that are not valid for replacement. When this option is selected, unusable logical table sources are hidden in the Replace Wizard screens. Click **Info** for details on why a logical table source that maps to that column does not appear in the list. Selecting this option might result in the Wizard page loading more quickly, especially for large repositories.
Allow first Connection Pool for Init Blocks	By default, when you select a connection pool for an initialization block, the first connection pool under the database object in the Physical layer does not display as available for selection. This behavior ensures that you cannot use the same connection pool for initialization blocks that you use for queries. If the same connection pool is used for initialization blocks and for queries, then queries might be blocked whenever initialization blocks run. Alternatively, initialization blocks used for authentication might be blocked by long-running queries, causing delayed or hanging logins. Select this option to change the default behavior and allow the first connection pool to be selected for initialization blocks. Note that selecting this option is not a best practice and might cause performance issues. See "About Connection Pools for Initialization Blocks" for more information.
Show Upgrade ID in Query Repository	Upgrade IDs are not displayed by default in the Query Repository dialog. When this option is selected, Upgrade IDs are displayed as a column in the Query Repository results. In addition, you can set a filter on Upgrade ID to search for a particular value. This option is useful for MDS XML format repositories in which the Upgrade ID is included in the file name.
Extender For BIAPPS	Depending on your configuration, this option might not be enabled.
Show Tenant Info in Online Login	If you are working in a multitenant environment, then select this option to show the **Tenant info** field in the Open Online dialog.
Display Translation Key in the presentation tree	By default, the names of the presentation objects display in the Presentation layer portion of the Administration Tool's main window. Select this option to instead display the translation key values for all presentation objects.
Edit presentation names	By default, the presentation object names are read-only. Select this option to allow the names of presentation objects to be modified.
Drag and drop: Show only hierarchal columns	For Essbase data sources, selecting this option hides presentation columns and shows only hierarchal columns in Answers.

3. On the Repository tab, you can set the following options:

   - **Show tables and dimensions only under display folders.** You can create display folders to organize objects in the Physical and Business Model and Mapping layers. They have no metadata meaning. After you create a display

folder, the selected objects appear in the folder as a shortcut and in the database or business model tree as an object. You can hide the objects so that only the shortcuts appear in the display folder.

See "Setting Up Display Folders in the Physical Layer" and "Setting Up Display Folders in the Business Model and Mapping Layer" for more information about creating display folders.

- **Hide level based measure.** By default, each level of a dimension hierarchy in the Business Model and Mapping layer shows both dimension columns that are assigned to that level, and level-based measures that have been fixed at that level. Level-based measures are objects that are not part of the dimension table, but that have been explicitly defined as being at a particular level.

  Hiding level-based measures in dimension hierarchies can reduce clutter. Note that the measures are still visible in the logical fact tables.

  See Example 10–1, "Level-Based Measure Calculations" for more information about level-based measures.

- **System logging level.** This option determines the default query logging level for the internal BISystem user. The BISystem user owns the Oracle BI Server system processes and is not exposed in any user interface.

  A query logging level of 0 (the default) means no logging. Set this logging level to 2 to enable query logging for internal system processes like event polling and initialization blocks.

  See "Managing the Query Log" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about the query log and query logging levels.

- **LDAP.** If you are using any alternative LDAP servers, the Oracle BI Server maintains an authentication cache in memory for user identifiers and properties, which improves performance when using LDAP to authenticate large numbers of users. Disabling the authentication cache can slow performance when hundreds of sessions are being authenticated. Note that the authentication cache is not used for Oracle WebLogic Server's embedded directory server.

  Properties for the authentication cache include:

  – **Cache refresh interval.** The interval at which the authentication cache entry for a logged on user is refreshed.

  – **Number of Cache Entries.** The maximum number of entries in the authentication cache, preallocated when the Oracle BI Server starts. If the number of users exceeds this limit, cache entries are replaced using the LRU algorithm. If this value is 0, then the authentication cache is disabled.

  You need to specify some additional LDAP properties when you are using a secure connection to your LDAP server. In other words, provide the following information when you have selected **SSL** on the Advanced tab of the LDAP Server dialog:

  – **Key file name.** The name of the key file that holds the client certificate and Certificate Authority (CA) certificate.

  – **Password and Confirm password.** The password for the key file.

  Note that the authentication cache properties and key file properties are shared for all defined LDAP server objects.

4. On the Sort Objects tab, specify which repository objects appear in the Administration Tool in alphabetical order. For example, if you want the database objects that appear in the Physical layer to appear in alphabetical order, select the **Database** option.
5. On the Source Control tab, you can create or edit a configuration file to integrate with a source control management system, as well as view and change the status of an MDS XML repository (either **Standalone** or **Use Source Control**). See "Creating an SCM Configuration File" for more information.
6. On the Cache Manager tab, select the columns you want to display in the Cache Manager. To change the order of columns in the Cache Manager, select an item, then use the **Up** and **Down** buttons to change its position.
7. On the Multiuser tab, specify the path to the multiuser development directory and the name of the local developer for this Administration Tool. See "Setting Up a Pointer to the Multiuser Development Directory" for more information.
8. On the More tab, you can set the scrolling speed for Administration Tool dialogs. To set the scrolling speed, position the cursor on the slider.
9. Click **OK** when you are finished setting preferences.

About Administration Tool Menus

The Administration Tool includes menus for File, Edit, View, Manage, Tools, Diagram, Window, and Help. These menus are described in the following sections.

File Menu

The File menu provides options to work with repositories, like **Open** and **Save**, as well as several server-related options like **Check Out All** that are only active when a repository is open in online mode. The File menu also provides a list of recently opened repositories.

Table 2–2 lists the options in the File menu.

Table 2–2 File Menu Options

Menu Option	Description
New Repository	Opens the Create New Repository Wizard and closes the currently open repository, if any. If a repository is currently open with unsaved changes, you are prompted to save them before proceeding. See "Creating a New Oracle BI Repository" for more information.
Open	Provides the following options: ■ **Offline:** Select this option to open an RPD file in offline mode. ■ **MDS XML:** Select this option to open a repository in MDS XML format in offline mode. ■ **Online:** Select this option to open an RPD file in online mode. See "Using Online and Offline Repository Modes" and "About MDS XML" for more information.

Table 2–2 (Cont.) File Menu Options

Menu Option	Description
Multiuser	Provides options to check out projects in a multiuser development environment and view multiuser development history. See Chapter 3, "Setting Up and Using the Multiuser Development Environment" for more information.
Source Control	Provides options for MDS XML format repositories that are integrated with your source control management system. See Chapter 4, "Using a Source Control Management System for Repository Development" for more information.
Close	Closes the currently open repository. If you have unsaved changes, you are prompted to save them.
Save	Saves your latest changes.
Save As	Provides the following options: ■ **Repository:** Opens the Save As dialog so that you can save the repository in RPD format. ■ **MDS XML Documents:** Opens the Browse For Folder dialog so that you can specify the root folder location for MDS XML output. The new repository remains open in the Administration Tool.
Copy As	Provides the following options: ■ **Repository:** Opens the Save Copy As dialog so that you can copy the repository to a different file in RPD format. ■ **MDS XML Documents:** Opens the Browse For Folder dialog so that you can specify the root folder location for MDS XML output. The current repository, not the new repository, remains open in the Administration Tool.
Change Password	Lets you change the repository password for the currently open repository. See "Changing the Oracle BI Repository Password" for more information.
Load Java Datasources	Used to connect to the Java Datasource server, which accesses the Java data sources, loads the metadata, and makes it available for import into the repository. See "About Setting Up JDBC and JNDI Data Sources" for more information.
Print Preview	Used with the Physical and Business Model Diagrams. Provides a preview of how the diagram will look when printed.
Print	Prints the Physical or Business Model Diagram.
Import Metadata	Opens the Import Metadata Wizard. See the following sections for more information: ■ "Importing Metadata from Relational Data Sources" ■ "Importing Metadata from Multidimensional Data Sources" ■ "Working with ADF Data Sources" ■ "About Importing Metadata from XML Data Sources"

Table 2–2 (Cont.) File Menu Options

Menu Option	Description
Compare	Prompts you to select the repository with which you want to compare the currently open repository and opens the Compare repositories dialog. See "Comparing Repositories" for more information.
Turn off Compare Mode	Turns off any highlighted changed objects. This option is only available if you have turned on compare mode by choosing **Mark** in the Compare repositories dialog.
Merge	Opens the Merge Repository Wizard. See "Merging Repositories" for more information.
Check Global Consistency	Checks the repository for consistency and opens the Consistency Check Manager. See "Checking the Consistency of a Repository or a Business Model" for more information.
Check Out All	Checks out all repository objects. This option is only available in online mode.
Check In Changes	Checks in all repository objects. This option is only available in online mode.
Undo All Changes	Rolls back all changes made since the last check-in. This option is only available in online mode.
Exit	Closes the currently open repository and then closes the Administration Tool. If you have unsaved changes, you are prompted to save them.

Edit Menu

The Edit menu provides access to the following basic editing functions for repository objects: **Cut**, **Copy**, **Paste**, **Duplicate**, and **Delete**. You can also choose **Properties** to view and edit properties for a selected object.

View Menu

The View menu options let you hide or display the panes that show the three layers of the repository (Presentation, Business Model and Mapping, and Physical). You can also display the Business Model Diagram and Physical Diagram.

Choose **Refresh** to refresh the repository view. This feature can be useful in online mode to reveal changes made by other clients. It can also be used in either online or offline mode when the repository view has become out of sync and does not display a recent change or addition. Refreshing the repository view collapses any expanded objects in the tree panes and helps reduce clutter.

Manage Menu

The Manage menu enables you to access the management functions described in Table 2–3.

Table 2–3 Manage Menu Options

Menu Option	Description
Sessions	Opens the Session Manager. In the Session Manager, you can monitor activity on the system, including the current values of repository and session variables. This option is available when a repository is open in online mode. See "Managing Server Sessions" in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition* for more information.
Cache	Opens the Cache Manager. The Cache Manager enables you to monitor and manage the cache. This option is available when a repository is open in online mode and caching is enabled. See "Using the Cache Manager" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about enabling the cache and using the Cache Manager.
Clusters	Opens the Cluster Manager. The Cluster Manager monitors and manages the operations and activities of the cluster. This option is available when the Oracle BI Cluster Server is installed. See "Using the Cluster Manager" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.
Identity	Opens the Identity Manager. The Identity Manager provides access to data access security functions and other identity-related options. See Chapter 14, "Applying Data Access Security to Repository Objects" for more information.
Joins	Opens the Joins Manager. The Joins Manager enables you to work with both physical and logical joins in a list format. The Joins Manager provides an alternative to working with joins in the Physical and Business Model Diagrams and shows all join types in one place. See "Defining Physical Joins with the Joins Manager" and "Defining Logical Joins with the Joins Manager" for more information.
Variables	Opens the Variable Manager. The Variable Manager enables you to create, edit, or delete variables and initialization blocks. See Chapter 19, "Using Variables in the Oracle BI Repository" for more information.
Projects	Opens the Project Manager. The Project Manager enables you to create, edit, or remove projects or project elements. Project elements include subject areas (formerly called presentation catalogs), logical fact tables, groups, users, variables, and initialization blocks. You use projects during multiuser development. See "Setting Up Projects" for more information.
Marketing	Applies to the Oracle Marketing Segmentation product. For information about using the Marketing options in Oracle Business Intelligence, see *Oracle Marketing Segmentation Guide*.

Tools Menu

The Tools menu options enable you to access the functions described in Table 2–4.

Table 2–4 Tools Menu Options

Menu Option	Description
Update All Row Counts	Updates row counts in the Physical layer. See "Displaying and Updating Row Counts for Physical Tables and Columns" for more information.

Table 2–4 (Cont.) Tools Menu Options

Menu Option	Description
Show Consistency Checker	Opens the Consistency Check Manager. See "Checking the Consistency of a Repository or a Business Model" for more information.
Query Repository	Opens the Query Repository dialog. See "Querying and Managing Repository Metadata" for more information.
Utilities	Opens the Utilities dialog, which lets you select from a list of Administration Tool utilities. See "Using Administration Tool Utilities" for more information.
Options	Opens the Options dialog, which lets you customize Administration Tool display preferences and other options. See "Setting Administration Tool Options" for more information.

Diagram Menu

The Diagram menu options are available when working with the Physical Diagram or Business Model Diagram. The options enable you to select elements, create new joins, create new tables, and perform other diagram operations. Every toolbar option for the diagrams has a Diagram menu equivalent.

Window Menu

The Window menu options enable you to cascade or tile open layer windows and toggle among them.

Help Menu

The Help menu provides the following options:

- **Help Topics.** Access the Help system for the Administration Tool.
- **Oracle BI on the Web.** Access the Oracle Business Intelligence home page on the Oracle Technology Network (OTN).
- **About Oracle BI Administration Tool.** Obtain version information about the Administration Tool. This information can be used for support and troubleshooting purposes. Note the following information included in this dialog:
  - **Version** – The installed version of Oracle BI EE you are using. For example, 11.1.1.9.0.
  - **Package** – The Oracle internal product build number corresponding to the Oracle BI EE version. This number is composed of the date and time when the product was compiled and built by Oracle. For example, 140728.1003.000.
  - **Repository version** – If you do not have a repository open, then the number displayed is the repository version currently supported by the Administration Tool. For example, 347. Any new repositories that you create are saved with the repository version shown.

    If you open a repository created with an earlier version of the Administration Tool, then the first number displayed is the version number that was used when the repository was created and saved. The second number displays the repository version currently supported by the Administration Tool. For example, 326/347. When you save the open repository, you can choose to save

the repository's version to the currently supported version. Note that the Administration Tool cannot open a repository with a version number higher than what it supports in the current release.

– **MUD version** – The multiuser development (MUD) environment version supported by the Oracle BI EE version.

Using the Physical and Business Model Diagrams

You can use the Physical and Business Model Diagrams in the Administration Tool to see a graphical view of physical and logical tables and joins. You can choose to view tables in expanded mode, with columns visible, or in collapsed mode, where only the name of the table is displayed. This section describes the layout and navigation capabilities for both diagrams.

After launching the Physical or Business Model Diagram, you can use toolbar options to zoom, pan, and control the layout of the tables. Table 2–5 describes the available toolbar options.

Table 2–5 Toolbar Options for the Physical and Business Model Diagrams

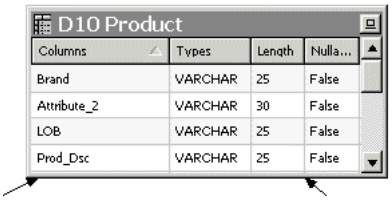
Option Name	Description
Auto Layout 	Select this option to revert to an automatically assigned symmetric table layout. Any customizations you have made to the layout (by manually moving individual tables) will be lost.
Expand All 	Select this option to show all tables in expanded view, with columns showing. Tables in expanded view appear like the following:  Note the following additional features for expanded tables: ■ Use the scrollbar to scroll down the full list of columns. ■ Click a column heading to sort based on that column. ■ Double-click a table in expanded view to launch the Properties dialog for that object. ■ Click the Collapse icon in the upper right corner to collapse an individual table object. ■ To resize expanded tables, select a table, mouse over a handle, and then click and drag the handle.
Collapse All 	Select this option to show all tables in collapsed view, with only the table name showing. Tables in collapsed view appear like the following:  You can double-click an individual table in collapsed view to expand only that object.

Table 2–5 (Cont.) Toolbar Options for the Physical and Business Model Diagrams

Option Name	Description
Marquee Zoom 	Select this option to use the Marquee Zoom tool, which lets you select a particular region to which you want to zoom. To use Marquee Zoom, left-click, hold, and drag to define a rectangular region where you want to zoom.
Zoom Out 	Select this option to cause the diagram view to zoom out one level.
Zoom In 	Select this option to cause the diagram view to zoom in one level.
Fit 	Select this option to cause the layout to dynamically adjust to the current diagram window size so that all objects fit in the window.
Pan 	Select this option to use the Pan tool, which lets you pan around the current layout. Left-click, hold, and drag to move the view. This option is especially useful when the diagram layout exceeds the available space.
Select 	Select this option to enable the ability to select objects in the diagram. You can double-click a join or expanded table object to access the Properties dialog, or you can select a particular table and drag it to a new location. Note that location information is not saved after you close the diagram or choose Auto Layout. You can select multiple objects using the SHIFT or CTRL keys. Press SHIFT and select multiple objects, or click and drag to define an area where you want all objects selected. Press CTRL to individually add or remove particular objects to the selection set.
New Table 	Select this option to create a new physical or logical table while in the diagram view. Left-click the background to launch the Properties dialog for the new object, and then provide details as necessary. For physical tables, you first need to select the parent object under which the new table will be created (such as a schema, catalog, or database object). See also the following sections: ■ "Creating and Managing Physical Tables and Physical Cube Tables" ■ "Creating Logical Tables"
New Join 	Select this option to create a new join while in the diagram view. First, left-click the first table in the join (the table representing many in the one-to-many join). Then, move the cursor to the table to which you want to join (the table representing one in the one-to-many join), and then left-click the second table to select it. Provide details in the Properties dialog for the new object as necessary. Joins in the Physical and Business Model Diagrams are represented by a line with an arrow at the "one" end of the join. Note that this display is different from the line with crow's feet at the "many" end of the join that was used in previous releases. See also the following sections: ■ "Defining Physical Joins with the Physical Diagram" ■ "Defining Logical Joins with the Business Model Diagram"

Note the following additional features of the Physical and Business Model Diagrams:

- All toolbar options for the diagram, such as Select, New Table, and New Join, are also available from the Diagram menu.
- Moving the mouse over a table causes the fully-qualified name for that table to appear in the status bar.
- You can have both the Physical Diagram and Business Model Diagram windows open at the same time.
- Any customizations you have made to the layout (by manually moving individual tables) are lost after you close the diagram or choose Auto Layout.
- You can cause fully-qualified table names to appear in diagrams by setting a preference in the Options dialog. See "Setting Administration Tool Options" for more information.
- You can use the **Print** and **Print Preview** options on the File menu to manage printing options for the diagrams. You can also use the **Print** option on the toolbar.

See also the following sections for more information about using the Physical and Business Model Diagrams:

- "Physical Diagram and Business Model Diagram Keyboard Shortcuts"
- "Working with the Physical Diagram"
- "About Working with the Business Model Diagram"

Editing, Deleting, and Reordering Objects in the Repository

This section provides information about editing, deleting, and reordering objects.

- To edit objects, double-click an object, or right-click an object and select **Properties**. Then, complete the fields in the dialog that is displayed. In some dialogs, you can click **Edit** to open the appropriate dialog.
- To delete objects, select one or more objects and click **Delete**, or press the delete key. You can also right-click an object and select **Delete**.
- To reorder objects, drag and drop an object to a new location. Note the following:
  - Reordering is only possible for certain objects and in certain dialogs.
  - In some dialogs, you can use an up or down arrow to move objects to a new location.
  - In the Administration Tool main window, you can drag and drop an object onto its parent to duplicate the object. For top-level objects like business models and subject areas, drag and drop the object onto white space to duplicate it.

About Naming Requirements for Repository Objects

All repository object names must follow these requirements:

- Names cannot be longer than 128 characters
- Names cannot contain leading or trailing spaces
- Names cannot contain characters such as single quotes, hash marks, question marks, or asterisks

Note that repository object names can include multibyte characters.

Using the Browse Dialog to Browse for Objects

The Browse dialog appears in many situations in the Administration Tool. You use it to find and select an object.

The Browse dialog is accessible from several dialogs that let you make a selection from among existing objects.

The left pane of the Browse dialog lets you browse the tree view for a particular object. It contains the following parts:

- A tree listing all of the objects in the Presentation layer, Business Model and Mapping layer, or the Physical layer of a repository.
- Tabs at the bottom of the left pane let you select a layer. Some tabs might not appear if objects from those layers are not appropriate for the task you are performing.

The right pane of the Browse dialog lets you search for the object you want. It contains the following parts:

- **Query** enables you to query objects in the repository by name and type. The **Name** field accepts an asterisk (*) as the wildcard character, so you can query for partial matches.
- The **Show Qualified Names** option lets you identify to which parents an object belongs.
- **View** lets you view properties of a selected object in read-only mode.

Note that in general, the left pane and the right pane of the Browse dialog are not connected. Rather, the panes provide alternate methods to locate the object you want.

The exception to this is the Synchronize Contents feature, which lets you synchronize an object from the query results list with the tree view. This feature is a helpful contextual tool that locates a particular object in the tree view.

Table 2–6 lists and describes the tasks you can perform in the Browse dialog.

Table 2–6 Tasks You Can Perform in the Browse Dialog

Task	Description
Querying for an object	Follow these steps to query for an object: 1. Select the object type from the **Type** list. 2. Type the name of the object, or a part of the name and the wildcard character (*), in the **Name** field. For example: - To search for logical tables that have names beginning with the letter Q, select **Logical Tables** from the **Type** list, and then type **Q*** in the Name field. - To search for logical tables that have names ending with the letters dim, type ***dim** in the name field. 3. Click **Query**. Relevant objects appear in the query results list.
Selecting an object	Use the tree view in the left pane or the filtered view in the right pane to locate the object you want, then double-click the object. The Browse dialog closes, and the object is displayed in the previous dialog.

Table 2–6 (Cont.) Tasks You Can Perform in the Browse Dialog

Task	Description
Synchronizing an object in the query results list with the tree view	Select an object in the Query list and then click the **Synchronize Contents** button. The object you selected is highlighted in the tree view in the left pane.
Finding multiple occurrences of an object in the tree view	Select an object in the tree view, such as a logical column, then click the down arrow button. The next occurrence of that object is highlighted in the tree view.

Changing Icons for Repository Objects

In the Administration Tool, you can change the icon that represents a particular object in the repository. Changing the icon for a particular object does not have any functional effect, and is not visible in Answers or other clients. This feature is intended as a useful way to visually distinguish objects for the convenience of repository developers.

For example:

- You can use a special icon for objects that are in the Business Model and Mapping layer, but not the Presentation layer, for easier maintenance of the repository.
- You can mark objects that are logical calculations with a separate icon.
- You can choose an icon to visually distinguish tables in the Presentation layer that appear as nested folders in Answers.
- You can use an icon to denote objects in a logical table that pertain to a specific functional area, or that are sourced from a particular logical table source.

You can only change the icon for individual objects. You cannot globally change the icon for all objects of a particular type.

To change the icon for a particular repository object:

1. In the Administration Tool, right-click an object in the Physical, Business Model and Mapping, or Presentation layer (for example, a particular logical table).
2. Select **Set Icon**.
3. In the Select Icon dialog, select the icon you want to use for that object and click **OK**.

Sorting Objects in the Administration Tool

Many dialogs in the Administration Tool show lists of objects, such as a list of physical columns in the Physical Table dialog, a list of logical levels for Preferred Drill Path in the Logical Level dialog, and a list of presentation hierarchies in the Presentation Table dialog.

You can click the header to sort the objects in ascending or descending order. An up arrow or down arrow icon is displayed next to the header name, indicating how the list has been sorted.

Each list also has a default order that is persisted from session to session. The default order appears when you view a list in a dialog for the first time each session. The default order is displayed when there is no ascending or descending arrow icon in the header. Click the header three times to toggle between ascending, descending, and default order. In some cases, the default order is the ascending or descending order.

Some dialogs provide the capability to move items up or down in a list. In these dialogs, if you click **Up** or **Down** while the list is sorted in ascending or descending order, the selected item moves, and the resulting order becomes the new default order. Note that clicking the header eliminates any manually determined order.

About Features and Options for Oracle Marketing Segmentation

Some features and options in the Administration Tool are for use by organizations that have the Oracle Marketing Segmentation product. For information about these features and options, see *Oracle Marketing Segmentation Guide*.

Note that additional information about Oracle Marketing Segmentation features is provided in the Presentation Services Help.

About the Oracle BI Server Command-Line Utilities

You can use a variety of command-line utilities with the Oracle BI Server to make programmatic changes to your repository file, run sample queries, delete unwanted repository objects, and perform other tasks.

Table 2–7 describes the Oracle BI Server command-line utilities.

Table 2–7 Oracle BI Server Command-Line Utilities

Utility Name	Description	Where to Go for More Information
`biserverextender`	Used to import flex object changes from ADF data sources and map them to the Business Model and Mapping layer and Presentation layer.	"Automatically Mapping Flex Object Changes Using the biserverextender Utility"
`biservergentypexml`	Used to compare data types of logical columns between a particular repository and a generated list of logical column types to ensure that the types match as expected.	"Generating a List of Logical Column Types" "Comparing Logical Column Types"
XML utilities (`biserverxmlgen`, `biserverxmlexec`, `biserverxmlcli`)	Primarily used to leverage the Oracle BI Server XML API for metadata migration, programmatic metadata generation and manipulation, metadata patching, and other functions. The XML utilities include: ▪ `biserverxmlgen`: generates XML from an existing RPD file. Also includes an option to generate repositories in MDS XML format. ▪ `biserverxmlexec`: executes the XML in offline mode to create or modify a repository file. Also includes an option to execute XML in MDS XML format. ▪ `biserverxmlcli`: executes the XML against the Oracle BI Server.	*Oracle Fusion Middleware XML Schema Reference for Oracle Business Intelligence Enterprise Edition*
`comparerpd`	Used to compare two repositories and generate a CSV diff file, an XML patch file, or an MDS XML diff.	"Comparing Repositories Using comparerpd"
`deleteapproles`	Used to upload a JSON file containing a list of application roles to delete from a specific server instance.	"Delete Application Role Command"

Table 2–7 (Cont.) Oracle BI Server Command-Line Utilities

Utility Name	Description	Where to Go for More Information
`deleteusers`	Used to upload a JSON file containing a list of users to delete from a specific server instance.	"Delete Users Command"
`downloadpd`	Used to download the repository to work offline on diagnostics and development tasks.	"Download Repository Command"
`equalizerpds`	Used to equalize objects in two repositories that have the same name, but different Upgrade IDs. Running this utility before merging repositories prevents unintended renaming during the merge.	"Equalizing Objects"
`externalizestrings`	Used to localize the names of Presentation layer subject areas, tables, hierarchies, columns and their descriptions.	"Using the Externalize Strings Utility"
`extractprojects`	Used to extract projects from a given repository.	"Using the extractprojects Utility to Extract Projects"
`listConnectionPool`	Used to create a list of connection pools in JSON format for a specific server instance.	"List Connection Pool Command"
`listrpdvariable`	Used to create a list of repository variables in JSON format for a specific service instance.	"List Repository Variables Command"
`mhlconverter`	Used to convert MUD history files from .mhl format to .xml format so that they can be checked in under source control.	"Checking In New Versions of the MUD Master and MUD Log File to Source Control"
`nqaggradvisor`	Used to invoke the Oracle BI Summary Advisor to generate an aggregate specification script that is run to create aggregates. This utility is only available for Oracle Business Intelligence running on an Oracle Exalytics machine.	"Using the nqaggradvisor Utility to Run the Oracle BI Summary Advisor"
`nqcmd`	Used to run test queries against the repository. Connects using an Oracle BI Server ODBC DSN.	"Using nqcmd to Test and Refine the Repository"
`nqlogviewer`	Used to view the query log.	*Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*
`obieerpdpwdchg`	Used to change the Oracle BI repository password.	"Changing the Oracle BI Repository Password Using the obieerpdpwdchg Utility"
`patchrpd`	Used to apply an XML patch file. This utility is especially useful for patching repository files on Linux or UNIX systems.	"Using patchrpd to Apply a Patch"
`prunerpd`	Used to delete unwanted repository objects from your repository file, such as databases, tables, columns, initialization blocks, and variables.	"Deleting Unwanted Objects from the Repository"
`renameapproles`	Used to upload a JSON file containing information about the application roles to rename for a specific server instance.	"Rename Application Role Command"

Table 2–7 (Cont.) Oracle BI Server Command-Line Utilities

Utility Name	Description	Where to Go for More Information
`renameusers`	Used to upload a JSON file containing a list of information about users to rename for a specific server instance.	"Rename Users Command"
`sametaexport`	Used to generate the information necessary for the Oracle Database SQL Access Advisor or IBM DB2 Cube Views tool to preaggregate relational data and improve query performance.	"Exchanging Metadata with Databases to Enhance Query Performance"
`updateConnectionPool`	Used to upload a modified JSON file containing updated connection pool values to a specific server instance.	"Update Connection Pool Command"
`updaterpdvariable`	Used to upload a JSON file or a modified JSON file containing variable information to a specific server instance.	"Update Repository Variables Command"
`uploadrpd`	Used to upload the repository to the Oracle BI Server and include changes in the Oracle Business Intelligence archive (BAR) file.	"Upload Repository Command"
`validaterpd`	Used to check the consistency of a repository.	"Using the validaterpd Utility to Check Repository Consistency"

About Options in NQSConfig.INI

Many configuration settings that affect the Administration Tool and repository development are managed in the NQSConfig.INI configuration file. Repository developers must be familiar with the NQSConfig.INI configuration settings to effectively work with the Administration Tool and with their repositories.

Some of the most common configuration settings that affect repository development include:

- **LOCALE:** This option is set in NQSConfig.INI. It specifies the locale in which data is returned from the server and determines the localized names of days and months.
- **DATE_TIME_DISPLAY_FORMAT, DATE_DISPLAY_FORMAT, TIME_DISPLAY_FORMAT:** These options are set in NQSConfig.INI. They control the display of date/time formats.
- **DEFAULT_PRIVILEGES:** This option is set in NQSConfig.INI. It determines the default privilege (NONE or READ) granted to users and application roles for repository objects without explicit permissions set.

See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for full information about NQSConfig.INI configuration settings.

About the SampleApp.rpd Demonstration Repository

Oracle Business Intelligence provides a sample repository called SampleApp.rpd that provides best practices for modeling many different types of objects described in this guide.

A basic version of SampleApp.rpd, called SampleAppLite.rpd, is automatically installed as the default repository when you install Oracle BI Enterprise Edition.

The full version of SampleApp.rpd contains many additional examples and features. This version can be found on the Oracle Technology Network at:

http://oracle.com/technetwork/middleware/bi-foundation/obiee-samples-167534.html

The default password for SampleAppLite.rpd is Admin123. For security reasons, you must immediately change this default password the first time you open SampleAppLite.rpd in the Administration Tool. See "Changing the Oracle BI Repository Password" for more information.

Download Repository Command

This topic contains the following sections:

What You Need to Know Before Using the Command

Note that the information in this section applies to the download and upload repository commands, and to the list and update connection pool, rename and delete users and application roles, list and update repository commands.

Note the following information about the commands.

System Privileges

For either the Oracle BI EE installation or client installation, you must have Oracle BI EE BI Service Administrator privileges to run the command line utility and issue any of the commands.

Passwords in Commands

The commands provide options for including a user's password and a repository passwords. If you do not supply passwords, then you will be prompted for passwords when you run the command.

For security purposes, Oracle recommends that you include passwords in the command only if you are using automated scripting to run the command.

Trust Store Key File for SSL

WebLogic Server provides Secure Sockets Layer (SSL) support for encrypting data transmitted between WebLogic Server clients and servers, Java clients, Web browsers, and other servers. When using SSL, you must use the WebLogic Server's trusted keys file if the server is using a self-signed certificate. This is the case when a domain is first created, as the server's identity certificate is generated when the domain is created.

If you replace the WebLogic Server's default self-signed identity certificate with a certificate signed by a recognized signing authority, then the standard Java trusted certificate list validates it and the extra settings are not needed.

The location of the WebLogic Server's trusted key file is:

<MW_HOME>/wlserver/server/lib/DemoTrust.jks

The default password for the DemoTrust.jks file is:

DemoTrustKeyStorePassPhrase

The location of the trusted key file and its password are passed to the system properties javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword. For example,

```
java \
-Djavax.net.ssl.trustStore=$ORACLE_HOME/wlserver/server/lib/DemoTrust.jks \
-Djavax.net.ssl.trustStorePassword=DemoTrustKeyStorePassPhrase \
-jar bi-commandline-tools.jar <args…>
```

Upon installation, the data-model-cmd.sh and data-model-cmd.cmd scripts are delivered with the trusted key file locations included. For Oracle BI EE installations, you do not need to update the trusted key file locations.

For Oracle BI EE client installation, you must put the trusted keys file in the correct location. To do this, Oracle suggests that you copy and paste the files from the WebLogic Server to the proper location.

Hostname, Port Number, and Use of SSL

For Oracle BI EE installations, the command line utility by default queries the Oracle BI EE endpoint manager which provides the host name, port number, and whether SSL is available. Therefore for Oracle BI EE installations, the user does not need to include these options in the command.

For Oracle BI EE client installation, you must include the `S` (Oracle BI EE host name), `N` (Oracle BI EE port number), and `SSL` (use SSL to connect to the WebLogic Server to run the command) options in the commands.

How to Use the Download Repository Command

Use the download repository command `downloadrpd` to download the repository used by the service instance.

This command downloads the repository from the Oracle Business Intelligence archive (BAR) file for the service instance. Oracle recommends working with the downloaded repository for offline diagnostic and development purposes such as testing, only. Note that in all other repository development and maintenance situations, you should use BAR to utilize BAR's repository upgrade and patching capabilities and benefits.

You execute the utility through a launcher script, data-model-cmd.sh on UNIX and data-model-cmd.cmd on Windows. You can find the launcher script at the following location:

Oracle_Home/user_projects/domains/bi/bitools/bin

See "What You Need to Know Before Using the Command" for more information.

Syntax

The `downloadrpd` command takes the following parameters:

```
downloadrpd -O RPDname [-W RPDpwd] -SI service_instance -U cred_username [-P cred_
password] [-S hostname] [-N port_number] [-SSL] [-H]
```

Where

`O` specifies the name of the repository that you want to download.

`W` specifies the password for the repository. If you do not supply the password, then you will be prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you are using automated scripting to run the command.

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for Oracle BI EE authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you do not supply the password, then you will be prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you are using automated scripting to run the command.

`S` specifies the Oracle BI EE host name. Only include this option when you are running the command from a client installation.

`N` specifies the Oracle BI EE port number. Only include this option when you are running the command from a client installation.

`SSL` specifies to use SSL to connect to the WebLogic Server to run the command. Only include this option when you are running the command from a client installation.

`H` displays the usage information and exits the command.

Example

```
data-model-cmd.sh downloadrpd -O sampleapplite.rpd -SI bi -U weblogic -S
server1.example.com -N 7777 -SSL
```

Using Online and Offline Repository Modes

You can open a repository for editing in either online or offline mode. The tasks you can perform depend on the mode in which you opened the repository.

This section contains the following topics:

- Editing Repositories in Offline Mode
- Editing Repositories in Online Mode
- Checking Out Objects
- Checking In Changes
- About Read-Only Mode

Editing Repositories in Offline Mode

Use offline mode to view and modify a repository while it is not loaded into the Oracle BI Server. If you attempt to open a repository in offline mode while it is loaded into the Oracle BI Server, the repository opens in read-only mode. Only one Administration Tool session at a time can edit a repository in offline mode. See "About Read-Only Mode" for more information.

You do not need to enter a user name and password to open a repository in offline mode. You only need to enter the repository password.

This section contains the following topics:

- Opening Repositories in Offline Mode
- Publish Offline Changes

Opening Repositories in Offline Mode

Follow these steps to open an RPD-format repository in offline mode:

1. In the Administration Tool, select **File > Open > Offline**.
2. Go to the repository you want to open, and then select **Open**.
3. In the Open Offline dialog, enter the repository password, and then click **OK**.

   If the server is running and the repository you are trying to open is loaded, the repository opens in read-only mode. If you want to edit the repository while it is loaded, you must open it in online mode. Also, if you open a repository in offline mode and then start the server, the repository becomes available to users. Any changes you make become available only when the server is restarted.

When you open an RPD-format repository in the Administration Tool in offline mode, the title bar displays the name of the open repository (for example, SampleAppLite).

You can also open MDS XML format repositories in offline mode, as follows:

1. In the Administration Tool, select **File > Open > MDS XML**.
2. Select the root folder location for your MDS XML files and click **OK**.
3. If this is the first time you have opened this MDS XML repository in the Administration Tool, you are prompted to specify whether this repository is a standalone MDS XML repository, or whether it is under source control. Select the appropriate option and click **OK**.

When you open an MDS XML format repository in the Administration Tool, the title bar displays the format and root folder location (for example, MDS XML C:\Root_ Folder).

Publish Offline Changes

Follow these steps to publish changes made to your repository in offline mode:

1. Publish the repository using the upload repository command. See "Upload Repository Command" for more information.

   You cannot upload MDS XML format repositories. To publish changes made to MDS XML repositories, you must first convert the repository to RPD format.
2. Restart all Oracle BI Server instances. You do not need to restart other BI system components.
3. In Presentation Services, click the **Reload Files and Metadata** link from the Administration page.

Editing Repositories in Online Mode

Use online mode to view and modify a repository while it is loaded into the Oracle BI Server. The Oracle BI Server must be running to open a repository in online mode. There are certain things you can do in online mode that you cannot do in offline mode. In online mode, you can perform the following tasks:

- Manage user sessions
- Manage the query cache
- Manage clustered servers
- Use the Oracle BI Summary Advisor (Oracle Exalytics Machine deployments only)

This section contains the following topics:

Opening Repositories in Online Mode

Follow these steps to open a repository in online mode:

1. In the Administration Tool, select **File > Open > Online** to display the Open Online Repository dialog.

   The Oracle BI Server DSNs that have been configured on your computer are displayed in the dialog. If no additional DSNs have been configured for this version of the Oracle BI Server, you might see only the default DSN that is configured for you during installation.

   See "Integrating Other Clients with Oracle Business Intelligence" in *Oracle Fusion Middleware Integrator's Guide for Oracle Business Intelligence Enterprise Edition* for information about how to create an ODBC DSN for the Oracle BI Server.

2. Provide a valid user name and password.

   The user name that you provide must have the ManageRepositories permission. See *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition* for information.

3. In a multitenant environment, specify the details for your tenant in the **Tenant info** field. Leave the **Tenant info** field blank if multitenancy is not configured.

   For multitenancy, enter the details in the form tenantguid:servicename, for example 1234101:service1. Contact the tenant administrator to obtain the GUID and service name. See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for information on GUIDs for tenants in the Identity Store. The Oracle BI Server uses the details that you specify to open the repository that is appropriate for your tenant.

4. If you expect to work extensively with the repository (for example, you plan to check out many objects), select the **Load all objects on startup** option. This loads all objects immediately, rather than as selected. The initial connect time might increase slightly, but opening items in the tree and checking out items is faster.

5. Select the appropriate DSN and click **OK**.

When you open a repository in the Administration Tool in online mode, the title bar displays the DSN for the Oracle BI Server to which you are connected, not the name of the current repository.

Publishing Online Changes

For a single-node deployment, changes made using the Administration Tool in online mode are available after reloading the metadata in Presentation Services by clicking the **Reload Files and Metadata** link from the Administration page.

In a clustered deployment, changes are published to the repository publishing directory, specified on the Repository tab of the Deployment page in Fusion Middleware Control. The master Oracle BI Server consumes these changes automatically, but you must restart all slave Oracle BI Servers for them to get the latest changes, and then reload metadata in Presentation Services by clicking the **Reload Files and Metadata** link from the Administration page.

You can restart the slave Oracle BI Servers using the RollingRestart ODBC procedure, or you can restart the slave servers using Fusion Middleware Control:

- To use the RollingRestart ODBC procedure, enter the following in `nqcmd`:

  ```
  call RollingRestart(timeout);
  ```

  where *timeout* is the number of seconds to wait for each slave Oracle BI Server to restart before moving on to the next one.

  For example:

  ```
  call RollingRestart(300);
  ```

  In this example, the system waits five minutes for each Oracle BI Server to restart. If the given Oracle BI Server restarts sooner, the system moves on to the next one immediately.

  See "Using nqcmd to Test and Refine the Repository" for more information about using `nqcmd`.

  Note: You must run the RollingRestart procedure directly against the master Oracle BI Server. Because the DSN created upon install for each Oracle BI Server is clustered by default, you must manually create a non-clustered DSN for the master Oracle BI Server to run the procedure against.

  See "Integrating Other Clients with Oracle Business Intelligence" in *Oracle Fusion Middleware Integrator's Guide for Oracle Business Intelligence Enterprise Edition* for information about how to create an ODBC DSN for the Oracle BI Server.

- To restart the slave servers using Fusion Middleware Control, first use the Cluster Manager in the Administration Tool in online mode to determine which Oracle BI Server is the master, and which are the slaves. Then, use the Process tab of the Availability page Fusion Middleware Control to restart the slave Oracle BI Servers. See "Using Fusion Middleware Control to Start and Stop Oracle Business Intelligence System Components and Java Components" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

It is a best practice to avoid making other configuration changes in Fusion Middleware Control or the configuration files when using the RollingRestart ODBC procedure or when restarting the slave Oracle BI Servers in Fusion Middleware Control. Because only the slave servers are restarted, a situation might result where the master Oracle BI Server has a different set of configuration settings loaded than the slave Oracle BI Servers. If this occurs, restart the master Oracle BI Server.

Guidelines for Using Online Mode

Use online mode only for small changes that do not require running consistency checks. Running consistency checks against the full online repository can take a long time. Instead, make more complex changes that require consistency checks in offline mode against a project extract of the repository.

Table 2–8 provides guidelines for when to perform online and offline edits.

Table 2–8 Guidelines for Online and Offline Repository Edits

Mode	Use This Mode For...	Example Use Cases
Online	■ Small changes that are required to fix things in a running system ■ Changes that need to be deployed quickly	■ Renaming Presentation layer metadata ■ Reorganizing Presentation layer metadata ■ Setting the logging level for an application role
Offline	■ Full-scale development or customization activities that require running consistency checks multiple times and iterating	■ Customizing existing fact or dimension tables ■ Adding new fact or dimension tables

In addition, you should limit the number of concurrent online users. The best practice is to have only one user working in online mode at a time. Even when users have different objects checked out, there might be dependencies between the objects that can cause conflicts when the changes are checked in. In general, only one user should make online changes in a single business model at a time.

If you must have multiple concurrent users in online mode, do not have more than five users. For situations where you need more than five users, use the multiuser development environment. See Chapter 3, "Setting Up and Using the Multiuser Development Environment" for more information.

Even with a single user making changes, be aware that online mode is riskier than offline mode because you are working against a running server. If you check in changes that are not consistent, it might cause the Oracle BI Server to shut down. When you work in online mode, make sure to have a backup of the latest repository so that you can revert to it if needed. You can also use **File > Undo All Changes** to roll back all changes made since the last check-in.

Checking Out Objects

When you are working in a repository open in online mode, you are prompted to check out objects when you attempt to perform various operations. Select the objects you want to check out and click **Yes** to check out the objects.

If you are performing a task in a wizard, the Checkout screen displays a summary of the objects that need to be checked out to complete the operation. Click **Next** to check out the objects and complete the task.

Checking In Changes

When you are working in a repository open in online mode, you are prompted to perform a consistency check before checking in the changes you make to a repository.

If you have made changes to a repository and then attempt to close the repository without first checking in your changes, a dialog opens automatically asking you to select an action to take. If you move an object from beneath its parent and then attempt to delete the parent, you are prompted to check in changes before the delete is allowed to proceed.

Use the Check in Changes dialog to make changes available immediately for use by other applications. Applications that query the Oracle BI Server after you have checked in the changes will recognize them immediately. Applications that are

currently querying the server will recognize the changes the next time they access any items that have changed.

To make changes available and have them saved to disk immediately:

- In the Administration Tool, select **File**, then select **Check In Changes**.

If the Administration Tool detects an invalid change, a message is displayed to alert you to the nature of the problem. Correct the problem and perform the check-in again. Note that you can select a message row and click **Go To**, or double-click a message row, to go directly to the affected object.

In some cases, you might see error 97005 (Transaction Failed). This error occurs when the Oracle BI Server does not accept the changes. In most cases, you can check the server log files to determine the cause of the problem.

You must save changes to persist the changes to disk. You must check in changes before you can save, but you do not need to save to check in changes.

About Read-Only Mode

Only one component (either the Oracle BI Server, or a single Administration Tool client in offline mode) can have a repository open in read/write mode at a time. If a second component opens a repository that is already in use, the repository is opened in read-only mode.

For example, assume the Oracle BI Server loads a repository in read/write mode. Any number of Administration Tool clients connecting to that repository in online mode will also get read/write mode, because they are accessing the repository through the Oracle BI Server. However, Administration Tool clients opening that repository in offline mode will get read-only mode, because the repository is already open for read/write through the Oracle BI Server.

Alternatively, assume an Administration Tool client opens a repository offline in read/write mode. When the Oracle BI Server starts, it will get read-only mode, as will any Administration Tool clients connecting to that repository in either offline or online modes. To enable the server to load the repository in read/write mode in this situation, you must first close the Administration Tool client that has the repository locked, and then restart the Oracle BI Server.

The Administration Tool also opens a repository in read-only mode when Oracle Business Intelligence has been clustered, and the Administration Tool is connected in online mode to a slave server. This occurs because the Master BI Server holds a lock on the repository. To avoid this situation when running in a clustered environment, ensure that the Oracle BI Server ODBC DSN used by the Administration Tool has been configured to point to the Cluster Controllers rather than to a particular Oracle BI Server.

Checking the Consistency of a Repository or a Business Model

Repository metadata must pass a consistency check before you can make the repository available for queries. The Consistency Check Manager lets you enable and disable rules for consistency checks, find and fix inconsistent objects, and limit the consistency check to specific objects. You can also use the `validaterpd` utility to check the validity of all metadata objects.

Note: A separate tool, Model Check Manager, identifies modeling problems that will affect Oracle BI Summary Advisor and aggregate persistence performance and results. Run Model Check Manager before running Oracle BI Summary Advisor or the Aggregate Persistence Wizard. See "Using Model Check Manager to Check for Modeling Problems" for more information.

This section contains the following topics:

About the Consistency Check Manager

The Consistency Check Manager checks the validity of your repository to ensure that it can load at run time, and to identify any syntax or semantic errors that may cause queries to fail.

In addition, running a consistency check might result in updates to your repository metadata. For example, invalid objects are deleted during Consistency Checks. This behavior might result in deleted expressions and filters on logical table sources and logical columns. Invalid references can occur when objects were deleted in the Physical layer without properly accounting for the references in the Business Model and Mapping layer objects.

Each time you save the repository, a dialog asks if you want to check global consistency. You have the following options:

- **Yes.** Checks global consistency and then saves the repository file.
- **No.** Does not check global consistency and then saves the repository file.
- **Cancel.** Does not check global consistency and does not save the repository file.

The Consistency Check Manager does not check the validity of objects outside the metadata using the connection. It only checks the consistency of the metadata and not any mapping to the physical objects outside the metadata. If the connection is not working or objects have been deleted in the database, the Consistency Check Manager does not report these errors.

The one exception to this rule is that the Consistency Check Manager does identify application roles that have been defined in the Administration Tool, but that have not yet been added to the policy store. Messages about placeholder application roles only appear when you perform a consistency check in online mode. Because of this, the set of consistency check messages returned for your repository might be different, depending on whether you have opened the repository in offline or online mode.

If you use lookup tables to store translated field names with multilingual schemas, note that consistency checking rules are relaxed for the lookup tables. See "Localizing Oracle Business Intelligence" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about localization and lookup tables.

The consistency checker returns the following types of messages:

- **Errors.** These messages describe errors that need to be fixed. Use the information in the message to correct the inconsistency, then run the consistency checker again. The following is an example of an error message:

  ```
  [38082] Type of Hierarchy '"0RT_C41"..."0RT_C41/MDF_BW_Q02"."Product Hierarchy
  for Material MARA"' in Cube Table '"0RT_C41"..."0RT_C41/MDF_BW_Q02"' needs to
  be set.
  ```

  If you disable an object and it is inconsistent, a message is displayed, asking if you want to make the object unavailable for queries.

- **Warnings.** These messages indicate conditions that may or may not be errors. For example, you might receive a warning message about a disabled join that was intentionally disabled to eliminate a circular join condition. Other messages may warn of inconsistent values, or feature table changes that do not match the defaults. The following is an example of a warning message:

  ```
  [39024] Dimension '"Paint"."MarketDim"' has defined inconsistent values in its
  levels' property 'Number of elements'.
  ```

In the Consistency Check Manager, you can sort the rows of messages by clicking the column headings. Additionally, the status bar provides a summary of all the rows displayed.

> **Note:** After upgrading from a previous software version and checking the consistency of your repository, you might notice messages that you had not received in previous consistency checks. This typically indicates inconsistencies that had been undetected before the upgrade, not new errors.

Checking the Consistency of Repository Objects

You can use the Administration Tool to check consistency in the following ways:

- To check consistency for all objects in the repository, select **File**, then select **Check Global Consistency**.
- To check the consistency of a particular repository object, such as a physical database, business model, or subject area, right-click the object and select **Check Consistency**.
- If you already have the Consistency Check Manager open, you can check global consistency by clicking **Check All Objects**.

To view the Consistency Check Manager without performing a global consistency check, select **Tools**, then select **Show Consistency Checker**. If you have checked consistency in the current session, the messages from the last check appear in the Messages pane.

To check the consistency of a repository:

1. In the Administration Tool, select **File**, then select **Check Global Consistency**. The Consistency Check Manager is displayed, listing any messages relating to the current repository.

> **Note:** If you disable an object and it is inconsistent, a dialog appears, asking whether you want to make the object unavailable for queries.

2. In the Consistency Check Manager, perform any required tasks using Table 2–9 as a guide.

Table 2–9 Consistency Check Manager Options

Option	Description
Show Qualified Name	Select this option to show the qualified name in the table's Object column.
Objects...	Click this option to see a list of all of the objects that were checked. This list displays objects by name and type.
Stats...	Click this option to see a list of errors and warnings by numeric code and the corresponding number of occurrences (for example, "Warning 38028, 10").
Save As...	Click this option to save the messages in the text, CSV, or XML format.
Check All Objects	Click this option to check consistency again and perform a global check. Or click the **Refresh** button in the top right corner to check only the objects that were listed as inconsistent in the last check.
Edit...	Select a row and click this option to open the properties dialog where you can edit the object.
Go To	Select one or more objects and click this option to go to the objects in the Administration Tool view of the repository. The selected objects appear highlighted in the Physical, Business Model and Mapping, or Presentation layer. Note that the Query Repository dialog closes when you choose this option.
Copy	Select one or more rows and click this option to copy the messages so that you can paste them in another file such as a spreadsheet. Note that if you click this option without selecting any rows, then all messages will be copied.

3. When finished, click **Close**.

To check the consistency of a single object in a repository:

1. In the Administration Tool, right-click an object, then select **Check Consistency**.

   If the object is not consistent, the Consistency Check Manager appears and displays a list of messages.

2. In the Consistency Check Manager, perform any required tasks using Table 2–9 as a guide.

3. When finished, click **Close**.

Using the validaterpd Utility to Check Repository Consistency

You can use the Oracle BI Server utility `validaterpd` to check the validity of all metadata objects in a repository. Running this utility performs the same validation checks as the Consistency Check Manager in the Administration Tool.

The `validaterpd` utility is available on both Windows and UNIX systems. You can run `validaterpd` against a binary RPD file, against an XML file based on the Oracle BI Server XML API, or against a set of MDS XML documents.

The location of the `validaterpd` utility is:

ORACLE_HOME/user_projects/domains/bi/bitools/bin

Using `validaterpd` with the `-L` option checks your repository metadata for issues that might affect the success of Oracle BI Summary Advisor or the aggregate persistence engine. See "Checking Models Using the validaterpd Utility" for more information about using `validaterpd` with the `-L` option.

Syntax

The `validaterpd` utility takes the following parameters:

```
validaterpd {-R repository_name | -I input_file_pathname |
-D MDS_XML_document_directory} [-P repository_password] {-O output_txt_file_name |
-C output_csv_file_name | -X output_xml_file_name} [-8] [-F fixed_rpd_name] [-S]
[-B]
```

Where:

repository_name is the name and path of the binary RPD file that you want to validate.

input_file_pathname is the name and path of the XML input file that you want to validate.

MDS_XML_document_directory is the location of the input MDS XML documents.

repository_password is the password for the repository that you want to validate.

Note that the *repository_password* argument is optional. If you do not provide the password argument, you are prompted to enter the password when you run the command. To minimize the risk of security breaches, Oracle recommends that you do not provide password arguments either on the command line or in scripts. Note that the password argument is supported for backward compatibility only, and will be removed in a future release. For scripting purposes, you can pass the password through standard input.

output_txt_file_name is the name and path of a text file where the validation results will be recorded.

output_csv_file_name is the name and path of a csv file where the validation results will be recorded.

output_xml_file_name is the name and path of an XML file where the validation results will be recorded.

Specify `-M` to specify that you want to execute MDS XML documents. If you specify `-D`, the `-M` argument is not needed. You only need to specify `-M` when you have a single MDS XML file that contains all the object definitions.

-8 specifies UTF-8 encoding in the output file.

Specify -F to create a new version of the repository in RPD format that includes automatic fixes for some internal validation errors. For *fixed_rpd_name*, provide the name and path of a binary RPD file where you want to save the fixes.

Specify -S to check server errors and navigation spaces only.

Specify -B to skip checks for business models availability.

Examples

The following example generates an output file called results.txt that contains validation information for the repository called repository.rpd, and saves a fixed version to fixed_repository.rpd:

```
validaterpd -R repository.rpd -O results.txt -F fixed_repository.rpd
```

```
Give password: my_rpd_password
```

The following example generates an output file called results.csv that contains validation information for the repository contained in the MDS XML documents located at C:\MDS_dir:

```
validaterpd -D C:\MDS_dir -C results.csv
Give password: my_rpd_password
```

Note: Be sure to provide the full pathnames to your repository files, both the input files and the output files, if they are located in a different directory.

Common Consistency Check Messages

Table 2–10 provides information about some commonly seen consistency check warnings and errors. Note that Table 2–10 provides a partial list only and does not show all possible warnings and errors.

Table 2–10 Common Consistency Check Messages

Validation Rule Example	Type	Description
[14031] The content filter of a source for logical table: FACT_ TABLE_NAME references multiple dimensions.	Error	The given logical table has a logical table source with a WHERE clause filter that references multiple dimensions. A WHERE clause with multiple dimensions is invalid.
[38126] 'Logical Table' ""Technology - WFA"."Fact WFA WO "' has name with leading or trailing space(s).	Error	Identifies an object with leading or trailing spaces in the object name. Repository objects can no longer have leading or trailing spaces in their names. Leading and trailing spaces in object names can cause query and reporting issues.
[38012] Logical column DIM_ Start_Date.YEAR_QUARTER_NBR does not have a physical data type mapping, nor is it a derived column. [38001] Logical column DIM_ Start_Date.YEAR_QUARTER_NBR has no physical data source mapping.	Error	Logical columns that are not mapped to any logical table source are reported as consistency errors, because the logical table source mappings are invalid and would cause queries to fail. Both of the given validation rules are related to the same issue.
[39062] Initialization Block 'Authorization' uses Connection Pool '"My_DB". "My_CP"' which is used for report queries. This may impact query performance.	Warning	Indicates that the same connection pool is being used for both queries and for initialization blocks. This configuration is not recommended. Instead, create a dedicated connection pool for initialization blocks. Otherwise, query performance might suffer, or user logins might hang if authorization initialization blocks cannot run.
[39028] The features in Database 'MyDB' do not match the defaults. This can cause query problems.	Warning	Some database feature defaults were changed in this release of Oracle BI EE. Unless you have specific customizations to your feature set, it is recommended that you reset your database features to the new defaults.
[39003] Missing functional dependency association for column: DIM_Offer_End_ Date.CREATE_DT.	Warning	This warning indicates that the given column is only mapped to logical table sources that are disabled. The warning brings this issue to the repository developer's attention in case the default behavior is not desired.

Table 2–10 (Cont.) Common Consistency Check Messages

Validation Rule Example	Type	Description
[39059] Logical dimension table MY_DIM has a source MY_DIM_DAILY at level Daily that joins to a higher level fact source MY_FACT_SUM.MTHLY_SUM	Warning	Even though this fact logical table source has an aggregate grain set in this dimension, no join was found that connects to any logical table source in this dimension (or a potentially invalid join was found). This means that either no join exists at all, or it does exist but is potentially invalid because it connects a higher-level fact source to a lower-level dimensional source. Such joins are potentially invalid because if followed, they might lead to double counting in query answers. For example, consider Select year, yearlySales. Even if a join exists between monthTable and yearlySales table on yearId, it should not be used because such a join would overstate the results by a factor of 12 (the number of months in each year). If you get a 39059 warning after upgrading, verify that the join is as intended and does not result in incorrect double counting. If the join is as intended, then ignore the 39059 warning.
[39055] Fact table "HR"."FACT - HC Budget" is not joined to tables in logical dimension "HR"."DIM - HR EmployeeDim". This will cause problems when extracting project(s).	Warning	This warning indicates that there is a physical join between the given fact and dimension sources, but there is not a corresponding logical join between the fact table and the dimension table.
[39054] Fact table "Sales - STAR"."Fact - STAR Statistics" is not joined to logical dimension table "Sales - STAR"."Dim - Plan". This will cause problems when extracting project(s).	Warning	This warning indicates that the aggregation content filter "Group by Level" in the logical table source of a fact table references logical dimension tables that are not joined to that fact table. If that fact table is extracted in the extract/MUD process, the dimensions that are not joined will not be extracted. In this case, the aggregation content of the extracted logical table source would not be the same as in the original logical table source.
[39057] There are physical tables mapped in Logical Table Source ""HR"."Dim - Schedule"."SCH_DEFN"" that are not used in any column mappings or expressions.	Warning	This warning indicates that the given logical table source has irrelevant tables added that are not used in any mapping. This situation will not cause any errors.

3

Setting Up and Using the Multiuser Development Environment

This chapter explains how to set up and use the multiuser development environment in Oracle Business Intelligence, including defining projects, setting up the multiuser development directory, checking out and publishing changes to projects, and merging metadata.

Multiuser development (MUD) provides a mechanism for concurrent development on overlapping code bases. Oracle Business Intelligence provides a MUD environment that manages subsets of metadata, in addition to multiple users, by providing a built-in versioning system for repository development.

See also the following resources:

- "Managing the Repository Lifecycle in a Multiuser Development Environment" for additional information about working in a multiuser development environment
- Chapter 4, "Using a Source Control Management System for Repository Development" for information about saving your repository in MDS XML format and integrating the Administration Tool with a third-party source control management system

This chapter contains the following topics:

About the Multiuser Development Environment

In Oracle Business Intelligence, multiuser development facilitates the development of application metadata in enterprise-scale deployments. Application metadata is stored in a centralized metadata repository (RPD) file. The Administration Tool is used to work with these repositories. You do not use a multiuser development environment with MDS XML-format repositories.

The following are examples of how you might use a multiuser development environment:

- Several developers work concurrently on subsets of the metadata and then merge these subsets back into a master repository without their work conflicting with other developers. For example, after completing an implementation of data warehousing at a company, an administrator might want to deploy Oracle Business Intelligence to other functional areas.

  Note: In this chapter, "master repository" refers to the copy of a repository in the multiuser development directory

- A single developer manages all development. For simplicity and performance, this developer might want to use the multiuser development environment to maintain metadata code in smaller chunks instead of in a large repository.

In both examples, an administrator creates projects in the repository file in the Administration Tool, then copies this repository file to a shared network directory (called the multiuser development directory). Developers are able to check out projects, make changes and then merge the changes into the master repository. When developers check out projects using the Administration Tool, files are automatically copied and overwritten in the background. Therefore, it is important for the administrator to perform setup tasks and for the developers to perform check-out and merge/publish procedures carefully, paying close attention to the Administration Tool messages that appear.

When developers check out projects, repository files are not automatically copied or overwritten. Instead, the Administration Tool creates two new files when projects are checked out: one to hold the original project data, and one to hold the project changes.

For example, when a repository developer checks out project A from master.rpd in the C:\multiuser development directory, the Administration Tool extracts all metadata related to project A and prompts the developer for a new file name to save the data. When the developer chooses a new file name, for example Mychanges.rpd, the Administration Tool creates two new files:

- A file called MyChanges.rpd that will contain the changes made by the developer
- A file called originalMyChanges.rpd that contains the original project data

The Administration Tool determines the developer's changes by comparing the Mychanges.rpd with the originalChanges.rpd. This information about what has changed is needed during the multiuser development merge process.

Note: To reduce storage needs, repositories in Oracle Business Intelligence Enterprise Edition 12*c* Release 1 (12.2.1) are stored in a compressed format. Because of this, you might notice that the size of an RPD file opened and saved in this release is significantly smaller than the size of RPD files from previous releases.

About the Multiuser Development Process

Multiuser development presupposes a clear understanding of customer technical and business objectives. It also requires that you follow clearly defined development processes and adhere rigorously to those processes, including consistent merging and reconciliation practices.

The following procedure shows the general steps to follow when deploying a multiuser development environment. The first three steps are usually performed by an administrator, and the remaining steps are usually performed by one or more developers.

> **Tip:** Oracle recommends that developers merge their changes as soon as possible or as often as possible. Frequent merges makes conflict resolution easier and simplifies the merge.

To deploy a multiuser development environment:

1. Define projects to organize voluminous metadata into manageable components. See "Creating Projects" for more information. Consider these tips:
   - Use smaller RPDs to shorten and simplify development effort and unit testing.
   - Organize development resources by projects to spread workload and reduce inconsistencies and overwrites.
2. Set up a shared network directory to use as the multiuser development directory.
3. Copy the master repository to the multiuser development directory.
4. Extract one or more projects or the entire repository for local development.
5. Merge repository objects and resolve conflicts.
   - Because metadata objects are often highly interrelated, several developers could be working on the same objects.
   - You can perform regular subset refreshes to merge your local changes with the latest version of the master. When configuration conflicts occur during the merge process, developers are prompted for the correct process.
6. Publish changes to the network.
   - A final subset refresh (merge) is performed during the publishing step. Many developers can simultaneously work on the same objects, but only one can publish at a time. The repository is locked during the publishing step.
7. Use Logging and Backup features to identify points of erroneous or incorrect configuration.
   - The log file tracks multi-development activity, along with comments.
   - The master repository and developer repositories are automatically backed up for future reference and for use in manual rollback.

Setting Up Projects

Projects are the central enabler of metadata management. A project consists of a discretely-defined subset of the repository metadata, in the form of groups of logical stars with associated metadata. A project has the following characteristics:

- Is largely defined by logical fact tables in the applicable business model
- Automatically adds related logical dimension tables and other metadata during extract
- Can have one to many logical fact tables

For projects that are just beginning, the best practice is to begin with a repository containing all the necessary physical table and join definitions. In this repository, you create a logical fact table as a placeholder in the Business Model and Mapping layer

and a subject area as a placeholder in the Presentation layer. As you add business model and subject area metadata, new projects based on individual subject areas and logical facts can be created.

Follow these guidelines when setting up projects:

- Only one person at a time can create projects in a master repository.
- Do not delete projects unless they are no longer under active development.
- Choose your project name carefully when creating a project. Do not rename projects.
- Use care when removing objects from projects to avoid problems with repository extract/check-out.

This section contains the following topics:

About Projects

Projects can consist of Presentation layer subject areas and their associated business model logical facts, dimensions, groups, users, variables, and initialization blocks. Administrators can create projects so that developers and groups of developers can work on projects in their area of responsibility.

The primary reason to create projects is to support multiuser development. During the development process, you can split up the work (metadata) between different teams within your company by extracting the metadata into projects so that each project group can access a different part of the metadata.

In addition to multiuser development, you may want to create projects for licensing reasons. Before releasing a new software version, you may want to ensure that only the metadata that is relevant to the licensed application is in a project and that everything is consistent and complete. You can accomplish this by adding only the fact tables that are relevant to the application.

Project extractions are fact table centric. This ensures that project extracts are consistent and makes licensing much easier to manage.

About the Project Dialog

In the Project dialog, the left pane contains objects that you can use to create a project. The objects in the right pane are all the objects you chose (directly or indirectly) that reflect the complete set of data that makes each addition consistent. For example, if you select a subject area from the top node of the left-hand tree to add to your project, underlying fact tables of other subject areas are automatically added if needed to make the extract consistent.

The following describes the left pane of the Project dialog:

- You can choose to group fact tables by Business Model or Subject Area, to help select the fact tables you want. Typically, grouping fact tables according to which ones are used by a particular subject area is a more convenient way to choose fact tables for your project. Note that a fact table can be associated with multiple subject areas, but belongs to one and only one business model.

Although it appears that you can add a subject area from the top node when you group facts by subject area, you are actually adding only the underlying fact tables. The subject areas only appear as choices to help you to add the elements you want in your project. Additionally, it adds any other objects that are necessary to make the extract consistent. To add an actual subject area, use the Presentation node at the bottom of the tree.

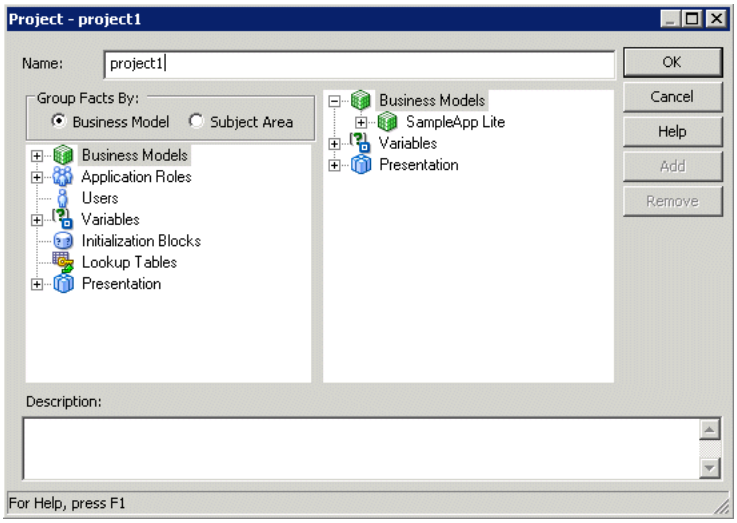
- When grouped by business model, the left pane displays only facts that belong to the business model.
- The Presentation node contains presentation layer objects. You must explicitly include these objects in your project if you want to work with them; they are not added automatically.

  If you add presentation objects that are not related to any fact tables in the project, a warning appears when you click **OK**. The consistency checker also notes this discrepancy.

The right pane of the Project dialog shows the objects you select to be extracted, such as fact tables (under the Business Models folder), Presentation layer objects (under the Presentation folder), users, application roles, variables, and initialization blocks. These objects are extracted when you click **OK**.

Figure 3–1 shows the Project dialog.

Figure 3–1 Project Dialog with Fact Tables Grouped by Business Model



Creating Projects

When creating a project, you typically select a subject area or a subset of logical fact tables related to the selected subject area, and the Administration Tool automatically adds any business model and Physical layer objects that are related. An object can be part of multiple projects. Alternatively, if you choose to group facts by business model, you can select a particular business model or a set of logical fact tables that are part of a business model. You also need to explicitly add any Presentation layer objects if you want them to be part of your project.

Note that although the project definition itself does not include Physical layer objects, these objects are extracted and determined through the project definition.

After you create projects, they become part of the metadata and are available to multiple developers who need to perform development tasks on the same master

repository. When defined this way, projects typically become a consistent repository after a developer checks out the projects and saves them as a new repository file.

To create a project for a multiuser development environment:

1. In the Administration Tool, choose **File > Open > Offline**.
2. In the Open dialog, select the repository that you want to make available for multiuser development, then click **OK**. Provide the repository password, then click **OK** again.
3. Select **Manage**, then select **Projects**.
4. In the Project Manager dialog, in the right pane, right-click and then select **New Project**.

   The left pane contains the objects that are available to be placed in a project. The right pane contains the objects that you select to be part of the project.
5. In the Project dialog, type a name for the project.
6. Choose whether to group facts by business model, or subject area. It is typically more convenient to group facts by subject area.
7. Perform one or more of the following steps to add fact tables to your project:
   - In the left pane, select a subject area or business model and then click **Add**. The Administration Tool automatically adds all the associated logical fact tables.
   - In the left pane, expand the subject areas or business models and select one or more logical fact tables that are related to the subject area or that are within the business model, then click **Add**.

     The project is defined as explicitly containing the selected logical fact tables and implicitly containing all logical dimension tables that are joined to the selected logical fact tables (even though they do not appear in the right pane).

   See "About the Project Dialog" for more information about the objects that appear in the left and right panes.
8. To remove fact tables from the project, in the right pane, select a fact table and click **Remove**. You can also remove all fact tables associated with a subject area or business model by selecting a subject area or business model and clicking **Remove**.
9. Optionally, add any application roles, users, variables, initialization blocks, or lookup tables needed for the project. Although objects like variables and initialization blocks that are directly referenced by other extracted objects are included automatically, you might want to include objects in your project that are not referenced. For example:
   - If you are using initialization blocks for authentication, include any necessary initialization blocks.
   - Include repository variables or other objects that are not yet referenced by other objects, but that you might want to use in future repository development.
   - Include users and application roles that are currently being used, or will be used in the future, as part of your data access security settings.

   Tip: You may want to add the top node for each object type (for example, Variables), then selectively remove individual objects from the right pane.

10. Select the Presentation layer objects that you want to include in your project from the left pane and click **Add**. You must add these objects to see them in the project; they are not added automatically.

    You can also remove particular presentation tables or columns from the project definition by double-clicking the object in the right pane, or selecting the object and clicking **Remove**.

    Note: If you do not see the set of subject areas you expect after the project is created, edit the project to explicitly add the subject areas you need.

11. Click **OK**.

About Converting Older Projects During Repository Upgrade

When you upgrade a repository from Oracle Business Intelligence versions before 10.1.3.2, the project definition is upgraded. During the upgrade, the project definition, subject areas, target levels, list catalogs, and existing fact tables are automatically converted into simple fact tables in the following way:

- Get presentation columns related to the target levels through the qualifying keys.
- Get presentation columns related to the list catalogs through the qualifying keys.
- Get presentation columns related to the subject areas.
- Get all the logical columns from all the presentation columns.
- Get all the logical columns from the fact tables in the project.
- Get the fact tables from all the logical columns.

After the upgrade, projects contain only simple fact tables. All the security objects remain unchanged.

In addition, projects in repositories from any version before 12*c* Release 1 (12.2.1) are upgraded so that they explicitly contain Presentation layer objects. In previous releases, Presentation layer objects were implicitly included based on the permissions of the users included in the project.

Setting Up the Multiuser Development Directory

To prepare for multiuser development, an administrator performs the following tasks:

- Identify or create a shared network directory that will be dedicated to multiuser development.
- After creating all projects, copy the repository file in which you created the projects to the multiuser development directory where it will be used as your master repository for multiuser development.

After the administrator has identified the multiuser development directory and copied the repository file, developers must set up the Administration Tool to point to the multiuser development directory before they can check out projects.

This section contains the following topics:

Identifying the Multiuser Development Directory

After defining all projects, the administrator must identify or create a shared network directory (called the multiuser development directory) that all developers can access, and then upload the new master repository to that location. This shared network directory should be used only for multiuser development. This directory typically contains copies of repositories that need to be maintained by multiple developers. The multiuser development directory must be on a Windows system.

Developers create a pointer to the multiuser development directory when they set up the Administration Tool on their computers.

Caution: The administrator must set up a separate, shared network directory that is dedicated to multiuser development. If not set up and used as specified, critical repository files can be unintentionally overwritten and repository data can be lost.

Copying the Master Repository to the Multiuser Development Directory

After the multiuser development directory is identified, the administrator must copy the master repository file to the multiuser development directory. Projects from this master repository will be extracted and downloaded by the developers who will make changes and then merge these changes back into the master repository.

After you copy the repository to the multiuser development network directory, notify developers that the multiuser development environment is ready.

Even after starting repository development, it might be necessary to make manual changes to the master repository from time to time. See "Manually Updating the Master MUD Repository" for specific instructions on how to do this.

Setting Up a Pointer to the Multiuser Development Directory

Before checking out projects, developers working in the multiuser development environment must set up their local copies of the Administration Tool to point to the multiuser development directory on the network. The Administration Tool uses this location when the developer checks out and checks in objects in the multiuser development directory.

Note: Until the pointer is set up, the multiuser options are not available in the Administration Tool.

Initially, the network directory contains the master repositories. The repositories in this location are shared with other developers. Later, the network directory contains additional multiuser development history files, including historical subsets and repository versions. Do *not* manually delete any files in the multiuser development directory; these files are important and are used by the system.

When setting up the pointer, the developer can also complete the **Full Name** field. Although the field is optional, it is recommended that the developer complete this field to allow other developers to know who has locked the repository.

To set up a pointer to the multiuser development directory:

1. From the Administration Tool menu, choose **Tools > Options**.
2. In the Options dialog, click the Multiuser tab.
3. In the Multiuser tab, for **Multiuser development directory**, enter the full path to the network directory.

   Alternatively, click **Browse**, select the multiuser development directory, and then click **OK**.
4. In the **Full Name** field, type your complete name, then click **OK**.

Making Changes in a Multiuser Development Environment

During check-out, refresh, and publish, a copy of the master repository is temporarily copied to the developer's local repository directory (typically, *ORACLE_INSTANCE*\bifoundation\OracleBIServerComponent\coreapplication_obis*n*\repository by default). After checking out projects or the entire repository and making changes in a local repository file, each developer can publish (merge) changes into the master repository or discard the changes.

This section contains the following topics:

- About Changing and Testing Metadata
- Making Changes to a Repository Using Projects
- Making Changes to an Entire Repository
- About Multiuser Development Menu Options

About Changing and Testing Metadata

Most types of changes that can be made to standard repository files are also supported for local repository files. Developers can add new logical columns, add new logical tables, change table definitions, change logical table sources, and so on. Developers might also work simultaneously on the same projects or entire repository locally. Keep in mind, however, that Oracle Business Intelligence assumes the individual developer understands the implications that these changes might have on the master repository. For example, if a developer deletes an object in a local repository, this change is propagated to the master repository when local changes are merged without a warning prompt.

To ensure metadata integrity, do not remove a physical column unless there are no logical table source mappings to that physical column. Because of this, if you use a multiuser development environment, you cannot delete a logical column and its associated physical column at the same time. Instead, you must first delete the logical column and perform a merge. Then, you can delete the physical column and perform another merge to safely remove the object.

In some cases, logical column types can change over the course of MUD development, which results in unexpected logical column types. When this occurs, you can generate a list of logical columns and their types using the Generate Logical Column Type Document utility in the Administration Tool or `biservergentypexml`. Then use the Compare Logical Column Types utility for subsequent MUD versions to ensure that the logical column types match as expected. For example, you can generate a logical column type list for repository version 20, and use the Compare Logical Column Types utility to compare the list against repository version 30. See "Generating a List of Logical Column Types" and "Comparing Logical Column Types" for more information.

Changes to physical connection settings are not propagated to the master repository upon merge and publish. This ensures that developers can apply settings for their local test data sources to perform unit testing of their model changes without impacting other developers.

In addition to physical connection settings, security settings and database feature table changes are not retained in a multiuser development merge to prevent developers from overwriting passwords and other important objects in the master repository.

After making changes to a local repository, the developer uploads the repository and tests the edited metadata.

Note: DSNs that are specified in the metadata must exist on the developer's workstation.

Making Changes to a Repository Using Projects

The following sections provide information on making changes in a multiuser development environment using projects:

- Checking Out Repository Projects
- Refreshing the Local Project Extract

Checking Out Repository Projects

After setting up a pointer to the multiuser development default directory, a developer can check out projects, change metadata, and test the metadata. In the **File > Multiuser** submenu, the **Checkout** option is available only when there is a multiuser development directory defined in the Multiuser tab of the Options dialog.

This section contains the following topics:

- About Repository Project Checkout
- Checking Out Projects
- Using the extractprojects Utility to Extract Projects

About Repository Project Checkout During checkout, the Administration Tool performs the following tasks:

- In the developer's local repository directory, the Administration Tool makes a temporary copy of the master repository.

  Note: If a repository with that name exists in this location, the developer is asked to confirm overwriting the existing repository. If the developer clicks **Yes**, the existing local repository is immediately overwritten in the background and after the new repository is saved, the temporary master repository file is automatically deleted.

- In the developer's local repository directory, the Administration Tool saves a local copy of the selected projects in a new repository, such as Metadata1.rpd. The developer provides a name for the local copy. The developer makes metadata changes in this file. The number is incremented for each checkout for that session.
- In the developer's local repository directory, the Administration Tool saves a second local copy of the new repository, adding "original" as the prefix (for example, originalMetadata1.rpd).

- After the developer saves the new repository file, check out is complete. In the developer's local repository directory, the temporary copy of the master repository is automatically deleted.

> **Caution:** When the developer selects and saves the projects to a local repository file, the Administration Tool does not place a lock on the projects in the master repository on the shared network drive. Therefore, nothing physically prevents others from working on the same project. To determine if a project has been checked out, you need to look in the log file in the multiuser development directory on the shared network drive.

Checking Out Projects This section explains how to check out projects using the Administration Tool.

To check out projects:

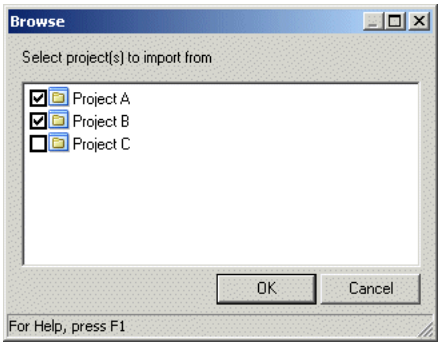
1. From the Administration Tool menu, choose **File > Multiuser > Checkout**.
2. If there is more than one repository in the multiuser development directory, then the Multiuser Development Checkout dialog is displayed. Select the appropriate repository, and click **OK**.

   This dialog is not displayed if there is only one repository in the multiuser development directory.
3. In the Extract from dialog, type the repository password, and click **OK**.

   If no projects exist in the repository, then a message is displayed and the repository does not open.
4. If there is more than one project in the master repository, then the Browse dialog is displayed. Select the projects that you want to be part of your project extract, and click **OK**.

   Figure 3–2 shows the Browse dialog for selecting projects.

Figure 3–2 Browse Dialog for Selecting Projects



   If only one project exists in the master repository, then it is selected automatically and the Browse dialog is not displayed.
5. In the Create new subset repository dialog, type a name for the new repository (for example, Metadata1.rpd) and click **Save**.

   A working project extract repository is saved on your local computer. The name is exactly as you specified and is opened in offline mode. A log file is also created.

Caution: A second copy of the project extract repository is saved in the same location. The name of this version contains the word "original" added to the beginning of the name that you assigned to the repository extract. Do not change the original project extract repository. It is used during the multiuser development merge process, and when you want to compare your changes to the original projects.

Using the extractprojects Utility to Extract Projects You can use the Oracle BI Server utility `extractprojects` to extract projects from a given repository without the overhead of the MUD environment. The `extractprojects` utility is available on both Windows and UNIX systems. You can use `extractprojects` only with binary repositories in RPD format.

The `extractprojects` utility generates an RPD file that includes the set of projects that you specify. The utility does not perform other tasks that are performed when you check out projects using the Administration Tool, such as saving an original repository file or tracking the extract as a check-out in the MUD directory.

The location of the `extractprojects` utility is:

ORACLE_HOME/user_projects/domains/bi/bitools/bin

Syntax

The `extractprojects` utility takes the following parameters:

```
extractprojects -B base_repository_name -O output_repository_name {-I input_
project_name} [-P repository_password] [-L] [-E project_list_file_name]
```

Where:

base_repository_name is the name and path of the repository from which you want to extract projects.

output_repository_name is the name and path of the repository generated by the utility.

input_project_name is the name of a project you want to extract. You can enter multiple projects. Be sure to precede each project entry with -I (for example, -I project1 -I project2). If the project name contains spaces, enclose it in double quotes (for example, "project 1").

repository_password is the password for the repository from which you want to extract projects.

Note that the *repository_password* argument is optional. If you do not provide the password argument, you are prompted to enter the password when you run the command. To minimize the risk of security breaches, Oracle recommends that you do not provide password arguments either on the command line or in scripts. Note that the password argument is supported for backward compatibility only, and will be removed in a future release. For scripting purposes, you can pass the password through standard input.

- L enables logging. When logging is enabled, a log file in the format ProjExtr.YYYYMMDD.HHMMSS.xml is created in the Oracle BI Server logging directory. For example:

```
ORACLE_INSTANCE/diagnostics/logs/OracleBIServerComponent/coreapplication_
obisn/ProjExtr.20100616.082904.xml
```

-E is an optional argument that lets you print a list of all projects contained in a repository into a file. Specify *project_list_file_name* after the option to specify the file name and location in which you want to store the project names. -E is only used with -B and -P and does not actually perform a project extract.

Note that -U and -F are visible in the syntax list, but are for internal use only.

Example

The following example extracts project1 and project2 from my_repos.rpd and creates a new repository called extract_repos.rpd:

```
extractprojects -B my_repos.rpd -O extract_repos.rpd -I project1 -I project2
Give password: my_rpd_password
```

Note: Be sure to provide the full pathnames to your repository files, both the input file and the output file, if they are located in a different directory.

Refreshing the Local Project Extract

Use the **Refresh Subset** option to refresh the local project extract with any changes that were made to the master repository. This option also merges the latest changes with the local project extract. Perform this task as an incremental step during development before publishing your final changes at the end of your development session.

It is a best practice to refresh your local project extract frequently so that conflicts during merge can be identified and dealt with on an incremental basis. If too many changes are merged at one time, then making the appropriate merge decisions for conflicts can be confusing and error-prone.

Making Changes to an Entire Repository

The preferred method for making changes to a repository in a multiuser development environment is to use projects. You might encounter situations during which you want to make changes that involve you accessing the entire repository at one time.

If you want to access the entire repository, including objects not assigned to projects, then select the **Whole Rpd Checkout** option in the **File > Multiuser** submenu. Multiple developers can access the entire repository at one time. Use this method only when necessary, because system performance will likely decrease, especially during merges of large repositories.

About Multiuser Development Menu Options

After the local developer makes changes, tests the changes, and saves the repository locally, the local developer can perform the following tasks from the **File > Multiuser** submenu:

- **Compare with Original.** Compares the working extracted local repository to the original extracted repository. When you select this option, the Compare repositories dialog is displayed and lists all the changes that were made to the working extracted repository since you checked out the projects or the entire repository.

- **Refresh Subset.** Refreshes the local project extract with any changes that were made to the master repository. The changes from the master are merged with your local changes.

  If changes have been made to the master repository, then the old project extract file (original*filename*.rpd) is replaced with a new project extract file called current*filename*.rpd.

- **Publish to Network.** Publishes changes made to the local project extract or the entire repository to the master repository. A lock is acquired to lock the master repository during the publish step. Publishing the changes automatically performs a Refresh Subset operation to merge the local changes with any additional changes from the master. Then, the merged changes are published to the master repository, the lock is released, and the local repository is closed.

- **Undo Publishing.** Used when mandatory consistency checks are enforced during the publishing step, and errors occur. When you are notified of consistency check errors during publishing, you can choose to fix the errors immediately, as part of the publishing step. The master repository is locked during this process. If you need to release the lock on the master and fix the changes later, then select **Undo Publishing** to release the lock and return to the latest subset extract repository.

- **Discard Local Changes.** Any time after check out and before publishing, you can discard your changes. When you select this option, the working repository closes without giving you an opportunity to save your work.

  Caution: If you select this option, there is no opportunity to change your mind. For example, no confirmation dialog is displayed.

Publishing Changes to Multiuser Development Repositories

After changing and testing the metadata on a local computer, the developer must publish local changes to the master repository in the multiuser development directory.

The Oracle BI repository development process uses a three-way merge to manage concurrent development. Metadata merges are done first on local environments and then merged with the master repository. A three-way merge identifies local changes based on the following repository characteristics:

- The Master RPD
- The Baseline RPD or Master RPD snapshot at time of project extraction
- The current locally developed and changed RPD

Changes are managed by merge and reconciliation. Most of the merging process is automatic, and changes do not conflict. In case of any conflicting metadata sources, developers can locate and resolve them.

An administrator can also merge the changes from multiple repositories manually, or import objects from different repositories outside of a particular MUD environment.

Ensure that you merge changes frequently. The merge process is very complex and can become difficult if there are too many changes. See Appendix D, "Merge Rules" for more information about how objects are merged during the merge process.

It is a best practice to refresh your subset repository regularly to identify conflicts early. Refreshing your subset performs a subset remerge with the latest version of the master, then leaves the repository open for you to continue making changes until you are ready to publish.

This section contains the following topics:

About the Multiuser Development Merge Process

The merge process involves the following files:

- **Local (subset) repository, either original or current (refreshed).** The local subset repository can be one of the following:
  - If no subset refresh has been performed, contains the state of the projects or the entire repository as originally extracted. This repository name begins with "original" (for example, originalDevelopment2.rpd).
  - If a subset refresh has been performed, contains the state of the projects or the entire repository since the last merge that occurred during the subset refresh. The repository name begins with "current" (for example, currentDevelopment2.rpd).
- **Modified local (subset) repository.** Contains the extracted projects after being modified by the developer. This version is stored in the same location as the original or current version of the local subset repository.
- **Latest master repository.** The latest master is copied locally for the merge, then published to the multiuser development directory after the merge. Note that this file might have been modified by other developers before this merge.

During the merge, the Administration Tool checks for added objects and if found, a warning message is displayed. The following list describes what happens during this step:

- **Warning about added objects.** A developer who checks out a project has the ability to modify that project in any way and check it back in. Deletions and modifications are ways in which the integrity of the project is maintained. However, adding objects might introduce objects into the repository that do not belong to any project. Therefore, all project-related objects are checked and if a new object is found, a warning message is displayed.

  Caution: You must add newly created metadata to the project definition in the master repository for it to be visible in future extracted versions of the project. For example, if a developer checks out a project, adds a new object, and checks it in, then the new object is not visible in extracted versions of the project until it is explicitly added to the project definition. See "Creating Projects" for instructions.

- **Aggregation of related objects.** In the warning message, only the parent object is reported. The Administration Tool aggregates all the objects to make the message more usable. For example, if a developer added a new business model, only the business model name is included in the warning message to the user, not the names of the tables, columns, dimensions, and so on.

When a developer publishes changes to the network, the following actions occur:

- The master repository in the multiuser development directory is overwritten with the repository that contains the developer's changes.

- The *master_repository*.lck file is deleted. If another developer checks out the changed project from the master repository or checks out the entire repository, then the changes made by the first developer are exposed to the other developer.

How Are Multiuser Merges Different from Standard Repository Merges?

The multiuser development publishing process uses the same technology as the standard repository merge process with a few important differences. For example, for MUD merges, you typically do not want to retain changes to security settings and data sources, to prevent developers from overwriting passwords and other important objects in the master repository. Because of this, changes to security settings and data source connections are not retained when you perform a MUD merge. In addition, inserts (created objects) are applied automatically.

See "Merge Rules and Behavior for Multiuser Development Merges" for full details.

Publishing to the Network

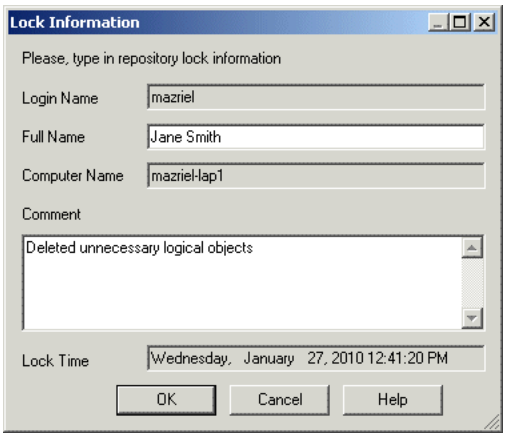
When the publishing process begins, the Administration Tool automatically copies the current version of the master repository from the multiuser development directory to the local repository directory on the developer's computer (typically *ORACLE_INSTANCE*\bifoundation\OracleBIServerComponent\coreapplication_obis*n*\repository) and updates the log files in the local and multiuser development directories. This is necessary because the master repository in the multiuser development directory might have changed after the developer checked out the projects, or since the last subset refresh.

To publish local changes to the master repository:

1. In the Administration Tool, select **File > Multiuser > Publish to Network**, then click **Yes** if prompted to save changes.
2. In the Lock Information dialog, in the **Comment** field, type a description of the changes that you made, then click **OK**.

   Figure 3–3 shows the Lock Information dialog.

Figure 3–3 Lock Information Dialog



3. If there are any conflicts, then the Merge Repository Wizard opens and the Define Merge Strategy screen is displayed. Make merge decisions about whether to include or exclude objects by selecting **Current** or **Modified** from the **Decision** list. When you select an object in the decision table, the read-only text box below the decision table describes what changes were made to that object in the current

repository. You can also click **View Change Statistics** to see a summary of changes. Click **Finish** when you are finished making merge decisions.

See "Performing Full Repository Merges" for additional information about the Define Merge Strategy screen.

A lack of conflicts does not mean that there are no differences between the repositories. Instead, it means that there are no decisions that have to be explicitly made by the developer to publish changes. See "How Are Multiuser Merges Different from Standard Repository Merges?" for information about conflicts that are resolved automatically in a MUD merge.

In both cases, a CSV file is created in the local directory that contains details of the merged changes.

4. After you confirm all the changes, click **Save**.

   The master repository in the multiuser development directory is overwritten with the copy of the repository containing the developer's changes.

Enforcing Consistent Repositories When Publishing Changes

You can enforce a mandatory consistency check during the Publish to Network step by setting the Mandatory Consistency Check option to Yes in the multiuser development options file. See "Setting Multiuser Development Options" for more information about the options file.

When this option is set to Yes, the Consistency Checker runs as part of the publishing process. Publishing cannot proceed unless no errors exist in the given repository.

If consistency errors occur, then a message is displayed. Select one of the following options:

- **Yes:** Select this option to fix the consistency check errors immediately. The master repository remains locked during this process.

  If you select this option and change your mind (such as when you need to release the lock on the master, or when fixing the changes is more complex than you anticipated), then select **File > Multiuser > Undo Publishing**. This option unlocks the master repository and returns you to your latest subset repository (as an extract from the currently merged master repository).

- **No:** Select this option to cancel the publishing step and fix the consistency check errors later. The master repository is unlocked, and you are returned to your latest subset extract repository.

Branching in Multiuser Development

Branching is a further refinement of the merging development process. Branching can provide higher efficiencies over large development teams that have overlapping releases, but it requires significant administrative overhead.

This section contains the following topics:

- About Branching
- Using the Multi-Team, Multi-Release Model in Oracle Business Intelligence
- Synchronizing RPD Branches

About Branching

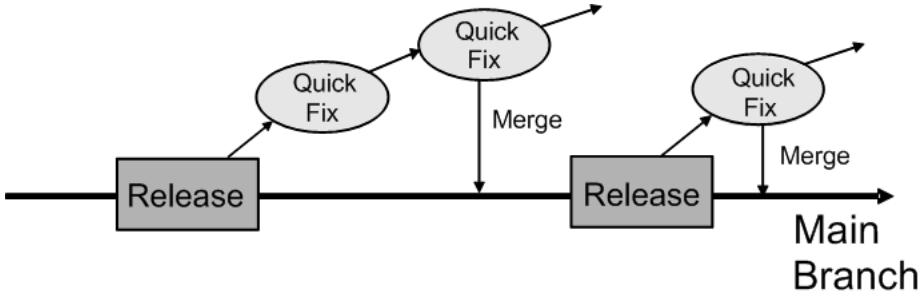
In branching, developers work on private branches to isolate their code from other developers and merge changes back to the main branch. Different strategies can be followed, depending on the size of the development team.

In the **Simple Development Model**, all development occurs on a single main branch. This strategy has the following characteristics:

- Only for emergency fixes
- Checkouts might not be the most current code
- Carries a stability risk for the mainline branch

Figure 3–4 shows the Simple Development Model.

Figure 3–4 Simple Development Model

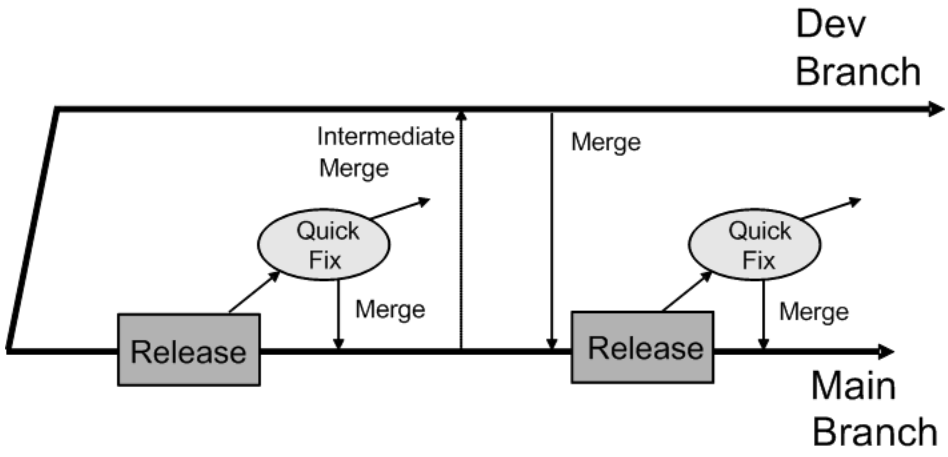


In the **Small Team Development Model**, development occurs on a single Dev branch, with a separate Main branch strictly for releases. This strategy has the following characteristics:

- The Mainline is the official release branch
- Development occurs on a separate branch
- Stable code is merged back to Main at key milestones
- Branches are synchronized periodically

Figure 3–5 shows the Small Team Development Model.

Figure 3–5 Small Team Development Model

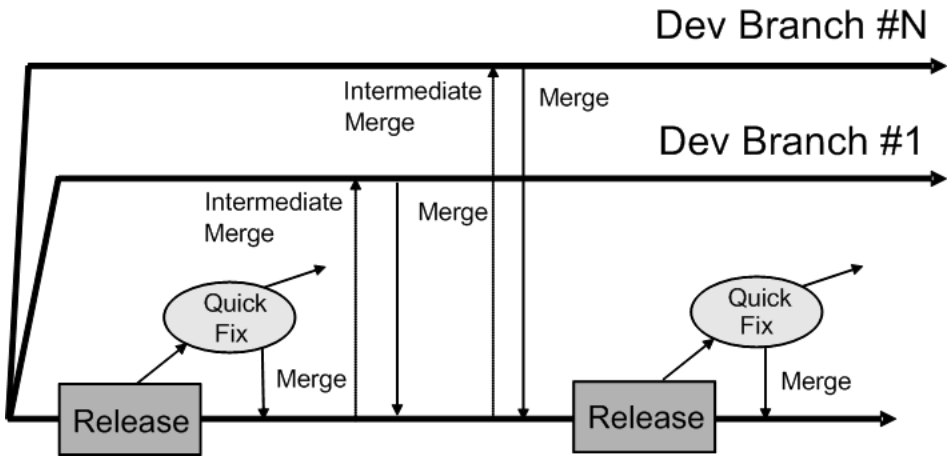


In the **Multi-Team, Multi-Release Model**, development occurs on multiple Dev branches, again with a separate Main branch strictly for releases. This strategy has the following characteristics:

- Supports more efficiency over disparate teams
- Development occurs on separate branches
- Stable code is merged back to Main at key milestones
- Branches are synchronized periodically

Figure 3–6 shows the Multi-Team, Multi-Release Model.

Figure 3–6 Multi-Team, Multi-Release Model



Using the Multi-Team, Multi-Release Model in Oracle Business Intelligence

Using complex branching strategies in Oracle Business Intelligence requires attentive organization of repository files, as well as altering the Multiuser setting in the Administration Tool.

The following procedure provides an overview of the required steps.

To use the multi-team, multi-release model branching strategy:

1. Create a Main repository (Master Repository) and store it in the Master multiuser development directory.
   - Projects must be explicitly defined.
   - Branch developers should not have access to the Master directory.
2. Create a subset of branch repositories by extracting from Main and storing them as the Team1 and Team2 multiuser development directories. The Main and Team RPDs must be stored and secured in separate directories on the network.
3. Developers must check out, develop, merge, and publish from their respective Team RPDs. Developers A1 through A3 and B1 through B3 should manage their metadata work and merge to their Team repository.
   - Teams 1 and 2 must maintain their own repositories and periodically synchronize from Main to Team branches.
   - The Team repositories must be merged back into and published in the Main repository.

4. One specific group (for example, release management) should manage all project definitions, perform merges, publish, and synchronize the Team RPDs back to Main.

Synchronizing RPD Branches

For large development teams, it is a good practice to perform periodic branch synchronization as Main changes, in order to ease the ultimate Team publishing step.

Use the Administration Tool to synchronize repositories in a three-way merge.

To synchronize repository branches:

1. Publish all changes from your Team development branch and open the RPD in the Administration Tool. This the **current** repository.
2. Extract a fresh Branch subset from Main. This is the **modified** repository.
3. In the Administration Tool, select **File**, then select **Merge** and browse to the backup of the previous Branch subset. This is the **original** repository.
4. Resolve all issues and perform the merge.

   The RPD named in the **Save merged repository as** field becomes the new branch development RPD and is called the Original in future synchronizations.

Viewing and Deleting History for Multiuser Development

You can view and delete the development history of a multiuser development repository.

This section contains the following topics:

- Viewing Multiuser Development History
- Deleting Multiuser Development History

Viewing Multiuser Development History

You can view the development history of a multiuser development repository. In the Administration Tool, multiuser development history is only available when no repository is open and after the administrator sets up the shared network directory. This prevents the confusion that could occur if a user opened a history log that did not match an open, unrelated repository.

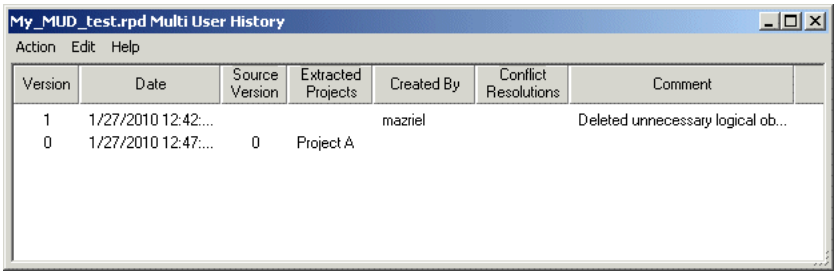
To view multiuser development history:

1. Open the Administration Tool.
2. Without opening a repository, select **File > Multiuser > History**.
3. In the Multiuser Development History dialog, select a repository.

   A list of all master repositories in the multiuser development directory appears. If the directory contains only one master repository, it is selected by default, and no list appears.
4. In the Open Offline dialog, type the password for the repository. The Multi User History dialog appears.

   Figure 3–7 shows the Multi User History dialog.

Figure 3–7 Multi User History Dialog



5. In the Multi User History dialog, right-click a row and select an option. Table 3–1 describes the options in the Multi User History dialog.

   Tip: To see details for all revisions, right-click in the background with no rows selected and select **View > Details**.

Table 3–1 Options in the Multi User History Dialog

Action	Description
View > Repository	Loads the selected master version of the repository in the Administration Tool in read-only mode.
View > Prior to Merge > Projects	Loads the selected version of a modified subset repository in the Administration Tool in read-only mode.
View > Conflict Resolution	Loads all necessary repositories of the selected version. Also shows the Merge dialog in read-only mode with all selected decisions as they were during the Merge Local Changes activity at that time. Double-clicking a row for a version with conflict resolutions has the same effect as selecting this menu item. **Note:** This menu item is only enabled for versions that had conflict resolutions.
View > Details	Displays a log with details for the selected versions, or displays details for all versions if no specific versions are selected.
View > Prior to Merge > Changes	Compares the modified subset repository of the selected version with the original subset repository and shows all changes made by the user in the selected version.
Find and Find Again	Lets you search the list.
Select All	Selects all items displayed in the dialog.
Delete	Available only to multiuser development administrators.

Deleting Multiuser Development History

Only multiuser development administrators can delete history. Administrators are defined in a special hidden option file in the multiuser development directory. See "Setting Multiuser Development Options" for more information.

An administrator can delete the entire MUD history, or the oldest 1 to *n* versions. It is not possible to delete versions in the middle of the range. For example, an administrator cannot delete version 3 if there are still versions 1 and 2. If an administrator deletes the entire MUD history, newly assigned version numbers restart at version 1.

If an administrator deletes a MUD history version from which a developer has checked out a subset, and the developer is still working on it, the developer will not be able to publish to the MUD directory. As a result, deleting all MUD history prevents any developer who has currently checked out a subset from publishing it. Because of

this, administrators should communicate with developers before the MUD history is cleared.

Setting Multiuser Development Options

You can create a multiuser development option file to specify options for multiuser development. The option file is a text file, in standard Windows INI format.

The option file has the following properties and characteristics:

- The option file must be placed in the multiuser development directory. The file has the same name as the corresponding master repository, but with an .opt extension. For example, for \\network\MUD\sales.rpd, create an option file called \\network\MUD\sales.opt.
- The file should have the **Hidden** flag turned on.
- In general, the option file should be managed only by multiuser development administrators. To ensure this, you may want to change the sharing permissions for the file.

The following example shows a multiuser development option file:

```
[Options]
BuildNumber = Yes
Enforce Build Number = 11.1.1.7.0
Enforce MUD Protocol Version Number = 1
Prevent Rpd Upgrade = Yes
Admin = admin1;admin2
Mandatory Consistency Check = Yes
Equalize During Merge = Yes
```

Options that are not explicitly set are turned off by default. To turn an option on, set its value to Yes. To turn an option off, either remove it from the option file, or set its value to No.

Table 3–2 explains the options in the multiuser development option file.

Table 3–2 Options in the Multiuser Development Option File

Option	Description
BuildNumber	When set to Yes, the build version of the Administration Tool is displayed in the MUD history.
Enforce Build Number	When specified, ensures that only users with an exact match of the given version of the Administration Tool can perform MUD operations. Select the **Help** menu, then select **About** to view the Administration Tool version. This option can be used in conjunction with Enforce MUD Protocol Version Number and Prevent Rpd Upgrade. Remove this line if you do not want to enforce Administration Tool version consistency for MUD operations.
Enforce MUD Protocol Version Number	When specified, ensures that only users with an exact match of the given MUD version can perform MUD operations. Select the **Help** menu, then select **About** to view the MUD version. This option can be used in conjunction with Enforce Build Number and Prevent Rpd Upgrade. Remove this line if you do not want to enforce MUD version consistency for MUD operations.

Table 3–2 (Cont.) Options in the Multiuser Development Option File

Option	Description
Prevent Rpd Upgrade	When specified, ensures that only users with an exact match of the given repository version can perform MUD operations. Select the **Help** menu, then select **About** to view the repository version. This option can be used in conjunction with Enforce Build Number and Enforce MUD Protocol Version Number. Remove this line if you do not want to enforce repository version consistency for MUD operations.
Admin	Lists multiuser development administrators. Administrators must be defined in the option file before they can delete MUD history. Administrators are defined by their computer/network login names. When multiple administrators exist, separate administrator names by semicolons. For example: Admin=jsmith;mramirez;plafleur
Mandatory Consistency Check	When set to Yes, the publish step performs a consistency check. Publishing cannot proceed unless there are no errors in the given repository.
Equalize During Merge	When set to Yes, the multiuser development merge process performs mandatory equalization during MUD merges. Note that setting this option to Yes affects the performance of the merge process.

4

Using a Source Control Management System for Repository Development

This chapter describes how to integrate the Oracle BI Administration Tool with third-party source control management systems for Oracle BI repository development.

The Administration Tool achieves this integration through the ability to save repository metadata as a set of XML documents in MDS XML format rather than as a single binary repository file (RPD). Using this integration, you can configure the Administration Tool to work with your own source control management system and save your repository output as MDS XML.

This chapter contains the following topics:

About Using a Source Control Management System with the Administration Tool

You can choose to integrate the Administration Tool with a third-party source control management system, such as Subversion, Rational ClearCase, or Git, during your repository development process.

This feature is centered around the following integration points:

- **Converting your binary RPD file to a set of MDS XML documents.** Rather than using a single large binary repository file, you can save your repository in MDS XML format. In this format, each repository object, such as a connection pool, physical table, or business model, is represented in its own XML file. The set of XML files that make up your repository can then be managed in your source control management system.
- **Setting up a Source Control Management (SCM) configuration file.** You can use the SCM Configuration Editor in the Administration Tool to specify commands specific to your SCM system, such as Add File, Delete, and Check Out, as well as environment variables required by your SCM system.
- **Designating your repository as under source control.** The first time you open your MDS XML repository in the Administration Tool, you are prompted to specify whether the repository is a standalone MDS XML repository, or whether it

is under source control. Choose **Use Source Control** to enable SCM integration for this repository in the Administration Tool.

About MDS XML

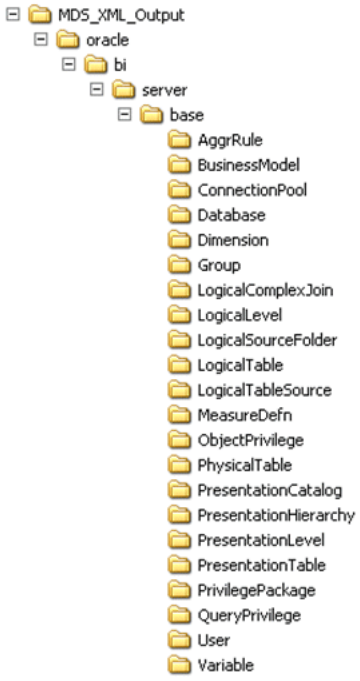
MDS XML format is typically used for repositories under source control. MDS XML represents the Oracle BI repository across a set of XML files, rather than in a single file.

MDS XML is not the same XML format that was used in previous releases to represent the Oracle BI repository in XML format. The previous Oracle BI Server XML schema, based on the xudml1.xsd XML schema file in *ORACLE_HOME*/bifoundation/server/bin, represents the Oracle BI repository in one large XML file.

For example, each repository connection pool is stored in its own file, with an XML representation like the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ConnectionPool mdsid="m80ca62c5-0bd5-0000-714b-e31d00000000"
name="SampleApp_Lite_Xml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.oracle.com/obis/repository"
password="94F9321C85340FC48E4D9093AA941FF28844074B88D5AA6364E4815DEED7F9B
8792EF452219C2155DB68F61EE1555B4FA886F77E060E2E17F45AD8D18CAB2E4D3EFA15B75E
30D8B4BFA8C7B2D70552BD" timeout="4294967295" maxConnDiff="10" maxConn="10"
dataSource="VALUEOF(BI_EE_HOME)/sample/SampleAppFiles/Data" type="Default"
reqQualifedTableName="false" isSharedLogin="false"
isConcurrentQueriesInConnection="false" isCloseAfterEveryRequest="true"
xmlRefreshInterval="2147483647" outputType="xml" ignoreFirstLine="false"
bulkInsertBufferSize="0" transactionBoundary="0" xmlaUseSession="false"
multiThreaded="false" supportParams="false" isSiebelJDBSecured="false"
databaseRef="/oracle/bi/server/base/Database/Sample App Lite Data_80ca62c4
-0bcf-0000-714b-e31d00000000.xml#m80ca62c4-0bcf-0000-714b-e31d00000000"
>
<Description>
<![CDATA[
SampleAppLite connection pool to XML datasource. This connection pool points the
database to the location where physical XML files are stored. The location uses
the value of an RPD variable : BI_EE_HOME.
This variable needs to be correctly set in order for the server to connect to the
files.
]]>
</Description>
</ConnectionPool>
```

The SampleAppLite repository will generate MDS XML files in a structure like the following:



Note that there is not a one-to-one relationship between repository objects in the Administration Tool and the set of files produced as XML output. For example, physical columns appear as independent objects in the Administration Tool, but in MDS XML they are considered part of the Physical Table object.

See "About the Oracle BI Server MDS XML API" in *Oracle Fusion Middleware XML Schema Reference for Oracle Business Intelligence Enterprise Edition* for full information about the MDS XML schema representation of repository objects.

Setting Up Your System for Repository Development Under Source Control Management

To set up your system for repository development under source control management, you must set up an SCM configuration file with commands specific to your SCM system, and generate an MDS XML repository and check it into your SCM system.

This section contains the following topics:

- Creating an SCM Configuration File
- Creating an MDS XML Repository and Checking In Files to the SCM System

Creating an SCM Configuration File

To integrate the Administration Tool with your source control management system, you must create an XML configuration file based on your specific SCM system. The configuration file contains the SCM system commands for adding, deleting, checking out, and renaming files. The Administration Tool will issue these commands to the SCM system when repository objects are created or updated, resulting in corresponding new or changed MDS XML files.

> **Note:** The Administration Tool does not commit the changes to the SCM system. The repository developer must always check the files into the SCM system directly. This way, the repository developer can view any conflicts or make merge decisions in the SCM environment rather than the Administration Tool environment.

To create a configuration file for your SCM system:

1. Open the Administration Tool and select **Tools**, then select **Options**.
2. Select the Source Control tab.
3. Click **New** to create a new configuration file. The Specify new configuration file window is displayed.

   Note: This procedure assumes that you do not have an open MDS XML repository in the Administration Tool. If you create or edit an SCM configuration file while an MDS XML repository is open, you must ensure that **Use Source Control** is selected to enable the **New** or **Edit** buttons.
4. Provide a file name and click **Save**. The file must have the XML file extension.

   The default location for SCM configuration files is *ORACLE_INSTANCE*/config/OracleBIServerComponent/coreapplication_obis*n*. Although templates are also available in this location, do not select a template file during this step. Instead, you can load a template in the next step.

   > **Note:** The SCM configuration template files are called scm-conf-ade.template.xml and scm-conf-svn.template.xml. In addition to being available in the *ORACLE_INSTANCE* location indicated, they are also available on the Oracle Technology Network (OTN) at:
   >
   > http://www.oracle.com/technetwork/middleware/bi-foundation/downloads/obieescmconfigfiles-1568980.zip

5. To load a template configuration file, click **Load** in the SCM Configuration Editor. Then, select a template file (for example, scm-conf-svn.template) and click **Open**.

   Unless it is your intention to modify the configuration file template itself, ensure that **Edit in Configuration Editor** is *not* selected. If you select this option, the file name displayed in the **Configuration File** field in the SCM Configuration Editor changes from the file name you provided in the proceeding step to the template file name, and changes are saved by default to the template file.
6. In the SCM Configuration Editor, provide an optional description, then enter or edit commands for your system in the Commands subtab. For longer commands, click the ellipsis button to enter commands in the Command Editor window.

   Use the ${file}, ${filelist}, ${from}, and ${to} tokens to define the commands. You can also use the **List File** option in conjunction with the ${filelist} command to set the behavior. The tokens can be used as follows:

   - **${file}** specifies that a command must be run sequentially, one file at a time. ${file} is required for the Add Folder and Add File commands.
   - The behavior of **${filelist}** varies, depending on whether **List File** is selected:
     - **${filelist}** without **List File** selected causes the Administration Tool to group as many files as possible for the given command (such as

Pre-Delete, Delete, or Check-out), staying under the 32k character limit for launching a process. Execution is repeated until all files have been processed.

– **${filelist}** with **List File** selected instructs the Administration Tool to create a temporary file that holds the file list. The file list format is one file name for each line, which is a typical format among SCM vendors. List File is valid for Pre-Delete, Delete or Check-out commands. It results in much faster operations and should always be selected for SCM systems that support it.

You can use either ${file} or ${filelist} for Pre-Delete, Delete, and Checkout. **List File** only works in conjunction with ${filelist}.

- **${from}** and **${to}** are used to specify the original file name and new file name in Rename commands.

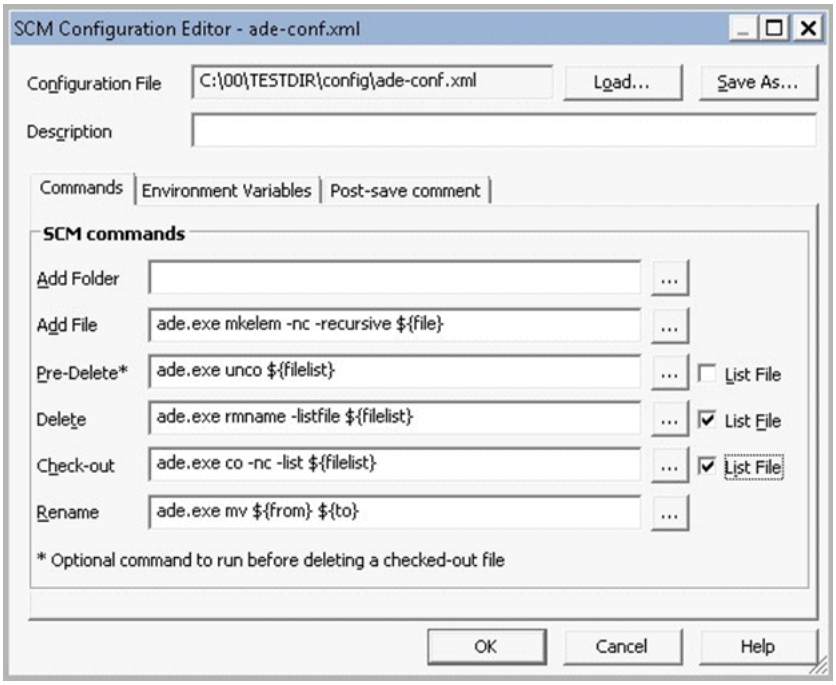
  Not all SCM systems support file rename operations natively. If this is the case, leave the **Rename** field blank rather than attempting to construct a rename operation by concatenating different commands. The Administration Tool will do this for you with greater efficiency.

Note: Some SCM systems do not include commands for working with folders. If this is the case, leave **Add Folder** blank. The Administration Tool always creates folders for you when needed.

Even if your SCM system does include folder management commands, the Administration Tool does not remove folders. You must remove folders directly in the SCM system if necessary.

Figure 4–1 shows the Commands tab of the SCM Configuration Editor.

Figure 4–1 Commands Tab of SCM Configuration Editor



7. Select the Environment Variables subtab, and then specify environment variables required by your SCM system.

   You can paste environment variables directly from your operating system variable list, paste environment variables from the clipboard, or manually add environment variables. Table 4–1 describes the options available for managing environment variables in the Environment Variables subtab.

Table 4–1 Options for Managing Environment Variables in the SCM Configuration Editor

Option	Description
Paste environment variables 	Enables you to paste environment variables directly from your operating system. Click this option to open a window where you can enter filter criteria, then click **OK**. Enter * in the filter window to paste all environment variables.
Paste from clipboard 	Enables you to paste text from the clipboard. To use this option, copy text in the following format: variable_name1=variable_value1 variable_name2=variable_value2 Each environment variable must be on its own line. **Note:** The standard Windows command `set` provides output in this format for environment variables.
Add 	Adds a row to the table so that you can manually enter environment variables. Provide the variable name in the **Variable** column, and its definition in the **Value** column. Leading and trailing white space is trimmed. You can use **%VAR%** in the variable definition to reference a previously defined variable.
Delete 	Deletes the given row in the environment variables table.

   Note that you should not store security-sensitive environment variables in the configuration file. If security-sensitive variables are required by your SCM system, to avoid the security risk, you can launch the Administration Tool from a DOS window with any security-sensitive variables already set.

8. Click **Test** in the Environment Variables subtab to open the Test SCM Configuration window. Then, enter a command and click **Execute** to test a particular command. If the environment is correct, the correct output should appear after executing the command.

9. Select the Post-save comment subtab to enter text that will appear after changes are saved in the Administration Tool. This capability is a way to remind developers to check files directly into their SCM system after saving. For example, a post-save comment might be:

   ```
   Files have been synchronized to source control. Remember to check in changes
   after testing.
   ```

10. Click **OK** to save the configuration file, or click **Save As** to save a copy if you loaded and modified a template configuration file.

Creating an MDS XML Repository and Checking In Files to the SCM System

To integrate with an SCM system, you must convert your Oracle BI repository to MDS XML format.

Use one of the following options to create an MDS XML repository and check it into your source control system:

- Saving an Existing Repository File in MDS XML Format
- Creating a New Repository in MDS XML Format
- Linking to Source Control Files to Convert Your Repository (Small Repositories Only)

Saving an Existing Repository File in MDS XML Format

If you have an existing repository file, follow these steps to convert it to MDS XML. The steps described in this section are the recommended method for initial import.

To save an existing repository file in MDS XML format:

1. Open your existing repository file (RPD) in the Administration Tool in offline mode.
2. Select **File**, then select **Save As**, then select **MDS XML Documents**.
3. Select a root location for your MDS XML repository files, and then click **OK**.
4. Perform the necessary steps in your source control management system to add and check in the files.

   Use the specialized commands for bulk file import, available for most SCM systems. These commands are optimized to deliver entire trees of files to source control in a very efficient way. For example, in Subversion, use the following command:

   ```
   svn import module_name -m "Initial import"
   ```

> **Note:** You can also use the `biserverxmlgen` utility with the `-M` and `-D` options to generate MDS XML from an existing RPD. See "Generating MDS XML from an Existing RPD Using a Command-Line Utility" in *Oracle Fusion Middleware XML Schema Reference for Oracle Business Intelligence Enterprise Edition* for more information.

Creating a New Repository in MDS XML Format

Use the following steps to create a new repository in MDS XML format.

To create a new repository in MDS XML format:

1. Open the Administration Tool and select **File**, then select **New Repository** to open the Create New Repository Wizard.
2. Select the **MDS XML Documents** option in the wizard. Complete the other wizard steps.
3. Perform the necessary steps in your source control management system to add and check in the files. For large repositories, use the specialized commands for bulk file import for your SCM system.

> **Note:** Do not create a new MDS XML-format repository, add objects, and then select **Link to Source Control**. This method will not work, and no SCM commands will be generated.

Linking to Source Control Files to Convert Your Repository (Small Repositories Only)

For very small repositories, you can use the Link to Source Control files method to convert a binary RPD file to MDS XML format.

To link to source control files to convert your repository:

1. Ensure that you have an SCM configuration file defined. See "Creating an SCM Configuration File" for more information.
2. Create an empty root folder for the MDS XML repository.
3. Open your existing RPD file in the Administration Tool in offline mode.
4. Select **File**, then select **Source Control**, then select **Link to Source Control Files**.
5. Select the root folder you created, and the appropriate SCM configuration file.

   Note: If you need to change the configuration file later, you can go **Tools > Options > Source Control** and click **Edit** to change the configuration file.
6. Click **Save**. An MDS XML repository is created, and the necessary add file operations are performed in your source control system.
7. Commit the changes in your SCM system.

Note: Using the Link to Source Control Files method to initially import your repository is only recommended for very small repositories. This method is too slow for large repositories (tens of thousands of files) because the Administration Tool imports the files one at a time using the standard "add file" command, rather than using specialized commands for bulk file import.

Note also that the repeated invocation of the "add file" command might increase the chances of transient errors. If these occur, you might need to restart the process a few times before all files are successfully imported to source control.

Using Source Control Management in Day to Day Repository Development

This section describes typical scenarios that occur during day to day repository development.

This section contains the following topics:

- Updating, Saving, and Checking In Changes for Repositories Under Source Control
- Handling Errors
- Testing Repositories Under Source Control
- Viewing the Source Control Log

Updating, Saving, and Checking In Changes for Repositories Under Source Control

After your MDS XML repository is set up under source control, follow these steps to update, save, and check in changes to your repository.

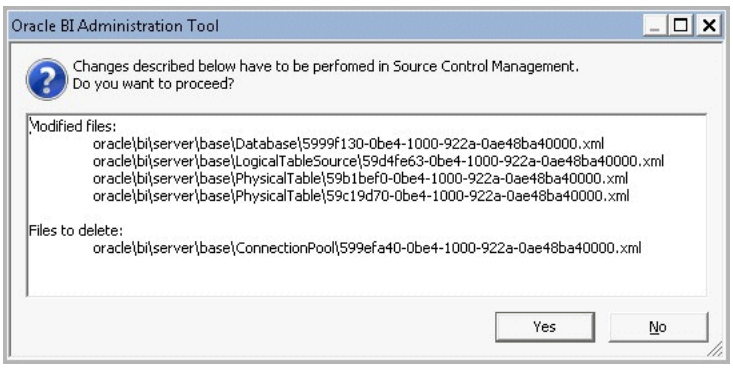
To update, save, and check in changes to your repository:

1. Ensure that you have a local copy of your working MDS XML repository files that are under source control by issuing the appropriate commands in your SCM system. For example, for Subversion, you can issue the command `svn info` as shown in the following example text:

```
C:\myProj\repos>svn info
Path: .
Working Copy Root Path: C:\myProj\repos
URL: file:///C:/SVN/myProj/trunk/sample1
Repository Root: file:///C:/SVN/myProj
Repository UUID: 6b995c92-3ec0-fa4b-9d58-c98e54f41792
Revision: 3
Node Kind: directory
Schedule: normal
Last Changed Author: joe_user
Last Changed Rev: 2
Last Changed Date: 2011-11-19 15:20:42 -0600 (Sat, 19 Nov 2011)
```

2. Open your MDS XML repository in the Administration Tool in offline mode. To do this, select **File**, then select **Open**, then select **MDS XML**.
3. Select the root folder location for your MDS XML files and click **OK**.
4. If this is the first time you have opened this MDS XML repository in the Administration Tool, you are prompted to specify whether this repository is a standalone MDS XML repository, or whether it is under source control. Select **Use Source Control** and click **OK**.

   This choice is saved for this repository. To view the status of this repository at any time, select **Tools**, then select **Options**, then select the Source Control tab.
5. After you make changes to your repository, select **File**, then select **Save**, or click **Save** on the toolbar. The Administration Tool displays a list of changes. For example:



6. Click **Yes**. The Administration Tool runs the necessary commands in the SCM system.

   After you accept the changes, you cannot cancel. Canceling in the middle would leave an inconsistent repository. You must wait for the SCM commands to be executed.

   Note also that when the Administration Tool issues the SCM commands, they may be rearranged into the most optimal order.
7. Check in the changes directly in your SCM system.

Handling Errors

Sometimes errors, such as an expired label or network problem, occur when the Administration Tool delivers changes to the SCM system.

If errors occur, perform the following steps.

To handle errors:

1. In the Administration Tool, select **File**, then select **Save As** to save the repository to a temporary location in RPD format or MDS XML format. Close the Administration Tool.

   Note: Saving to a binary RPD is the simplest option for transient problems like network errors, where you just need to try again later. Saving as MDS XML is required when some sort of work is required to fix the problem, such as merging conflicting changes.

2. Take action to resolve the issue. For example, refresh an expired label or test and view a failed network connection.

   In the case of an expired label, you also need to merge the contents of the refreshed label with the temporary saved MDS XML repository. Use a third-party merge tool to do this.

   For detailed information about the MDS XML representation of repository objects so that you can successfully make merge decisions, see *Oracle Fusion Middleware XML Schema Reference for Oracle Business Intelligence Enterprise Edition*.

3. Open the saved repository in the Administration Tool.
4. Select **File**, then select **Source Control**, then select **Link to Source Control**.
5. Click **Save** to save changes from the saved repository into the MDS XML files under source control.

Steps 4 and 5 of this procedure cause the Administration Tool to keep memory objects loaded from the saved RPD file or MDS XML files, but to then consider them to belong to the source control MDS XML repository instead. When you click **Save**, the Administration Tool saves the memory objects to the source control repository.

Testing Repositories Under Source Control

During the course of repository development, you will need to perform testing in online mode to validate your repository. You can only load an Oracle BI repository in RPD format into the Oracle BI Server to make it available for queries. Because of this, you must save your development MDS XML repository in RPD format from time to time when you want to perform online testing. To do this, open your MDS XML repository in offline mode and select **Save As**, then select **Repository**.

See "Making the Repository Available for Queries" for more information about uploading repositories.

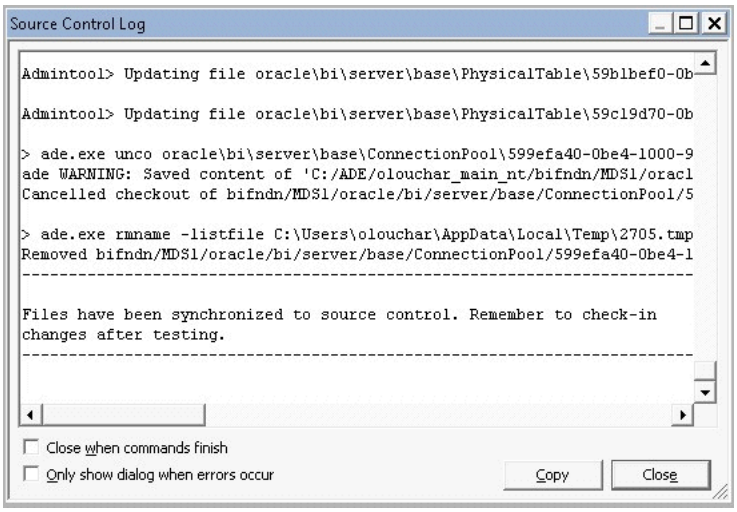
Viewing the Source Control Log

The Source Control Log window shows the commands that the Administration Tool issues to your SCM system. It also shows any post-save text you specified in the Post-save comment tab of the SCM Configuration Editor.

By default, the Source Control Log window appears when SCM commands are being executed. Alternatively, you can select **File**, then select **Source Control**, then select **View Logs** to see the Source Control Log window.

Figure 4–2 shows the Source Control Log window.

Figure 4–2 Source Control Log Window



You can choose the following options for this dialog:

- **Close when commands finish:** Causes the log window to close automatically when commands are complete, unless errors occur.
- **Only show dialog when errors occur:** Hides the window during SCM command execution unless errors occur. By default, the Source Control Log appears automatically when SCM commands are being executed unless this option is selected.

The text displayed in the Source Control Log is persistent until you close the repository. This means that all SCM command output is available for view, regardless of whether the dialog is open during individual operations.

The Source Control Log does have a 32K character limit. When the window buffer becomes full, then the oldest commands are removed from the Source Control Log display to make room for the latest command output. To see the full output, go to the Administration Tool log at:

ORACLE_INSTANCE/diagnostics/logs/OracleBIServerComponent/coreapplication_obisn/*user_name*_NQSAdminTool.log

> **Note:** While SCM commands are being executed, the **Close** button is disabled until the SCM commands have finished or have stopped with an error (unless **Only show dialog when errors occur** has been selected).

Using Source Control Management with MUD

You can use source control management with your multiuser development environment.

This section provides a general outline of how you might use both processes together.

For example, if you have an existing repository under multiuser development and you want to begin using source control management, you might follow the steps described in the following subsections:

- Putting the MUD Master Repository and MUD Log File Under Source Control
- Checking In New Versions of the MUD Master and MUD Log File to Source Control

Putting the MUD Master Repository and MUD Log File Under Source Control

Use the following steps to put the MUD master repository and MUD log file under source control.

To put the MUD master repository and MUD log file under source control:

1. Convert your master MUD RPD to a set of MDS XML files on the file system. See "Saving an Existing Repository File in MDS XML Format" for information on how to do this.
2. Run the `mhlconverter` command-line utility to convert your MUD log file (*.mhl) to an XML file. At the command prompt, type `mhlconverter` with the input MHL file name and path, and the output XML file name and path. For example:

   ```
   mhlconverter -I C:\MUD\mud_repository.mhl -O C:\MUD\mud_repository.xml
   ```

3. Check the MDS XML files and XML-format MUD log file into your SCM system.

Checking In New Versions of the MUD Master and MUD Log File to Source Control

After creating and checking in the initial version of the master MUD repository, you need to check in updated versions of the MUD master repository on an ongoing basis.

This section describes two different ways to perform this task.

To check in new versions of the MUD master and MUD log file to source control:

- Manually Checking In the Updated MUD Master Repository and Log File
- Using a Script to Check In the Updated MUD Master Repository and Log File

Manually Checking In the Updated MUD Master Repository and Log File

Follow these steps to manually check in changes to the master RPD and log file that have occurred as part of the multiuser development process.

To manually check in the updated MUD master repository and log file:

1. Open the latest copy of the master RPD in the Administration Tool.
2. Create or select the appropriate SCM configuration file. See "Creating an SCM Configuration File" for more information.
3. Select **File**, then select **Source Control**, then select **Link to Source Control**. Select the directory that contains the MDS XML version of the master MUD repository.

   Using **Link to Source Control** is not recommended for large repositories and might cause time-outs. Consider using the automated check-in method described in "Using a Script to Check In the Updated MUD Master Repository and Log File" if you have a large repository.
4. Click **Save** to save changes from the master MUD repository into the MDS XML files under source control. The Administration Tool determines which files to add, check out, modify, and delete and issues the commands to your SCM system.

5. Close the Administration Tool.
6. Follow these steps to update the MUD log file:
   a. In your SCM system, check out the XML-format MUD log file.
   b. Use the `mhlconverter` utility to overwrite the XML-format MUD log file with the latest changes from the .mhl version.
   c. Check in the latest XML-format MUD log file to your SCM system.
7. Check all changes into your SCM system.

It is recommended that you perform the steps in this section regularly to avoid having too many changes in a single transaction.

Using a Script to Check In the Updated MUD Master Repository and Log File

As an alternative to manually checking in changes, you can create a script to perform the check-in tasks and then schedule it to run at regular intervals.

To use a script to check in the updated MUD master repository and log file:

1. Identify the latest copy of the master RPD that you want to check into your SCM system.
2. Identify the last version of the master RPD that was checked into the SCM system. You can review the latest XML-format MUD log file under source control to determine this version.

> **Note:** If you do not have the last checked-in version of the master repository in RPD format, you can use the `biserverxmlexec` utility with the -D option to read the latest MDS XML files checked into source control and re-create an RPD version.

3. Use the `comparerpd` utility with the `-M` option to compare the latest copy of the master RPD (the modified version) with the version that was last checked in (the original version). An MDS XML format diff is generated. See "Comparing Repositories Using comparerpd" for more information.
4. Create a script that does the following:
   a. Reads the MDS XML diff directory to identify which files are present.
   b. Issues commands in source control to check out the identified files or add new files.
   c. Copies the latest version of the files from the MDS XML diff directory to the source control directory.
   d. Reads the oracle\bi\server\base\DeletedFiles.txt file inside the MDS XML diff directory to determine which files to delete.
   e. Issues commands in source control to delete the appropriate files.
   f. Checks out the MDS XML-format MUD log file, runs the `mhlconverter` utility to convert the latest MHL-format log file to XML format, overwrites the existing MDS XML-format MUD log file with the new one, and checks it in.
   g. Performs all necessary check-in steps in the SCM system.

5

Importing Metadata and Working with Data Sources

This chapter describes how to create a new Oracle BI repository, set up back-end data sources, and import metadata using the Import Metadata Wizard in the Oracle BI Administration Tool. It also describes how to use a standby database with Oracle Business Intelligence.

This chapter contains the following topics:

About Importing Metadata and Working with Data Sources

After you create an Oracle BI repository file, you can import metadata from your data sources into the Physical layer of your repository. The Physical layer of the Administration Tool defines the data sources to which the Oracle BI Server submits queries, and the relationships between physical databases and other data sources that are used to process multiple data source queries.

Metadata imports to an Oracle BI repository must occur through an ODBC or native database connection to the underlying data source. Metadata can also be imported from software such as Microsoft Excel through an ODBC connection.

Importing metadata directly from each data source saves you time and effort by importing the structure for the Physical layer. Data from these sources can be displayed on Oracle BI Interactive Dashboards and other clients. You can only import metadata from supported data sources.

After you import metadata, properties in the associated database object and connection pool are typically set automatically. However, you may want to adjust database or connection pool settings. See Chapter 7, "Setting Up Database Objects and Connection Pools" for more information.

Although you can create the Physical layer manually rather than importing metadata, it is a labor-intensive and error-prone activity. It is strongly recommended that you import metadata.

Creating a New Oracle BI Repository

You can use the Create New Repository Wizard in the Administration Tool to create a new Oracle BI repository in either binary (RPD) or MDS XML format.

You do not need to create a new repository if you already have an existing repository. For an existing repository, you can use the existing data source settings in that file as a template to connect to different data sources. To do this, use the existing data source settings and just change the database type and connection pool information. See "Setting Up Database Objects" and "Creating or Changing Connection Pools" for details.

To create a new Oracle BI repository:

1. In the Administration Tool, select **File**, then select **New Repository**. The Create New Repository Wizard appears.

   If an existing repository is open, you are prompted to save your changes, and the existing repository is closed.

2. Select **Binary** if you want to create a repository in RPD format. To create a repository in MDS XML format, select **MDS XML Documents**.

3. For binary repositories, type a name for the repository. Keep the name to 156 characters or less to avoid problems with the metadata dictionary URL. An RPD file extension is automatically added if you do not explicitly specify it.

4. For **Location**, follow the steps appropriate for your repository type:

   - For binary repositories, select a location for the RPD file. By default, new binary repositories are stored in the repository subdirectory, located at *ORACLE_INSTANCE*\bi\bifoundation\server.
   - For MDS XML format repositories, select a root folder location for the set of MDS XML files.

5. If you want to import metadata into the repository now, select **Yes** (the default) for **Import Metadata**. If you do not want to import metadata, select **No**.

6. Enter and confirm the password you want to use for this repository. The repository password must be eight characters, containing one or more numerals.

   You enter the repository password when you open the repository in online or offline mode. It is used to encrypt the repository contents.

7. If you selected **Yes** for **Import Metadata**, click **Next**.

   Refer to the following sections for information about the Import screens, according to your data source type:

   - Importing Metadata from Relational Data Sources
   - Importing Metadata from Multidimensional Data Sources
   - About Importing Metadata from XML Data Sources
   - Working with ADF Data Sources

Note that you many need to set up your data sources before you import information into the repository. See "Performing Data Source Preconfiguration Tasks" for more information.

8. If you selected **No** for **Import Metadata**, click **Finish** to create an empty repository.

Performing Data Source Preconfiguration Tasks

In some cases, you must perform configuration steps so that Oracle Business Intelligence can access the data sources. These configuration steps are sometimes required before you can import physical objects from your data sources into your repository file, or set up connection pools to your data sources.

For many data sources, you need to install client components. Client components are typically installed on the computer hosting the Oracle BI Server for query access, and on the computer hosting the Administration Tool (if different) for offline operations such as import. In some cases, client components must be installed on the computer where the JavaHost process is located.

Note: See also the following related topics:

- If the Oracle BI Server is running on a non-Windows platform, see Chapter 16, "Setting Up Data Sources on Linux and UNIX" for additional instructions.
- See Chapter 6, "Working with ADF Data Sources" for information about setting up ADF data sources.
- See "System Requirements and Certification" for information about the data source versions supported by Oracle Business Intelligence.

This section contains the following topics:

- Setting Up ODBC Data Source Names (DSNs)
- Setting Up Oracle Database Data Sources
- About Setting Up Oracle OLAP Data Sources
- About Setting Up JDBC and JNDI Data Sources
- About Setting Up Oracle TimesTen In-Memory Database Data Sources
- About Setting Up Essbase Data Sources
- About Setting up Cloudera Impala Data Sources
- About Setting Up Apache Hive Data Sources
- About Setting Up Hyperion Financial Management Data Sources
- Setting Up SAP/BW Data Sources
- Setting Up Oracle RPAS Data Sources
- Setting Up Teradata Data Sources
- Enabling NUMERIC Data Type Support for Oracle Database and TimesTen

Setting Up ODBC Data Source Names (DSNs)

Before you can import from a data source through an ODBC connection, or set up a connection pool to an ODBC data source, you must first create an ODBC Data Source Name (DSN) for that data source on the client computer. You reference this DSN in the Import Metadata Wizard when you import metadata from the data source.

You can only use ODBC DSNs for import on Windows systems.

To set up an ODBC DSN on Windows:

1. In Windows, locate and open the ODBC Data Source Administrator. The ODBC Data Source Administrator dialog appears.
2. In the ODBC Data Source Administrator dialog, click the System DSN tab, and then click **Add**.
3. From the Create New Data Source dialog, select the driver appropriate for your data source, and then click **Finish**.

   The remaining configuration steps are specific to the data source you want to configure. Refer to the documentation for your data source for more information.

ODBC DSNs on Windows systems are used for both initial import, and for access to the data source during query execution. On UNIX systems, ODBC DSNs are only used for data access. For information about setting up ODBC data sources on UNIX, see Chapter 16, "Setting Up Data Sources on Linux and UNIX."

See also "Setting Up Teradata Data Sources" for additional information specific to Teradata.

Setting Up Oracle Database Data Sources

When you import metadata from an Oracle Database data source or set up a connection pool, you can include the entire connect string for **Data Source Name**, or you can use the net service name defined in the tnsnames.ora file.

If you choose to enter only the net service name, you must set up a tnsnames.ora file in the following location within the Oracle Business Intelligence environment, so that the Oracle BI Server can locate the entry:

BI_DOMAIN\config\fmwconfig\bienv\core

You should always use OCI when importing metadata from or connecting to an Oracle Database. Before you can import schemas or set up a connection pool, you must add a TNS names entry to your tnsnames.ora file. See the Oracle Database documentation for more information.

This section contains the following topics:

See also "Enabling NUMERIC Data Type Support for Oracle Database and TimesTen" for related information.

Oracle 12*c* Database In-Memory Data Sources

For all Oracle 12*c* Database In-Memory data sources, the Oracle BI Server creates tables in memory.

Oracle 12*c* Database In-Memory is a high-performance, in-memory data manager. It uses In-Memory Column Store, which stores copies of tables and partitions in a special columnar format that exists in memory and provides for rapid scans. See the 12*c* Release 1 *Oracle Database Concepts Guide* and *Oracle Database Administrator's Guide* for more information.

Oracle 12*c* on Exadata Data Sources

For Oracle 12*c* Database on Exadata and Oracle 12*c* Database In-Memory on Exadata data sources, the Oracle BI Server creates tables in memory. Oracle BI Server uses Exadata Hybrid Columnar Compression (EHCC) by default.

Oracle Exadata Database Machine is the optimal platform for running Oracle Database. Both Oracle 12*c* Database and Oracle 12*c* Database In-Memory run on the Oracle Exadata Database Machine. See the documentation included with the Exadata Database Machine for more information.

Advanced Oracle Database Features Supported by Oracle BI Server

To take advantage of native Oracle Database functionality and significantly improve query time, the Oracle BI Server supports the compression, Exadata Hybrid Columnar Compression, and In-Memory features.

When you import metadata or specify a database type in the General tab of the Database dialog, the set of features for that database object is automatically populated with default values appropriate for the database type. These are the SQL features that the Oracle BI Server uses with this data source. When a feature is marked as supported (checked) in the Features tab of the Database dialog, the Oracle BI Server typically pushes the function or calculation to the data source for improved performance. When a function or feature is not supported in the data source, the calculation or processing is performed in the Oracle BI Server.

Note the following information about Oracle Database features supported by Oracle BI Server.

- **Compression**– Compression reduces the size of the database. Because compressed data is stored in fewer pages, queries need to read fewer pages from the disk, thereby improving the performance of I/O intensive workloads. Compression is used by default for Oracle 11*g* Database, Oracle 12*c* Database, and Oracle 12*c* Database In-Memory. If you create aggregates on your Oracle 11*g* and Oracle 12*c* databases, then compression is applied to the aggregate tables by default.

  When you create a database object for any of the above mentioned Oracle databases, the COMPRESSION_SUPPORTED feature is automatically applied to the object.

- **Exadata Hybrid Columnar Compression (EHCC)** – Oracle's EHCC is optimized to use both database and storage capabilities on Exadata and enables the highest level of data compression to provide significant performance improvements. By default, Oracle 11*g* Database on Exadata, Oracle 12*c* Database on Exadata, and Oracle 12*c* Database In-Memory on Exadata use this type of compression.

  When you create a database object for any of the above mentioned Oracle databases, the EHCC_SUPPORTED feature is automatically applied to the object.

Note that by default, compression is turned off for objects in the above mentioned Oracle databases. To turn compression on for an object, set the object's PERF_PREFER_COMPRESSION flag to on.

- **In-Memory** – In memory retrieval eliminates seek time when querying the data, which provides faster and more predictable performance than disk. The in memory feature creates tables in memory for Oracle 12*c* Database In-Memory and Oracle 12*c* Database In-Memory on Exadata. If you create aggregates on these databases, then the aggregates are created in memory.

  When you create a database object for any of the above mentioned Oracle databases, the INMEMORY_SUPPORTED feature is automatically applied to the object.

Oracle Database Fast Application Notification and Fast Connection Failover

If Fast Application Notification (FAN) events and Fast Connection Failover (FCF) are enabled on the Oracle Database, then the Oracle Call Interface (OCI) loaded by the BI Server will utilize the FAN events and enable FCF for the Oracle Database data sources.

This functionality will run in the background. When an Oracle Business Intelligence query initiated by an analysis user fails due to the Oracle database being unavailable, the query will fail quickly and the user can then retry the query rather than waiting for the database request to time out.

Additional Oracle Database Configuration for Client Installations

You must install the Oracle Database Client on the computer where you performed the client installation. You must perform the Oracle Database Client install before you can import from Oracle Database sources. Use either the Administrator or Runtime client install option.

After installing the Oracle Database Client, create an environment variable called ORACLE_HOME and set it to the Oracle home for the Oracle Database Client. Then, create an environment variable called TNS_ADMIN and set it to the location of the tnsnames.ora file, which is *BI_DOMAIN*\config\fmwconfig\bienv\core.

Configuring Oracle BI Server When Using a Firewall

The presence of a firewall between the Oracle BI Server and the Oracle Database can result in very long query times. For example, if using a simple nqcmd query takes two to three minutes to return results, or if Answers does not respond after executing or validating a SQL statement initiated in Presentation Services.

To improve query time, go to the sqlnet.ora file in *BI_DOMAIN*\config\fmwconfig\bienv\core and add the `BREAK_POLL_SKIP` and `DISABLE_OOB` parameters as follows:

```
BREAK_POLL_SKIP=10000
DISABLE_OOB=ON
```

This configuration only needs to be performed on the Oracle BI Server. This configuration is not needed on the Oracle Database or on user client desktops.

Oracle Database Connection Errors in Windows 7 64-bit Environments

If you are running Oracle BI EE on a Windows 7 64-bit computer, you must ensure that the default authentication service is not set to use Windows domain credentials. Otherwise, you might receive a connection error when importing from an Oracle

Database because the Administration Tool will attempt to log in using your Windows domain credentials.

Check the sqlnet.ora file in *BI_DOMAIN*\config\fmwconfig\bienv\core to ensure that the `AUTHENTICATION_SERVICES` parameter appears as follows:

```
SQLNET.AUTHENTICATION_SERVICES= (NONE)
```

About Setting Up Oracle OLAP Data Sources

Before you import from an Oracle OLAP data source, ensure that the data source is a standard form Analytic Workspace. In addition, the biadminservlet Java process must be running to import from Oracle OLAP data sources, for both offline and online imports. You can use Fusion Middleware Control to check the status of the biadminservlet Java process.

Additional Oracle OLAP Configuration for Client Installations

You must install the Oracle Database Client on the computer where you performed the client installation before you can import from Oracle OLAP sources. Use either the Administrator or Runtime client install option.

After installing the Oracle Database Client, create an environment variable called ORACLE_HOME and set it to the Oracle home for the Oracle Database Client. Then, create an environment variable called TNS_ADMIN and set it to the location of the tnsnames.ora file, which is *BI_DOMAIN*\config\fmwconfig\bienv\core.

About Setting Up JDBC and JNDI Data Sources

Before you can include JDBC and JNDI data sources in the repository, you must perform the required set up tasks in this section.

Configuring Java Data Sources In Weblogic Server

If you will use the JDBC (JNDI) connection type, then the remote Java data sources must connect to Weblogic Server. This is done by configuring JNDI in Weblogic Server. This configuration step is not required if you will use JDBC (Direct Driver).

For information about how to perform this configuration, see "Using WebLogic JNDI to Connect a Java Client to a Single Server" in the BEA Weblogic Server and Weblogic Express 8.1 Documentation library.

Loading Java Data Sources

To make Java data sources available for import into the repository, you must first connect to the Java Datasource server to load the Java metadata.

To load Java metadata:

1. In the Administration Tool, go to the click **File** and then click **Load Java Datasources**. The Connect to Java Datasource server dialog displays.
2. Enter the enter hostname, port, and credentials to access the server and load the Java metadata.
3. Click **OK**. The Java metadata has been loaded from the server and is now available for import into the repository.

About Setting Up Oracle TimesTen In-Memory Database Data Sources

These preconfiguration instructions assume that you have already installed TimesTen; see *Oracle TimesTen In-Memory Database Installation Guide* for more information. Oracle TimesTen In-Memory Database is a high-performance, in-memory data manager that supports both ODBC and JDBC interfaces.

> **Note:** If you plan to create aggregates on your TimesTen source, you must also ensure that PL/SQL is enabled for the instance, and that the PL/SQL first connection attribute PLSQL is set to 1. You can enable PL/SQL at install time, or run the ttmodinstall utility to enable it post-install. See *Oracle TimesTen In-Memory Database Reference* for more information.

This section contains the following topics:

- Configuring TimesTen Data Sources
- Improving Use of System Memory Resources with TimesTen Data Sources
- Configuring OBIS to Access the TimesTen DLL on Windows

See also "Enabling NUMERIC Data Type Support for Oracle Database and TimesTen" for related information.

Configuring TimesTen Data Sources

You must configure TimesTen before you can use it as a data source for Oracle Business Intelligence.

To set up TimesTen data sources:

1. On the computer where TimesTen has been installed, create a Data Manager DSN (as a system DSN). See "Defining a Data Manager DSN" in *Oracle TimesTen In-Memory Database Operations Guide* for more information.
2. Perform an initial connection to the data store to load the TimesTen database into memory, and then create users and grant privileges, if you have not done so already. See "Managing Access Control" in *Oracle TimesTen In-Memory Database Operations Guide* for more information. Note that the default user of the data store is the instance administrator, or in other words, the operating system user who installed the database.
3. On the computer running the Oracle BI Server, install the TimesTen Client. See *Oracle TimesTen In-Memory Database Installation Guide* for more information.
4. On the computer where the TimesTen Client has been installed, create a Client DSN (as a system DSN). See "Creating Client DSNs" in *Oracle TimesTen In-Memory Database Operations Guide* for more information.

   See "Configuring Database Connections Using Native ODBC Drivers" for information about how to perform this step when the Oracle BI Server is running on Linux or UNIX.

Note that if the TimesTen database is installed on the same computer as the TimesTen client, you can specify either the Data Manager DSN or the Client DSN in the Import Metadata Wizard.

After importing data from your TimesTen source, or when manually setting up a database object and connection pool, ensure that your database type and version are set correctly in the **Database** field of the General tab of the Database dialog. You must

also ensure that the **Call interface** field in the General tab of the Connection Pool dialog is set correctly. See the following resources:

- "Creating a Database Object Manually in the Physical Layer"
- "Setting Connection Pool Properties in the General Tab"
- *Oracle Fusion Middleware Installation and Administration Guide for Oracle Exalytics In-Memory Machine* for specific instructions on setting up TimesTen sources on the Oracle Exalytics Machine

See "System Requirements and Certification" for information about supported TimesTen versions for Oracle Business Intelligence.

Improving Use of System Memory Resources with TimesTen Data Sources

To improve the use of system memory resources, Oracle recommends that you increase the maximum number of connections for the TimesTen server.

1. In your TimesTen environment, open the ttendaemon.options file for editing. You can find this file at:

   install_dir\srv\info

2. Add the following line:

   ```
   -MaxConnsPerServer number_of_connections
   ```

   To determine *number_of_connections*, use the following formula: if there are M connections for each connection pool in the Oracle BI repository, N connection pools in the Oracle BI repository, and P Oracle BI Servers, then the total number of connections required is M * N * P.

3. Save and close the file.
4. In the ODBC DSN you are using to connect to the TimesTen server, set the Connections parameter to the same value you entered in Step 2:
   - On Windows, open the TimesTen ODBC Setup wizard from the Windows ODBC Data Source Administrator. The Connections parameter is located in the First Connection tab.
   - On UNIX, open the odbc.INI file and add the Connections attribute to the TimesTen DSN entry, as follows:

     ```
     Connections=number_of_connections
     ```

5. Stop all processes connecting to TimesTen, such as the ttisql process and the Oracle BI Server.
6. Stop the TimesTen process.
7. After you have verified that the TimesTen process has been stopped, restart the TimesTen process.

Note: To avoid lock timeouts, you might also want to adjust the LockWait interval for the connection as appropriate for your deployment. See "LockWait" in *Oracle TimesTen In-Memory Database Reference* for more information.

Configuring OBIS to Access the TimesTen DLL on Windows

If the user that starts OBIS does not have the path to the TimesTen DLL ($TIMESTEN_HOME\lib) in their operating system PATH variable, then you must add the TimesTen DLL path as a variable in the obis.properties file.

To update obis.properties to include TimesTen variables on Windows:

1. Open obis.properties for editing. You can find obis.properties at:

   BI_DOMAIN\config\fmwconfig\bienv\obis

2. Add the required TimesTen variable `TIMESTEN_DLL`, and also update the `LD_LIBRARY_PATH` variable, as shown in the following example.

   ```
   TIMESTEN_DLL=$TIMESTEN_HOME\lib\libttclient.so
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$TIMESTEN_HOME\lib
   ```

3. Save and close the file.
4. Restart OBIS1.
5. Repeat these steps on each computer that runs the Oracle BI Server process. If you are running multiple Oracle BI Server instances on the same computer, be sure to update the ias-component tag appropriately for each instance in obis.properties (for example, ias-component id="coreapplication_obis1", ias-component id="coreapplication_obis2", and so on).

About Setting Up Essbase Data Sources

The Oracle BI Server uses the Essbase client libraries to connect to Essbase data sources. The Essbase client libraries are installed by default with Oracle BI Enterprise Edition. No additional configuration is required to enable Essbase data source access for full installations of Oracle BI Enterprise Edition.

About Setting up Cloudera Impala Data Sources

Use the information in this section to set up Cloudera Impala data sources in the Oracle BI repository. This topic contains the following sections:

- Obtaining Windows ODBC Driver for Client Installation
- Importing Cloudera Impala Metadata Using the Windows ODBC Driver

Obtaining Windows ODBC Driver for Client Installation

If you performed a client installation, then you will *not* have the Windows ODBC driver required for you to import Cloudera Impala metadata.

To obtain the required drivers, you must perform the following procedure. Note that if you used the Oracle Business Intelligence Installer to install the Administration Tool, then you do not have to perform this procedure.

To obtain the Windows ODBC driver:

1. Go to Cloudera's website, located at:

   http://www.cloudera.com

2. Click the Downloads link and then click the Impala ODBC Drivers & Connectors link.
3. In the Download list, locate the required ODBC driver for your Administration Tool platform and click **Download Bits** to download the installer.

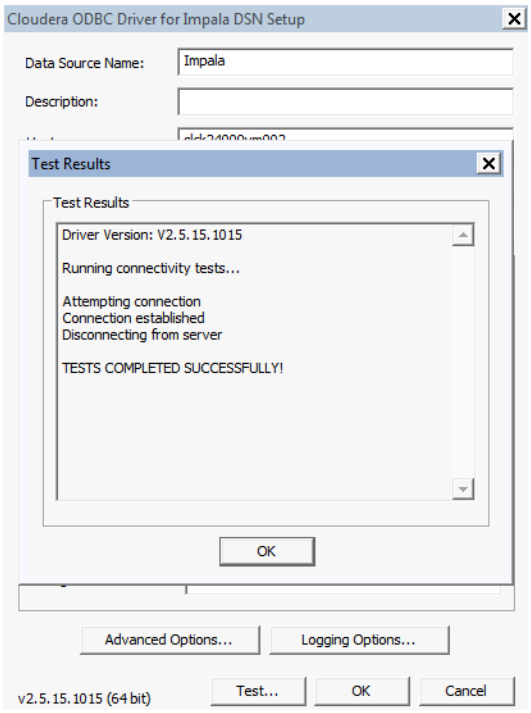
4. Run the ODBC driver installer to install the driver.

Importing Cloudera Impala Metadata Using the Windows ODBC Driver

Cloudera Impala is a massively parallel processing (MPP) SQL query engine that runs natively in Apache Hadoop. Perform this procedure to import Cloudera Impala metadata into the Oracle BI repository.

To perform this procedure, you must have the required Windows ODBC driver. If you have a client installation of the Administration Tool, then you must follow the "Obtaining Windows ODBC Driver for Client Installation" procedure to install the required Windows ODBC driver.

1. In Windows, locate and open the ODBC Data Source Administrator. The ODBC Data Source Administrator dialog appears.
2. In the ODBC Data Source Administrator dialog, click the System DSN tab, and then click **Add**. The Create New Data Source dialog appears.
3. In the driver list, locate and select a Cloudera Impala driver. Click **Finish**. The Cloudera ODBC Driver for Impala DSN Setup dialog appears.
4. Enter the connection details for your Impala instance. Note the following:
   - In the **Data Source Name** field, enter the data source name specified in the connection pool defined in the repository.
   - In the **Host** field, enter the fully qualified host name or the IP address.
   - In the **Port** field, enter the port number. The default is 21050.
   - In the **Database** field, specify the database. This value is usually Default.
5. If you are setting up a data source for Cloudera Impala driver, then click **Test**. If you are setting up a data source for DataDirect Impala driver, then click **Test Connect**. The Test Results dialog appears.



6. In the Administration Tool, select **File**, then select **Import Metadata**. The Import Metadata Wizard appears.
7. In the Select Data Source screen, confirm that ODBC 3.5 displays in the **Connection Type** field. Then, select the Impala DSN that you created, and provide a user name and password. Click **Next**.
8. In the Select Metadata Types screen, click **Next** to accept the default values.
9. In the Select Metadata Objects screen, go to the Data source view list and select the Impala tables for import and click the **>** (Import selected) button to move them to the Repository view list.
10. Click **Finish**. The Impala database appears in the Physical Layer of the Oracle BI repository.
11. In the Physical Layer of the repository, double click the Impala database. The Database dialog appears.
12. In the **Database type** field, choose "Cloudera Impala." Click **OK**.
13. Click **Save** to save the repository.
14. (Optional) Model the newly imported data as necessary in the Business Model and Mapping layer and the Presentation layer.

About Setting Up Apache Hive Data Sources

This topic contains the following sections:

Obtaining Windows ODBC Driver for Client Installation

If you have a client install of the Administration Tool, then you will *not* have the Windows ODBC driver required for you to import Apache Hive metadata.

To obtain the Windows driver required to perform the import, log in to the My Oracle Support web site support.oracle.com and access DocID 1520733.1. The technical note associated with this DocID includes the required Windows driver, together with the instructions to install the driver and to perform the metadata import from the Hive data source.

Limitations on the Use of Apache Hive with Oracle Business Intelligence

This section describes the limitations on the use of Hadoop and Hive with Oracle Business Intelligence. This section contains the following topics:

Hive Limitation on Dates Hive supports the Timestamp data type. Timestamp columns should be given the type DATE or DATETIME in the repository's Physical layer.

Hive Does Not Support Count (Distinct M) Together with Group By M Queries of the form:

```
SELECT M, COUNT(DISTINCT M) ... FROM ... GROUP BY M ...
```

may cause Hive to crash.

The situation occurs when the attribute in the COUNT(DISTINCT...) definition is queried directly and if that attribute is also part of the table or foreign key or level key.

Note that, as COUNT(DISTINCT X) together with GROUP BY X always results in the count being 1, a significant number of occurrences of this case are unlikely to happen.

To avoid such errors, when using COUNT(DISTINCT...) on a measure, do not include the exact attribute or any attribute in the same level.

Hive Does Not Support Differing Case Types Hive requires a strict check on types of the various parts of the Case statement. This causes a presentation query such as the following to fail in Hive:

```
select supplierid, case supplierid when 10 then 'EQUAL TO TEN' when 20 then
'EQUAL TO TWENTY' else 'SOME OTHER VALUE' end as c2 from supplier order by c2
asc, 1 desc
```

The full error message in Hive for this query is:

```
FAILED: Error in semantic analysis: Line 2:32 Argument type mismatch '10':
The expressions after WHEN should have the same type with that after CASE:
"smallint" is expected but "int" is found
```

Exception Thrown for Locate Function with an Out-of-Bounds Start Position Value The full syntax of the Locate function is of the form:

```
LOCATE ( charexp1, charexp2, [, startpos] )
```

where `charexp1` is the string to search for within the string `charexp2`.

The optional parameter `startpos` is the character position within `charexp2` at which to begin the search.

If `startpos` has a value that is longer than the length of `charexp2`, such as in the following example:

```
select locate('c', 'abcde', 9) from employee
```

then Hive throws an exception instead of returning 0.

Hive May Crash on Queries Using Substring Some queries that use the Substring function with a start position parameter value may cause Hive to crash, for example:

```
select substring(ProductName, 2) from Products
```

Hive Does Not Support Create Table As the Apache Hive ODBC driver does not support SQLTransact, which is used for creating tables, CREATE TABLE is not supported by Hive.

Hive May Fail on Long Queries With Multiple AND and OR Clauses The following WHERE clauses are examples of conditions that may cause queries to fail in Hive due to their excessive length:

Example 1

```
WHERE (Name = 'A' AND Id in (1))
   OR (Name = 'B' AND Id in (2))
   OR  .......
   OR (Name = 'H' AND Id in (8))
```

Example 2

```
WHERE (Id BETWEEN '01' AND '02')
   OR (Id BETWEEN '02' AND '03')
   OR  .......
   OR (Id BETWEEN '07' AND '08'))
```

In general, long queries may fail in Hive, but particularly if they have conditions with many OR clauses, each grouping together combinations of AND and BETWEEN sub-clauses, as shown in the preceding examples.

Queries with Subquery Expressions May Fail Queries with subquery expressions may fail in Hive. If subquery expressions are used, the physical query that Oracle BI Server generates may include mixed data types in equality conditions. Because of Hive issues in equality operators, the query result may not be correct.

For example, for the following query:

```
select ReorderLevel from Product where ReorderLevel in
  (select AVG(DISTINCT ReorderLevel) from Product);
```

Oracle BI Server generates the following physical query that includes '`ReorderLevel = 15.0`' where `ReorderLevel` is of type Int and `15.0` is treated as Float:

```
Select T3120.ReorderLevel as c1 from Products T3120
 where (T3120.ReorderLevel = 15.0)
```

This can be corrected by using the following command:

```
select ReorderLevel from Product where ReorderLevel in
  (select cast(AVG(DISTINCT ReorderLevel) as integer) from Product);
```

Hive Does Not Support Distinct M and M in Same Select List Queries of the following form are not supported by Hive:

```
SELECT DISTINCT M, M  ... FROM TABX
```

About Setting Up Hyperion Financial Management Data Sources

This section describes required configuration steps to use Hyperion Financial Management as a data source.

Hyperion Financial Management 11.1.2.3.*x* or 11.1.2.4.*x* can use the ADM native driver or the ADM thin client driver. The ADM thin client driver can be installed and configured on Linux.

Note: You can use the Hyperion Financial Management 11.1.2.3.*x* and 11.1.2.4.*x* data sources with Oracle BI EE running in a Windows or Linux deployment.

Installing the Hyperion Financial Management ADM driver includes both the drivers (ADM native driver and ADM thin client driver). For both Windows and Linux deployments, ensure that you perform the configuration using the Enterprise Performance Management Configurator.

- In the Windows and Linux configurations, provide the details for the Hyperion Shared Services Database to register with the Foundation server and the Hyperion Financial Management server.
- During configuration, make sure to enable DCOM configuration.
- If you are configuring for Windows, then in the DCOM User Details page, enter a domain user as the user for connecting to the Hyperion Financial Management server. Note that if you are configuring the ADM thin client driver for Linux, then you do not need to perform this step.

In addition, you must edit the obijh.properties file on each system that is running the Oracle BI JavaHost process to include environment variables that are required by Hyperion Financial Management.

Note that the JavaHost process must be running to import from Hyperion Financial Management data sources, for both offline and online imports. If you have a client installation of the Administration Tool, then see also "Peforming Additional Hyperion Configuration for Client Installations" for JavaHost configuration steps.

To configure Hyperion Financial Management:

1. Run the setOBIJHEnv command to properly set the Oracle BI Javahost environment. For example, on Windows, run *ORACLE_HOME*\bi\modules\oracle.bi.cam.obijh\setOBIJHEnv.cmd.

   Use the following example as a guide. Update the actual values as appropriate for your installation.

   ```
   SET EPM_ORACLE_HOME=C:\Oracle\Middleware\EPMSystem11R1
   SET EPM_ORACLE_INSTANCE=C:\Oracle\Middleware\user_projects\epmsystem1
   # SET HFM_ADM_TRACE=2

   SET OBIJH_ARGS=%OBIJH_ARGS% -DEPM_ORACLE_HOME=%EPM_ORACLE_HOME%
   SET OBIJH_ARGS=%OBIJH_ARGS% -DEPM_ORACLE_INSTANCE=%EPM_ORACLE_INSTANCE%
   SET OBIJH_ARGS=%OBIJH_ARGS% -DHFM_ADM_TRACE=2
   ```

2. Open the obijh.properties file for editing. You can find obijh.properties at:

   ORACLE_HOME\bi\modules\oracle.bi.cam.obijh\env\obijh.properties

3. In the OBIJH_ARGS section, add the DEPM_ORACLE_HOME and DEPM_ORACLE_INSTANCE configuration properties:

   Use the following example as a guide. Update the actual values as appropriate for your installation.

   ```
   -DEPM_ORACLE_HOME=C:\Oracle\Middleware\EPMSystem11R1
   -DEPM_ORACLE_INSTANCE=C:\Oracle\Middleware\user_projects\epmsystem1 -DHFM_ADM_
   TRACE=2
   ```

4. Open the loaders.xml file for editing. You can find loaders.xml at:

   ORACLE_HOME\bi\bifoundation\javahost\config\loaders.xml

5. Add the new variables that are required for Hyperion Financial Management. Use the following example as a guide. Update the actual values as appropriate for your installation.

```
EPM_ORACLE_HOME=\scratch\aime1\Oracle\Middleware\EPMSystem11R1;
EPM_ORACLE_INSTANCE=scratch\aime1\Oracle\Middleware\user_projects\epmsystem1
```

6. Locate <!-- BI Server integration code -->, and in <ClassPath> add the fm-adm-driver.jar, fm-web-objectmodel.jar, epm_j2se.jar, and epm_hfm_web.jar files. Use the following example as a guide. Update the actual values as appropriate for your installation.

```
<ClassPath>
  $EPM_ORACLE_HOME\common\hfm\11.1.2.0\lib\fm-adm-driver.jar$;
  $EPM_ORACLE_HOME\common\hfm\11.1.2.0\lib\fm-web-objectmodel.jar$;
  $EPM_ORACLE_HOME\common\jlib\11.1.2.0\epm_j2se.jar;
  $EPM_ORACLE_HOME\common\jlib\11.1.2.0\epm_hfm_web.jar
</ClassPath>
```

7. Save and close the file.
8. Go to the *ORACLE_HOME*\bi\bifoundation\javahost\lib\obisintegration\adm directory and delete all jar files except for admintegration.jar and admimport.jar.
9. Restart OBIS1.
10. Repeat these steps on each computer that runs the Oracle BI JavaHost process. If you run multiple JavaHost instances on the same computer, then ensure that you update the ias-component tag appropriately for each instance in obis.properties (for example, ias-component id="coreapplication_obijh1", ias-component id="coreapplication_obijh2", and so on).

    Run more than one JavaHost process to ensure that JavaHost is not a single point of failure for access to Hyperion Financial Management. To do this, scale out the JavaHost process using Fusion Middleware Control. See "Scaling Your Deployment" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for information about scaling out processes.

Peforming Additional Hyperion Configuration for Client Installations

If you install the Administration Tool using the Oracle Business Intelligence Enterprise Edition Plus Client Installer, you must perform additional configuration before you can perform offline imports from Hyperion Financial Management data sources. To import from Hyperion Financial Management data sources in offline mode, the Administration Tool must point to the location of a running JavaHost.

The steps in this section are only required for client installations of the Administration Tool.

To point the Administration Tool at a running JavaHost:

1. Close the Administration Tool, if it is open.
2. On the same computer as the Administration Tool, open the local NQSConfig.INI file in a text editor. You can find this file at:

    BI_DOMAIN\config\fmwconfig\biconfig\OBIS

3. Locate the JAVAHOST_HOSTNAME_OR_IP_ADDRESSES parameter, near the bottom of the file. Update this parameter to point to a running JavaHost, using a fully-qualified host name or IP address and port number. For example:

```
JAVAHOST_HOSTNAME_OR_IP_ADDRESSES = "myhost.example.com:9810"
```

Note that in a full (non-client) Oracle Business Intelligence installation, you cannot manually edit this setting because it is managed by Oracle Enterprise Manager Fusion Middleware Control.

4. Save and close the file.

Setting Up SAP/BW Data Sources

You can connect to SAP/BW data sources using either the XMLA connection type, or the SAP BW Native connection type (BAPI). SAP BW Native connections might not be available on certain platforms.

See "System Requirements and Certification" for more information.

To connect to SAP/BW data sources using the SAP BW Native connection type, you must first download the **OBIEE BAPI Adapter for SAP** from the Bristlecone Web site at:

`http://www.bristleconelabs.com/edel/downloads.html`

Then, follow the configuration instructions in the documentation provided with the download.

No preconfiguration steps are required to connect to SAP/BW over XMLA.

Setting Up Oracle RPAS Data Sources

Oracle BI Server can connect to Oracle RPAS (Retail Predictive Application Server) data sources through ODBC DSNs.

To set up Oracle RPAS data sources, you must first install the Oracle RPAS ODBC driver. During set up of the ODBC DSN, you must select the **SQLExtendedFetch** option, select DBMS from the **Authentication Method** list, and select No from the **Normalize Dimension Tables** list. See "About Importing Metadata from Oracle RPAS Data Sources" for more information.

On Windows systems, you can connect to Oracle RPAS data sources for both initial import and for access to the data source during query execution. On UNIX systems, you can only connect to Oracle RPAS data sources for data access.

See "Configuring Oracle RPAS ODBC Data Sources on AIX UNIX" for information about how to enable ODBC access to Oracle RPAS data sources when the Oracle BI Server is running on AIX UNIX.

Setting Up Teradata Data Sources

You can use ODBC to access Teradata data sources.

See "Setting Up ODBC Data Source Names (DSNs)" for information about setting up ODBC connections for Teradata.

After you have installed the latest Teradata ODBC driver and set up an ODBC DSN, you must add the lib directory for your Teradata data source to your Windows system Path environment variable. For example:

C:\Program Files\Teradata\Client\15.00\ODBC Driver for Teradata nt-x8664\Lib

In addition, you must manually edit obis.properties on each computer running the Oracle BI Server to include required Teradata variables.

To update obis.properties to include Teradata variables on Windows:

1. Open obis.properties for editing. You can find obis.properties at:

BI_DOMAIN\config\fmwconfig\bienv\obis

2. In PATH, LD_LIBRARY_PATH, and LIBPATH enter the required variable information as shown in the following example.

```
PATH=C:\Program Files\Teradata\Client\15.00\ODBC Driver for Teradata
nt-x8664\Lib;
LD_LIBRARY_PATH=C:\Program Files\Teradata\Client\15.00\ODBC Driver for
Teradata nt-x8664\Lib;
LIBPATH=C:\Program Files\Teradata\Client\15.00\ODBC Driver for Teradata
nt-x8664\Lib;
```

Important: If you use the default location when installing the Teradata client, then the PATH variable might exceed the 1024 character limit imposed by Windows. To avoid this issue, install the Teradata client in a directory with a shortened path name (such as C:\TD), or use shortened 8.3 file names (such as "C:\PROGRA~1\Teradata\Client\13.10\ODBCDR~1\Bin" instead of "C:\Program Files\Teradata\Client\13.10\ODBC Driver for Teradata\Bin").

To determine the correct 8.3 file names, run "dir /x" from the appropriate directory. For example:

```
C:\>dir /x
 Volume in drive C has no label.
 Volume Serial Number is 0000-XXXX
 Directory of C:\
08/25/2008  03:36 PM   <DIR>    DATAEX~1    DataExplorer
04/20/2007  01:38 PM   <DIR>                dell
08/28/2010  10:49 AM   <DIR>    DOCUME~1    Documents and Settings
07/28/2008  04:50 PM   <DIR>    ECLIPS~2    EclipseWorkspace
09/07/2007  11:50 AM   <DIR>                Ora92
09/07/2007  11:50 AM   <DIR>                oracle
05/21/2009  05:15 PM   <DIR>                OracleBI
05/21/2009  05:12 PM   <DIR>    ORACLE~1    OracleBIData
03/02/2011  04:51 PM   <DIR>    PROGRA~1    Program Files
```

3. Save and close the file.
4. Restart OBIS1.
5. Repeat these steps on each computer that runs the Oracle BI Server process. If you are running multiple Oracle BI Server instances on the same computer, be sure to update the ias-component tag appropriately for each instance in obis.properties (for example, ias-component id="coreapplication_obis1", ias-component id="coreapplication_obis2", and so on).

Avoiding Spool Space Errors for Queries Against Teradata Data Sources

Some queries against Teradata might get a "No more spool space" error from the data source. This error can occur for DISTINCT queries resulting from selecting All Choices in the Filters pane in Answers.

To avoid this error, you can ensure that the Oracle BI Server rewrites the query to use GROUP BY rather than DISTINCT for these queries. To do this, ensure the following conditions are met:

- There is only one dimension column in the projection list, and it is a target column rather than a combined expression.
- The original query from Answers is requesting DISTINCT, and does not include a GROUP BY clause
- The FROM table is a real physical table rather than an opaque view.
- The FROM table is an atomic table, not a derived table.
- The following ratio must be less than the threshold:

  (the distinct number of the projected column) / (number of rows of FROM table)

  Both values used in this ratio come from the repository metadata. To populate these values, click **Update Row Count** in the Administration Tool for both of the following objects:

  – The FROM physical table

  – The physical column for the projected column

  By default, the threshold for this ratio is 0.15. To change the threshold, create an environment variable on the Oracle BI Server computer called SA_CHOICES_CNT_SPARSITY and set it to the new threshold.

Enabling NUMERIC Data Type Support for Oracle Database and TimesTen

You can enable NUMERIC data type support for Oracle Database and TimesTen data sources. When NUMERIC data type support is enabled, NUMBER columns in Oracle Database and TimesTen data sources are treated as NUMERIC in Oracle Business Intelligence, which provides greater precision. In addition, literals are instantiated as NUMERIC instead of DOUBLE for Oracle Database and TimesTen data sources.

See also "Numeric Literals" for related information.

To enable NUMERIC data type support for Oracle Database and TimesTen data sources:

1. Set `ENABLE_NUMERIC_DATA_TYPE` to `YES` in NQSConfig.INI. For instructions, see "ENABLE_NUMERIC_DATA_TYPE" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.
2. Enable the NUMERIC_SUPPORTED database feature in the Physical layer database object. See "Specifying SQL Features Supported by a Data Source" for more about how to set database features.

After you have performed these steps, data imported into the Oracle BI repository from Oracle Database and TimesTen will have decimal/numeric data set to NUMERIC, and decimal/numeric SQL entered by users will be treated as NUMERIC. Note that decimal/numeric data from other database types is still mapped as DOUBLE, even when this parameter is set to YES.

The data type of physical columns imported prior to changing this setting will remain the same. In other words, a column in Oracle Database or TimesTen that is declared as DOUBLE instead of NUMBER is still imported as DOUBLE in Oracle Business Intelligence, regardless of how this parameter is set. For existing DOUBLE physical columns, manually update the data type to NUMBER as needed.

Note the following:

- Numeric data types can be cast to other Number data types, and vice versa.

- Numeric data type support is not available through the Oracle BI Server JDBC driver.
- There might be a performance overhead of enabling the numeric data type because of the higher number of bits for numeric data.

Configuring Essbase to Use a Shared Logon

When using the Essbase Cube Deployment Services wizard to create a new Essbase connection pool in the Oracle BI repository, there is no opportunity to enter the shared logon username and password. However, in the current version of Oracle BI a shared logon is required for all Essbase connection pools.

An Essbase connection pool that is configured without a shared logon causes the Oracle BI Server to use the CSS shared token authentication, which is no longer supported for Essbase connections from Oracle BI. This might cause Essbase connections from certain Oracle BI components such as Visual Analyzer to fail with the following error:

```
[nQSError: 96002] Essbase Error: Login failed with error 1051293. Verify the URL
and the login credentials.
```

There are two ways to work around this issue:

- While using the Essbase Cube Deployment Services wizard, the user can select an existing Essbase data source and connection pool from the repository that already has the shared logon information provided.
- After completing the Essbase Cube Deployment Services wizard, the user can open the repository in the Administration Tool and manually update the connection pool created by the wizard to provide the required shared logon information. The user can also uncheck the SSO option if it is not required.

  See "Multidimensional Connection Pool Properties in the General Tab" for information about how to use the SSO and Shared Logon options in Essbase connection pools.

Importing Metadata from Relational Data Sources

You can import metadata for supported relational data source types by selecting the appropriate connection type in the Import Metadata Wizard. To import metadata, you must have all database connections set up on your local computer. You can import metadata in both offline and online modes.

See "Importing Metadata from Multidimensional Data Sources" and Chapter 6, "Working with ADF Data Sources" for information about importing from other data sources.

When you import physical tables, be careful to limit the import to only those tables that contain data that are likely to be used in the business models you create. You can use the Find feature to locate and select the tables that you want to import. Importing large numbers of extraneous tables and other objects adds unnecessary complexity and increases the size of the repository.

When you import metadata for most data sources, the default is to import tables, primary keys, and foreign keys. It is recommended that you import primary and foreign keys along with your tables so that the keys are automatically created in the Physical layer. If you do not import keys, you must create them manually, which can be a time-consuming process.

You can also import database views, aliases, synonyms, and system tables. Import these objects only if you want the Oracle BI Server to generate queries against them.

If you are importing metadata into an existing database in the Physical layer, then confirm that the COUNT_STAR_SUPPORTED option is selected in the Features tab of the Database properties dialog. If you import metadata without the COUNT_STAR_SUPPORTED option selected, then the **Update Row Count** option will not display in the right-click menu for the database's physical tables.

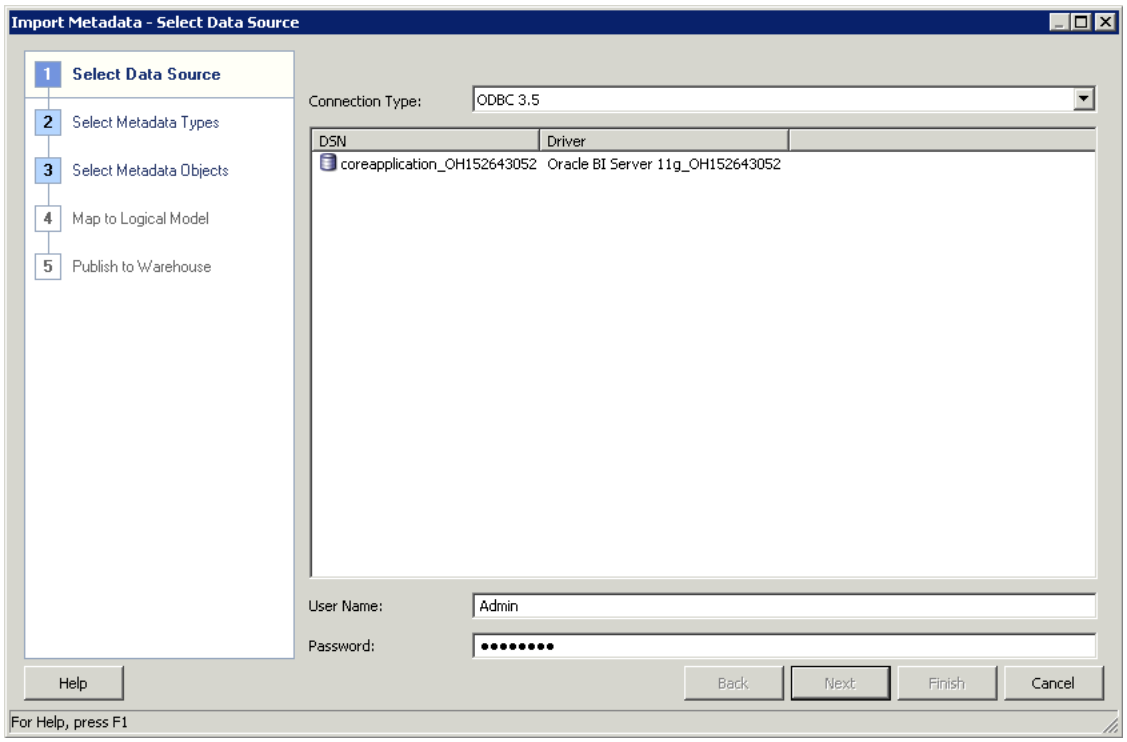
To import metadata from a relational data source:

1. In the Administration Tool, select **File**, then select **Import Metadata**. The Import Metadata Wizard appears.

   Note: If you have already defined an existing database and connection pool, you can right-click the connection pool in the Physical layer and select **Import Metadata**. The Import Metadata Wizard appears with the information on the Select Data Source screen pre-filled.

   Figure 5–1 shows the Import Metadata Wizard.

Figure 5–1 Import Metadata Wizard: Relational Data Source



2. In the Select Data Source screen, in the **Connection Type** field, select the type of connection appropriate for your data source, such as **ODBC 3.5**.

   Make sure to choose **OCI 10g/11g** if your data source is an Oracle Database. Using OCI as your connection protocol to an Oracle Database ensures better performance and provides access to native database features that are not available through ODBC.

Note: For non-Oracle databases, it is recommended that you use ODBC 3.5 or DB2 CLI (Unicode) for importing schemas with International characters, such as Japanese table and column names.

The remaining fields and options on the Select Data Source screen vary according to the connection type you selected:

– For **ODBC 2.0** and **ODBC 3.5** data sources, in the **DSN** list, select a data source from which to import the schema. Then, provide a valid user name and password for the data source.

  Note that when you import through the Oracle BI Server, the DSN entries are on the Oracle BI Server computer, not on the local computer.

– For **OCI 10g/11g** and **DB2 CLI (Unicode)** data sources, provide the name of the data source in the **Data Source Name** field, then provide a valid user name and password for the data source.

  For Oracle Database data sources, the data source name is either a full connect string or a net service name from the tnsnames.ora file. If you enter a net service name, you must ensure that you have set up a tnsnames.ora file within the Oracle Business Intelligence environment, in:

  BI_DOMAIN\config\fmwconfig\bienv\core

Other data source types are described in other sections:

- See "Importing Metadata from Multidimensional Data Sources" for **Essbase**, **XMLA**, **Oracle OLAP**, **Hyperion ADM**, and **SAP BW Native**. This section also describes importing from Oracle RPAS data sources over ODBC 3.5.
- See "About Importing Metadata from XML Data Sources" for **XML**.
- See Chapter 6, "Working with ADF Data Sources" for **OracleADF_HTTP**.

When you have finished providing information in the Select Data Source screen, click **Next**. The Select Metadata Types screen appears.

3. Select the options for the types of objects that you want to import (for example, **Tables**, **Keys**, and **Foreign Keys**). Some options are automatically selected. Different types of data sources have different default selections, based on what is typical for that data source.

   If you want to import joins, select both **Keys** and **Foreign Keys**. If you want to import system tables, you must have the system privilege for your data source. To import from Customer Relationship Management (CRM) tables, select **Metadata from CRM tables**.

4. Click **Next**. The Select Metadata Objects screen appears.

5. Select the objects you want to import in the **Available** list and move them to the **Selected** list, using the **>** (Import selected) or **>>** (Import all) buttons. You can also move objects from the **Selected** list back to the **Available** list, using the **<** (Remove selected) and **<<** (Remove all) buttons.

   To search for a particular item, enter a keyword in the **Find** box and then click **Find Down** or **Find Up**.

   Select **Show complete structure** to view all objects, including those that have already been imported. Deselecting this option shows only the objects that are available for import. When this option is selected, objects that have already been imported appear grayed out.

6. Click **Finish**.

   If some objects could not be imported, a list of warning messages appears. In the dialog displaying the messages, you can perform the following actions:

   – To search for specific terms, click **Find** and then **Find Again**.

   – To copy the contents of the window so that you can paste the messages in another file, click **Copy**.

After you import metadata, you should check to ensure that your database and connection pool settings are correct. In rare cases, the Oracle BI Server cannot determine the exact database type during import and instead assigns an approximate type to the database object. See "Setting Up Database Objects" and "Creating or Changing Connection Pools" for more information about working with these objects.

It is also a good practice to visually inspect the imported data in the Physical layer, such as physical columns and tables, to ensure that the import completed successfully.

About the Map to Logical Model and Publish to Warehouse Screens

The Map to Logical Model and Publish to Warehouse screens are available only for ADF data sources.

For more information about the Map to Logical Model screen, see "Automatically Mapping Flex Object Changes to the Logical Model".

Importing Metadata from Multidimensional Data Sources

You can import metadata from a multidimensional data source to the Physical layer of the Oracle BI repository. Using multidimensional data sources enables the Oracle BI Server to connect to and extract data from sources such as Essbase, Oracle OLAP, Hyperion Financial Management, Hyperion Planning, Microsoft Analysis Services, and SAP/BW (SAP/Business Warehouse).

The primary differences between setting up multidimensional data sources and relational data sources are in the Physical layer. The setup in the Business Model and Presentation layers for multidimensional data sources and relational data sources is almost identical.

During the import process, each cube in a multidimensional data source is created as a single physical cube table. The Oracle BI Server imports the cube metadata, including its metrics, dimensions, and hierarchies. After importing the cubes, you need to ensure that the physical cube columns have the correct aggregation rule and that the hierarchy type is correct. See "Working with Physical Hierarchy Objects" for more information.

> **Caution:** Manually creating a physical schema from a multidimensional data source is labor-intensive and error prone. Therefore, it is strongly recommended that you use the import method.

It is recommended that you remove hierarchies and columns from the Physical layer if they will not be used in the business model. This eliminates maintaining unnecessary objects in the Administration Tool and might result in better performance.

If you are importing metadata into an existing database in the Physical layer, then confirm that the COUNT_STAR_SUPPORTED option is selected in the Features tab of

the Database properties dialog. If you import metadata without the COUNT_STAR_SUPPORTED option selected, then the **Update Row Count** option will not display in the right-click menu for the database's physical tables.

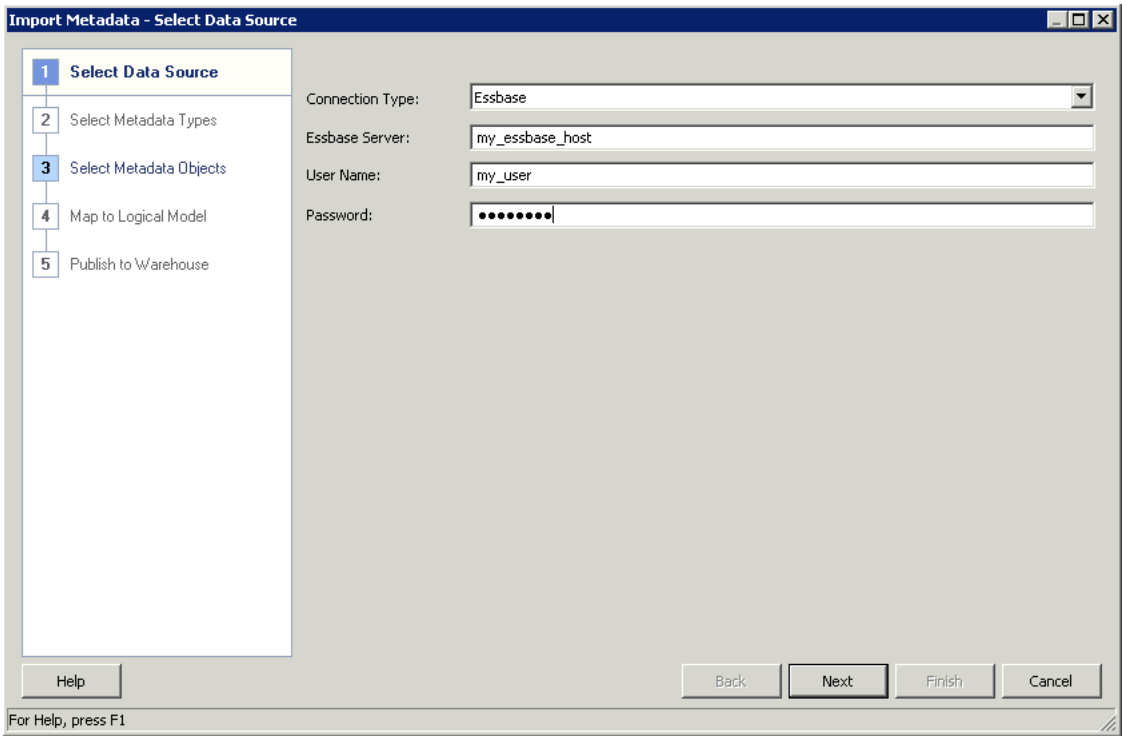
To import metadata from a multidimensional data source:

1. In the Administration Tool, select **File**, then select **Import Metadata**. The Import Metadata Wizard appears.

   Note: If you have already defined an existing database and connection pool, you can right-click the connection pool in the Physical layer and select **Import Metadata**. The Import Metadata Wizard appears with the information on the Select Data Source screen pre-filled. You can also use this method to perform incremental imports.

   Figure 5–2 shows the Import Metadata Wizard.

Figure 5–2 Import Metadata Wizard: Multidimensional Data Source



   Note that the Map to Logical Model and Publish to Warehouse screens are available only for ADF data sources. "About the Map to Logical Model and Publish to Warehouse Screens" for more information.

2. In the Select Data Source screen, in the **Connection Type** field, select the type of connection appropriate for your data source.

   The remaining fields and options on the Select Data Source screen vary according to the connection type you selected. Table 5–1 describes the multidimensional connection types.

Table 5–1 Multidimensional Connection Options

Connection Type	Description
ODBC 3.5	The ODBC 3.5 connection type is used for Oracle RPAS data sources. Select the DSN entry and provide the user name and password for the selected data source. See "Setting Up ODBC Data Source Names (DSNs)" for more information.
Essbase 9+	Use this option for Essbase 9 or Essbase 11. Provide the host name of the computer where the Essbase Server is running in the **Essbase Server** field, then provide a valid user name and password for the data source. This information should be obtained from your data source administrator. If the Essbase Server is running on a non-default port, or if it is part of an Essbase Cluster, you must include the port number in the **Essbase Server** field, in the format `hostname:port`. Note the following: ■ You can import metadata from an Essbase Cluster, but you must still specify an individual Essbase Server host name and port number in the Essbase Server field. ■ See "Working with Essbase Data Sources" for information about how data from Essbase data sources is modeled in the Physical layer.
XMLA	The XMLA connection type is used for Microsoft Analysis Services and SAP/BW. Enter the URL of a data source from which to import the schema. Then, enter the **Provider Type** (such as Analysis Services 2000 or SAP/BW 3.5/7.0) and a valid user name and password for the data source. For **Target Database**, do one of the following: ■ Select **New** and enter the name you want to use for the new database object. ■ Select **Existing** and then click **Browse** to select an existing database object.

Table 5–1 (Cont.) Multidimensional Connection Options

Connection Type	Description
Oracle OLAP	Provide the name of the data source (net service name) in the **Data Source Name** field, then provide a valid user name and password for the data source. The data source name is the same as the entry you created in the tnsnames.ora file. Make sure that the name you use is from the tnsnames.ora file within the Oracle Business Intelligence environment. You can also choose to enter a full connect string rather than the net service name. For **URL**, provide the URL of the biadminservlet. This servlet is used for Oracle OLAP metadata imports. The name of the servlet is `services`. For example, enter a string similar to the following in the URL field: http://localhost:9704/biadminservlet/services Note that the servlet must be up and running in order to use it. If you receive an import error, then check the status of the servlet in the Administration Console. You can also refer to the Administration Server diagnostic log and the Domain log. See "Working with Oracle OLAP Data Sources" for information about how data from Oracle OLAP data sources is modeled in the Physical layer. **Note:** Oracle Database data sources with the OLAP option can contain both relational tables and multidimensional tables. However, you should avoid having both table types in the same database object in the Administration Tool, because you may need to specify different database feature sets for the different table types. For example, Oracle OLAP queries fail if the database feature `GROUP_BY_GROUPING_SETS_SUPPORTED` is enabled. However, you may need this feature enabled for Oracle Database relational tables. As a best practice, create two separate database objects for relational and multidimensional tables.

Table 5–1 (Cont.) Multidimensional Connection Options

Connection Type	Description

Hyperion ADM	Provide the URL for the Hyperion Financial Management or Hyperion Planning server.
	
	For Hyperion Financial Management 11.1.2.1 and 11.1.2.2 using the ADM native driver, include the driver and application name (cube name), in the following format:
	
	`adm:native:HsvADMDriver:`*`ip_or_host`*`:`*`application_name`*
	
	For example:
	
	`adm:native:HsvADMDriver:192.0.2.254:UCFHFM`
	
	For Hyperion Financial Management 11.1.2.3 and 11.1.2.4 using the ADM thin client driver, include the driver and application name (cube name), in the following format:
	
	`adm:thin:com.hyperion.ap.hsp.HspAdmDriver:[ip_or_host]:[port]:[application_name`
	
	For example:
	
	`adm:thin:com.hyperion.ap.hsp.HspAdmDriver:192.0.2.254:8300:UCFHP`
	
	For Hyperion Planning 11.1.2.4 or later, the installer does not deliver all of the required client driver .jar files. To ensure that you have all of the needed .jar files, go to your instance of Hyperion, locate and copy the adm.jar, ap.jar, and HspAdm.jar files and paste them into *`MIDDLEWARE_HOME`*`\oracle_common\modules`.
	
	For Hyperion Planning 11.1.2.4 or later using the ADM thin client driver, include the driver and application name (cube name), in the following format:
	
	`adm:thin:com.hyperion.ap.hsp.HspAdmDriver:[server]%3A[port]:[application_name`
	
	For example:
	
	`adm:thin:com.hyperion.ap.hsp.HspAdmDriver:server_name.example.com%3A8300:PFP_OFP3`
	
	You also need to select the provider type and enter a valid user name and password for your data source.
	
	Note that the JavaHost process must be running to import from Hyperion Financial Management or Hyperion Planning data sources, for both offline and online imports.
	
	See "Working with Hyperion Financial Management and Hyperion Planning Data Sources" for information about how data from Hyperion Financial Management and Hyperion Planning data sources is modeled in the Physical layer.
	
	Note: Be sure to complete the preconfiguration steps described in "About Setting Up Hyperion Financial Management Data Sources" before import.

Table 5–1 (Cont.) Multidimensional Connection Options

Connection Type	Description
SAP BW Native	Provide the following information: ■ **System IP or Hostname:** The host name or IP address of the SAP data server. This field corresponds to the parameter `ashost` in the SAP/BW connect string. ■ **System Number:** The SAP system number. This is a two-digit number assigned to an SAP instance, also called Web Application Server, or WAS. This field corresponds to the parameter `sysnr` in the SAP/BW connect string. ■ **Client Number:** The SAP client number. This is a three-digit number assigned to the self-contained unit called Client in SAP. A Client can be a training, development, testing, or production client, or it can represent different divisions in a large company. This field corresponds to the parameter `client` in the SAP/BW connect string. ■ **Language:** The SAP language code used when logging in to the data source (for example, EN for English or DE for German). This field corresponds to the parameter `lang` in the SAP/BW connect string. ■ **Additional Parameters:** Additional connection string parameters in the format param=value. Delimit multiple parameters with a colon. This field is optional. ■ **User Name:** A valid user name for the data source. ■ **Password:** The corresponding user password. The password is case-sensitive. The first five fields constitute the elements of the SAP/BW connect string, in the format: `ashost=`*`value`*`:sysnr=`*`value`*`:client=`*`value`*`:lang=`*`value`*`:`*`additional_param=value`* For example: `ashost=10.30.0.19:sysnr=00:client=100:lang=EN` **Note:** Be sure to complete the preconfiguration steps described in "Setting Up SAP/BW Data Sources" before import.

Other data source types are described in other sections:

- See "About Importing Metadata from XML Data Sources" for **XML**.
- See "Importing Metadata from Relational Data Sources" for **ODBC 2.0**, **OCI 10g/11g**, and **DB2 CLI (Unicode)**.
- See Chapter 6, "Working with ADF Data Sources" for **OracleADF_HTTP**.

When you have finished providing information in the Select Data Source screen, click **Next**.

3. For Oracle RPAS data sources only, the Select Metadata Types screen is displayed. For Oracle RPAS, select **Tables**, **Keys**, and **Foreign Keys**. Then, click **Next**.

   See "About Importing Metadata from Oracle RPAS Data Sources" for more information.

4. In the Select Metadata Objects screen, select the objects you want to import in the **Available** list and move them to the **Selected** list, using the **>** (Import selected) or **>>** (Import all) buttons. You can also move objects from the **Selected** list back to the **Available** list, using the **<** (Remove selected) and **<<** (Remove all) buttons.

To search for a particular item, enter a keyword in the **Find** box and then click **Find Down** or **Find Up**.

Select **Show complete structure** to view all objects, including those that have already been imported. Deselecting this option shows only the objects that are available for import. When this option is selected, objects that have already been imported appear grayed out

For Essbase data sources, select **Import UDAs** if you want to import UDAs (user-defined attributes).

5. Click **Finish**.

   If some objects could not be imported, a list of warning messages appears. In the dialog displaying the messages, you can perform the following actions:

   – To search for specific terms, click **Find** and then **Find Again**.

   – To copy the contents of the window so that you can paste the messages in another file, click **Copy**.

After you import metadata, you should check to ensure that your database and connection pool settings are correct. In rare cases, the Oracle BI Server cannot determine the exact database type during import and instead assigns an approximate type to the database object. See "Setting Up Database Objects" and "Creating or Changing Connection Pools" for more information about working with these objects.

It is also a good practice to visually inspect the imported data in the Physical layer, such as physical columns and hierarchical levels, to ensure that the import completed successfully.

For Essbase data sources, all hierarchies are imported as Unbalanced by default. Review the **Hierarchy Type** property for each physical hierarchy and change the value if necessary. Supported hierarchy types for Essbase are **Unbalanced**, **Fully balanced**, and **Value**.

About Importing Metadata from Oracle RPAS Data Sources

This section provides important information about using the Administration Tool to import metadata from Oracle RPAS, as follows:

- Oracle RPAS schemas can only be imported on Windows.
- Before you import RPAS schemas, you must set the **Normalize Dimension Tables** field value in the ODBC DSN Setup page to **Yes** for the following reasons:

  – Setting this value to **Yes** uses an appropriate schema model (the snowflake schema) that creates joins correctly and enables drill down in the data.

  – Setting this value to **No** uses a less appropriate schema model (the star schema) that creates joins between all of the tables, causing drill down to not work correctly. Many of the joins created in this way are unwanted, and would need to be removed manually.

  See "Setting Up ODBC Data Source Names (DSNs)" for more information.

- When you import RPAS schemas in the Administration Tool, you must import the data with joins. To do this, select the metadata types **Keys** and **Foreign Keys** in the Import Metadata Wizard.
- After you have imported RPAS schemas, you must change the **Normalize Dimension Tables** field value in the ODBC DSN Setup page back to **No**. You need

to revert this setting back to **No** after import to enable the Oracle BI Server to correctly generate optimized SQL against the RPAS driver.

Note: If you do not change the **Normalize Dimension Tables** setting value to **No**, most queries will fail with an error message similar to the following:

```
[nQSError: 16001] ODBC error state: S0022 code: 0 message: [Oracle Retail][RPAS
ODBC]Column:YEAR_LABEL not found..[nQSError: 16014] SQL statement preparation
failed. Statement execute failed.
```

- If Oracle RPAS is the only data source, you must set the value of `NULL_VALUES_SORT_FIRST` to `ON` in the NQSConfig.INI file. See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about setting values in NQSConfig.INI.

After you import metadata from an Oracle RPAS data source, a database object for the schema is automatically created. Depending on your version of RPAS, you might need to adjust the data source definition: Database property (located in the Database dialog's General tab) for the following reason.

If RPAS is specified in the **data source definition Database** field and the version of RPAS is prior to 1.2.2, then the BI Server performs aggregate navigation when the SQL is generated and sent to the database and not as it typically does in the logical table source navigation. Because the table name used in the generated SQL is automatically generated, there will be a mismatch between the generated SQL and the database table name. To enable the SQL to run, you must:

- Change the names of tables listed in the metadata so that the generated names are correct.
- Create tables in the database with the same names as the generated names.

If the database does not have tables with the same name or if you want to have the standard aggregate navigation within Oracle Business Intelligence, then you must change the **data source definition: Database** field from RPAS to ODBC Basic. See "Creating a Database Object Manually in the Physical Layer" for more information.

About Importing Metadata from XML Data Sources

This section describes how to import metadata from Extensible Markup Language (XML) documents. The Oracle BI Server supports two XML access modes: access through the Oracle BI Server XML Gateway, and access through an XML ODBC driver.

This section contains the following topics:

About Using XML as a Data Source

The Oracle BI Server supports the use of XML data as a data source for the Physical layer in the repository. Depending on the method used to access XML data sources, a data source may be represented by a URL pointing to a source.

Note the following sources:

- A static XML file or HTML file that contains XML data islands on the Internet (including intranet or extranet). For example:

  tap://216.217.17.176/[DE0A48DE-1C3E-11D4-97C9-00105AA70303].XML

- Dynamic XML generated from a server site. For example:

  tap://www.aspserver.com/example.asp

- An XML file or HTML file that contains XML data islands on a local or network drive. For example:

  d:\xmldir\example.xml

  d:\htmldir\island.htm

  You can also specify a directory path for local or network XML files, or you can use the asterisk (*) as a wildcard with the filenames. If you specify a directory path without a filename specification (like d:/xmldir), all files with the XML suffix are imported. For example:

  d:\xmldir\

  d:\xmldir\exam*.xml

  d:\htmldir\exam*.htm

  d:\htmldir\exam*.html

- An HTML file that contains tables, defined by a pair of <table> and </table> tags. The HTML file may reside on the Internet (including intranet or extranet), or on a local or network drive. See "About Using HTML Tables as a Data Source" for more information.

URLs can include repository or session variables, providing support for HTTP data sources that accept user IDs and passwords embedded in the URL. For example:

```
http://somewebserver/cgi.pl?userid=valueof(session_variable1)&password=
valueof(session_variable2)
```

This functionality also lets you create an XML data source with a location that is dynamically determined by some run-time parameters. For more information about variables, see Chapter 19.

If the Oracle BI Server needs to access any nonlocal files (network files or files on the Internet, for example), you must run the Oracle BI Server using a valid user ID and password with sufficient network privileges to access these remote files.

Importing Metadata from XML Data Sources Using the XML Gateway

Using the Oracle BI Server XML Gateway, the metadata import process flattens the XML document to a tabular form using the stem of the XML filename (that is, the filename without the suffix) as the table name and the second level element in the XML document as the row delimiter. All leaf nodes are imported as columns belonging to the table. The hierarchical access path to leaf nodes is also imported.

The Oracle BI Server XML Gateway uses the metadata information contained in an XML schema. The XML schema is contained within the XML document, or is referenced within the root element of the XML document.

Where there is no schema available, all XML data is imported as text data. In building the repository, you can alter the data types of the columns in the Physical layer, overriding the data types for the corresponding columns defined in the schema. The gateway converts the incoming data to the desired type as specified in the Physical

layer. You can also map the text data type to other data types in the Business Model and Mapping layer of the Administration Tool, using the `CAST` operator.

The Oracle BI Server XML Gateway does not support:

- Resolution of external references contained in an XML document (other than a reference to an external XML schema, as demonstrated in the example file in "Examples of XML Documents Generated by the Oracle BI Server XML Gateway").
- Element and attribute inheritance contained within the Microsoft XML schema.
- Element types of a mixed content model (such as XML elements that contain a mixture of elements and CDATA, such as hello Joe, how are you doing?).

If you are importing metadata into an existing database in the Physical layer, then confirm that the COUNT_STAR_SUPPORTED option is selected in the Features tab of the Database properties dialog. If you import metadata without the COUNT_STAR_SUPPORTED option selected, then the **Update Row Count** option will not display in the right-click menu for the database's physical tables.

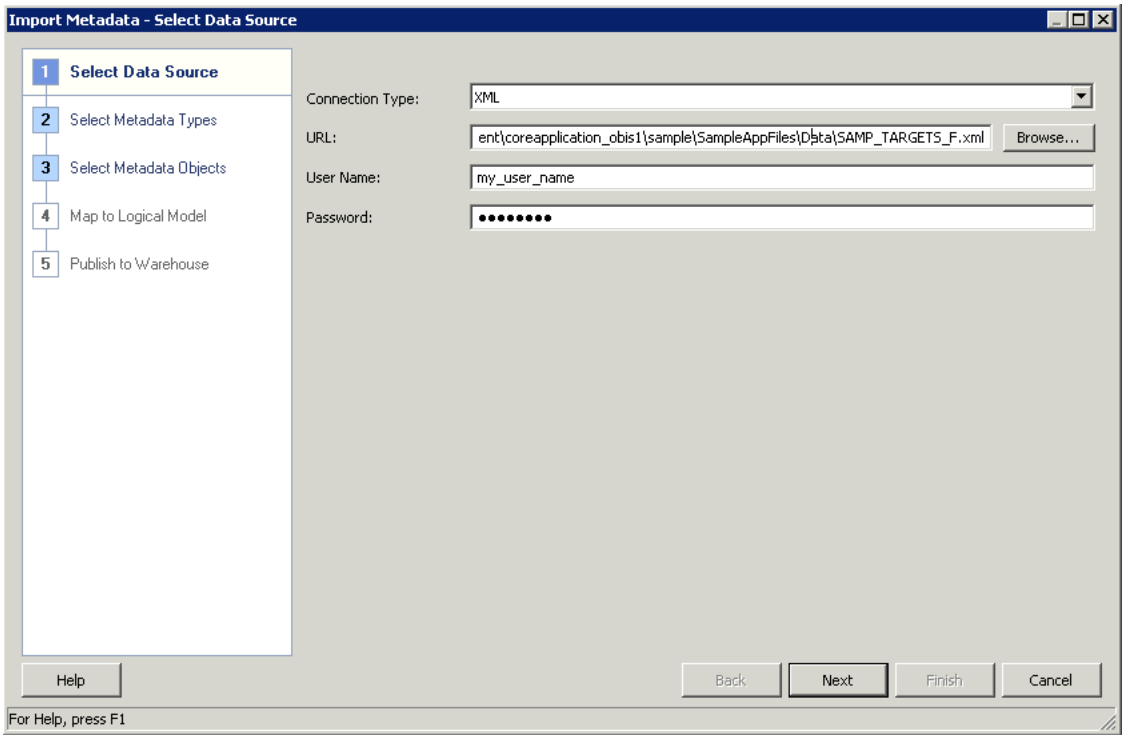
To import XML data using the Oracle BI Server XML Gateway:

1. In the Administration Tool, select **File**, then select **Import Metadata**. The Import Metadata Wizard appears.

   Note: If you have already defined an existing database and connection pool, you can right-click the connection pool in the Physical layer and select **Import Metadata**. The Import Metadata Wizard appears with the information on the Select Data Source screen pre-filled.

   Figure 5–3 shows the Import Metadata Wizard.

Figure 5–3 Import Metadata Wizard: XML Data Source



Note that the Map to Logical Model and Publish to Warehouse screens are available only for ADF data sources. "About the Map to Logical Model and Publish to Warehouse Screens" for more information.

2. In the Select Data Source screen, select **XML** for **Connection Type**. Then, provide the following values:

   - For **URL**, specify the XML data source URL. The Oracle BI Server XML Gateway supports all data sources described in the previous section.

     URLs can include repository or session variables. For more information about variables, see Chapter 19.

     If you click **Browse**, the Select XML File dialog appears, from which you can select a single file. For XML documents, the file name in the URL that you specify must have the suffix `.xml`. Otherwise, the documents are treated as HTML documents.

   - Type an optional user name and password in the appropriate fields for connections to HTTP sites that employ the HTTP Basic Authentication security mode.

     In addition to HTTP Basic Authentication security mode, the Oracle BI Server XML Gateway also supports Secure HTTP protocol and Integrated Windows Authentication (for Windows 2000), formerly called NTLM or Windows NT Challenge/Response authentication.

   When you have finished providing information in the Select Data Source screen, click **Next**. The Select Metadata Types screen appears.

3. Select the options for the types of objects that you want to import (for example, **Tables**, **Keys**, and **Foreign Keys**). The most typical options are automatically selected.

If you want to import joins, select both **Keys** and **Foreign Keys**. If you want to import system tables, you must have the system privilege for your data source.

4. Click **Next**. The Select Metadata Objects screen appears.
5. Select the objects you want to import in the **Available** list and move them to the **Selected** list, using the **>** (Import selected) or **>>** (Import all) buttons. You can also move objects from the **Selected** list back to the **Available** list, using the **<** (Remove selected) and **<<** (Remove all) buttons.

   To search for a particular item, enter a keyword in the **Find** box and then click **Find Down** or **Find Up**.

   Select **Show complete structure** to view all objects, including those that have already been imported. Deselecting this option shows only the objects that are available for import. When this option is selected, objects that have already been imported appear grayed out.
6. Click **Finish**.

After you import XML data, you must adjust connection pool settings. See "Creating or Changing Connection Pools" for complete information. Minimally, you can do the following:

- In the Connection Pool dialog, type a name and optional description for the connection on the General tab.
- Click the XML tab to set additional connection properties, including the URL refresh interval and the length of time to wait for a URL to load before timing out.

  Because XML data sources are typically updated frequently and in real time, you can specify a refresh interval for Oracle BI Server XML Gateway data sources. The default timeout interval for queries (URL loading time-out) is 15 minutes. For more information, see "About the Refresh Interval for XML Data Sources" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

Examples of XML Documents Generated by the Oracle BI Server XML Gateway

The examples in this section show sample XML documents and the corresponding columns that are generated by the Oracle BI Server XML Gateway.

Example 5–1 XML Schema Contained in an External File

The following sample XML data document (mytest.xml) references an XML schema contained in an external file. The schema file is shown following the data document. The generated XML schema information available for import to the repository is shown at the end.

```
<?xml version="1.0"?>
<test xmlns="x-schema:mytest_sch.xml">

<row>
<p1>0</p1>
<p2 width="5">
      <p3>hi</p3>
      <p4>
         <p6>xx0</p6>
         <p7>yy0</p7>
      </p4>
      <p5>zz0</p5>
</p2>
```

```
</row>

<row>
<p1>1</p1>
<p2 width="6">
      <p3>how are you</p3>
      <p4>
         <p6>xx1</p6>
         <p7>yy1</p7>
      </p4>
      <p5>zz1</p5>
</p2>
</row>

<row>
<p1>a</p1>
<p2 width="7">
      <p3>hi</p3>
      <p4>
         <p6>xx2</p6>
         <p7>yy2</p7>
      </p4>
      <p5>zz2</p5>
</p2>
</row>

<row>
<p1>b</p1>
<p2 width="8">
      <p3>how are they</p3>
      <p4>
         <p6>xx3</p6>
         <p7>yy3</p7>
      </p4>
      <p5>zz2</p5>
</p2>
</row>
</test>
```

The corresponding schema file follows:

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
      xmlns:dt="urn:schemas-microsoft-com:datatypes">
      <ElementType name="test" content="eltOnly" order="many">
         <element type="row"/>
      </ElementType>
      <ElementType name="row" content="eltOnly" order="many">
      <element type="p1"/>
         <element type="p2"/>
      </ElementType>
      <ElementType name="p2" content="eltOnly" order="many">
         <AttributeType name="width" dt:type="int" />
         <attribute type="width" />
         <element type="p3"/>
         <element type="p4"/>
         <element type="p5"/>
      </ElementType>
      <ElementType name="p4" content="eltOnly" order="many">
         <element type="p6"/>
         <element type="p7"/>
      </ElementType>
```

```
      <ElementType name="p1" content="textOnly" dt:type="string"/>
      <ElementType name="p3" content="textOnly" dt:type="string"/>
      <ElementType name="p5" content="textOnly" dt:type="string"/>
      <ElementType name="p6" content="textOnly" dt:type="string"/>
      <ElementType name="p7" content="textOnly" dt:type="string"/>
</Schema>
```

The name of the table generated from the preceding XML data document (mytest.xml) would be `mytest` and the column names would be `p1`, `p3`, `p6`, `p7`, `p5`, and `width`.

In addition, to preserve the context in which each column occurs in the document and to distinguish between columns derived from XML elements with identical names but appearing in different contexts, a list of fully qualified column names is generated, based on the XPath proposal of the World Wide Web Consortium, as follows:

```
//test/row/p1
//test/row/p2/p3
//test/row/p2/p4/p6
//test/row/p2/p4/p7
//test/row/p2/p5
//test/row/p2@width
```

Example 5–2 Nested Table Structures in an XML Document

The following example is a more complex example that demonstrates the use of nested table structures in an XML document. You can optionally omit references to an external schema file, in which case all elements are treated as being of the Varchar character type.

```
===Invoice.xml===
<INVOICE>
   <CUSTOMER>
      <CUST_ID>1</CUST_ID>
      <FIRST_NAME>Nancy</FIRST_NAME>
      <LAST_NAME>Fuller</LAST_NAME>
      <ADDRESS>
         <ADD1>507 - 20th Ave. E.,</ADD1>
         <ADD2>Apt. 2A</ADD2>
         <CITY>Seattle</CITY>
         <STATE>WA</STATE>
         <ZIP>98122</ZIP>
      </ADDRESS>
      <PRODUCTS>
           <CATEGORY>
              <CATEGORY_ID>CAT1</CATEGORY_ID>
              <CATEGORY_NAME>NAME1</CATEGORY_NAME>
              <ITEMS>
                  <ITEM>
                     <ITEM_ID>1</ITEM_ID>
                     <NAME></NAME>
                     <PRICE>0.50</PRICE>
                     <QTY>2000</QTY>
                  </ITEM>
                  <ITEM>
                     <ITEM_ID>2</ITEM_ID>
                     <NAME>SPRITE</NAME>
                     <PRICE>0.30</PRICE>
                     <QTY></QTY>
                  </ITEM>
              </ITEMS>
          </CATEGORY>
```

```
        <CATEGORY>
           <CATEGORY_ID>CAT2</CATEGORY_ID>
           <CATEGORY_NAME>NAME2</CATEGORY_NAME>
           <ITEMS>
               <ITEM>
                  <ITEM_ID>11</ITEM_ID>
                  <NAME>ACOKE</NAME>
                  <PRICE>1.50</PRICE>
                  <QTY>3000</QTY>
               </ITEM>
               <ITEM>
                  <ITEM_ID>12</ITEM_ID>
                  <NAME>SOME SPRITE</NAME>
                  <PRICE>3.30</PRICE>
                  <QTY>2000</QTY>
               </ITEM>
           </ITEMS>
      </CATEGORY>
   </PRODUCTS>
</CUSTOMER>
<CUSTOMER>
   <CUST_ID>2</CUST_ID>
   <FIRST_NAME>Andrew</FIRST_NAME>
   <LAST_NAME>Carnegie</LAST_NAME>
   <ADDRESS>
      <ADD1>2955 Campus Dr.</ADD1>
      <ADD2>Ste. 300</ADD2>
      <CITY>San Mateo</CITY>
      <STATE>CA</STATE>
      <ZIP>94403</ZIP>
   </ADDRESS>
   <PRODUCTS>
        <CATEGORY>
           <CATEGORY_ID>CAT22</CATEGORY_ID>
           <CATEGORY_NAME>NAMEA1</CATEGORY_NAME>
           <ITEMS>
               <ITEM>
                  <ITEM_ID>122</ITEM_ID>
                  <NAME>DDDCOKE</NAME>
                  <PRICE>11.50</PRICE>
                  <QTY>2</QTY>
               </ITEM>
               <ITEM>
                  <ITEM_ID>22</ITEM_ID>
                  <NAME>PSPRITE</NAME>
                  <PRICE>9.30</PRICE>
                  <QTY>1978</QTY>
               </ITEM>
           </ITEMS>
      </CATEGORY>
        <CATEGORY>
           <CATEGORY_ID>CAT24</CATEGORY_ID>
           <CATEGORY_NAME>NAMEA2</CATEGORY_NAME>
           <ITEMS>
               <ITEM>
                  <ITEM_ID>19</ITEM_ID>
                  <NAME>SOME COKE</NAME>
                  <PRICE>1.58</PRICE>
                  <QTY>3</QTY>
               </ITEM>
```

```
                <ITEM>
                   <ITEM_ID>15</ITEM_ID>
                   <NAME>DIET SPRITE</NAME>
                   <PRICE>9.30</PRICE>
                   <QTY>12000</QTY>
                </ITEM>
             </ITEMS>
          </CATEGORY>
       </PRODUCTS>
    </CUSTOMER>
    <CUSTOMER>
       <CUST_ID>3</CUST_ID>
       <FIRST_NAME>Margaret</FIRST_NAME>
       <LAST_NAME>Leverling</LAST_NAME>
       <ADDRESS>
          <ADD1>722 Moss Bay Blvd.</ADD1>
          <ADD2> </ADD2>
          <CITY>Kirkland</CITY>
          <STATE>WA</STATE>
          <ZIP>98033</ZIP>
       </ADDRESS>
       <PRODUCTS>
            <CATEGORY>
               <CATEGORY_ID>CAT31</CATEGORY_ID>
               <CATEGORY_NAME>NAMEA3</CATEGORY_NAME>
               <ITEMS>
                   <ITEM>
                      <ITEM_ID>13</ITEM_ID>
                      <NAME>COKE33</NAME>
                      <PRICE>30.50</PRICE>
                      <QTY>20033</QTY>
                   </ITEM>
                   <ITEM>
                      <ITEM_ID>23</ITEM_ID>
                      <NAME>SPRITE33</NAME>
                      <PRICE>0.38</PRICE>
                      <QTY>20099</QTY>
                   </ITEM>
               </ITEMS>
          </CATEGORY>
            <CATEGORY>
               <CATEGORY_ID>CAT288</CATEGORY_ID>
               <CATEGORY_NAME>NAME H</CATEGORY_NAME>
               <ITEMS>
                   <ITEM>
                      <ITEM_ID>19</ITEM_ID>
                      <NAME>COLA</NAME>
                      <PRICE>1.0</PRICE>
                      <QTY>3</QTY>
                   </ITEM>
                   <ITEM>
                      <ITEM_ID>18</ITEM_ID>
                      <NAME>MY SPRITE</NAME>
                      <PRICE>8.30</PRICE>
                      <QTY>123</QTY>
                   </ITEM>
               </ITEMS>
          </CATEGORY>
       </PRODUCTS>
    </CUSTOMER>
```

```
</INVOICE>
```

The generated XML schema consists of one table (INVOICE) with the following column names and their corresponding fully qualified names.

Column	Fully Qualified Name
ADD1	//INVOICE/CUSTOMER/ADDRESS/ADD1
ADD2	//INVOICE/CUSTOMER/ADDRESS/ADD2
CITY	//INVOICE/CUSTOMER/ADDRESS/CITY
STATE	//INVOICE/CUSTOMER/ADDRESS/STATE
ZIP	//INVOICE/CUSTOMER/ADDRESS/ZIP
CUST_ID	//INVOICE/CUSTOMER/CUST_ID
FIRST_NAME	//INVOICE/CUSTOMER/FIRST_NAME
LAST_NAME	//INVOICE/CUSTOMER/LAST_NAME
CATEGORY_ID	//INVOICE/CUSTOMER/PRODUCTS/CATEGORY/CATEGORY_ID
CATEGORY_NAME	//INVOICE/CUSTOMER/PRODUCTS/CATEGORY/CATEGORY_NAME
ITEM_ID	//INVOICE/CUSTOMER/PRODUCTS/CATEGORY/ITEMS/ITEM/ITEM_ID
NAME	//INVOICE/CUSTOMER/PRODUCTS/CATEGORY/ITEMS/ITEM/NAME
PRICE	//INVOICE/CUSTOMER/PRODUCTS/CATEGORY/ITEMS/ITEM/PRICE
QTY	//INVOICE/CUSTOMER/PRODUCTS/CATEGORY/ITEMS/ITEM/QTY

Only tags with values are extracted as columns. An XML query generates fully qualified tag names, to help ensure appropriate columns are retrieved.

The following shows the results of a sample query against the INVOICE table:

```
SELECT first_name, last_name, price, qty, name FROM invoice
-------------------------------------------------------------
FIRST_NAME  LAST_NAME         PRICE   QTY    NAME
-------------------------------------------------------------
Andrew      Carnegie           1.58     3    SOME COKE
Andrew      Carnegie          11.50     2    DDDCOKE
Andrew      Carnegie           9.30   12000  DIET SPRITE
Andrew      Carnegie           9.30    1978  PSPRITE
Margar      Leverling          0.38   20099  SPRITE33
Margar      Leverling          1.0      3    COLA
Margar      Leverling         30.50   20033  COKE33
Margar      Leverling          8.30    123   MY SPRITE
Nancy       Fuller             0.30          SPRITE
Nancy       Fuller             0.50    2000
Nancy       Fuller             1.50    3000  ACOKE
Nancy       Fuller             3.30    2000  SOME SPRITE
-------------------------------------------------------------
Row count: 12
```

About Using HTML Tables as a Data Source

The Oracle BI Server XML Gateway also supports the use of tables in HTML files as a data source. The HTML file can be identified as a URL pointing to a file on the internet (including intranet or extranet) or as a file on a local or network drive.

Even though tables, defined by the <table> and </table> tag pair, are native constructs of the HTML 4.0 specification, they are often used by Web designers as a general formatting device to achieve specific visual effects rather than as a data structure. The Oracle BI Server XML Gateway is currently the most effective in extracting tables that include specific column headers, defined by <th> and </th> tag pairs.

For tables that do not contain specific column headers, the Oracle BI Server XML Gateway employs some simple heuristics to make a best effort to determine the portions of an HTML file that appear to be genuine data tables.

The following is a sample HTML file with one table.

```
<html>
   <body>
      <table border=1 cellpadding=2 cellspacing=0>
         <tr>
            <th colspan=1>Transaction</th>
            <th colspan=2>Measurements</th>
         </tr>
         <tr>
            <th>Quality</th>
            <th>Count</th>
            <th>Percent</th>
         </tr>
         <tr>
            <td>Failed</td>
            <td>66,672</td>
            <td>4.1%</td>
         </tr>
         <tr>
            <td>Poor</td>
            <td>126,304</td>
            <td>7.7%</td>
         </tr>
         <tr>
            <td>Warning</td>
            <td>355,728</td>
            <td>21.6%</td>
         </tr>
         <tr>
            <td>OK</td>
            <td>1,095,056</td>
            <td>66.6%</td>
         </tr>
         <tr>
            <td colspan=1>Grand Total</td>
            <td>1,643,760</td>
            <td>100.0%</td>
         </tr>
      </table>
   </body>
</html>
```

The table name is derived from the HTML filename, and the column names are formed by concatenating the headings (defined by the <th> and </th> tag pairs) for the corresponding columns, separated by an underscore.

Assuming that our sample file is named 18.htm, the table name would be `18_0` (because it is the first table in that HTML file), with the following column names and their corresponding fully qualified names:

Column	Fully Qualified Name
Transaction_Quality	\\18_0\Transaction_Quality
Measurements_Count	\\18_0\Measurements_Count
Measurements_Percent	\\18_0\Measurements_Percent

If the table column headings appear in more than one row, the column names are formed by concatenating the corresponding field contents of those header rows.

For tables without any heading tag pairs, the Oracle BI Server XML Gateway assumes the field values (as delimited by the <td> and </td> tag pairs) in the first row to be the column names. The columns are named by the order in which they appear (c0, c1, and so on).

See "Importing Metadata from XML Data Sources Using XML ODBC" and "Examples of XML Documents" for additional XML examples.

Importing Metadata from XML Data Sources Using XML ODBC

Using the XML ODBC database type, you can access XML data sources through an ODBC interface. The data types of the XML elements representing physical columns in physical tables are derived from the data types of the XML elements as defined in the XML schema.

In the absence of a proper XML schema, the default data type of string is used. Data Type settings in the Physical layer do not override those defined in the XML data sources. When accessing XML data without XML schema, use the `CAST` operator to perform data type conversions in the Business Model and Mapping layer of the Administration Tool.

If you are importing metadata into an existing database in the Physical layer, then confirm that the COUNT_STAR_SUPPORTED option is selected in the Features tab of the Database properties dialog. If you import metadata without the COUNT_STAR_SUPPORTED option selected, then the **Update Row Count** option will not display in the right-click menu for the database's physical tables.

To import XML data using ODBC:

1. To access XML data sources through ODBC, you first need to license and install an XML ODBC driver.
2. Create ODBC DSNs that point to the XML data sources you want to access, making sure you select the XML ODBC database type.
3. In the Administration Tool, select **File**, then select **Import Metadata**.
4. Follow the instructions in the dialogs to import the ODBC DSNs into the repository. See "Importing Metadata from Relational Data Sources" for more information.

Caution: Due to XML ODBC limitations, you must select the **Synonyms** option in the Select Metadata Types screen, or no tables are imported.

Example of an XML ODBC Data Source

The example in this section shows an XML ODBC data source in the Microsoft ADO persisted file format. Both the data and the schema could be contained inside the same document.

Example 5–3 XML ODBC Example

```
<xml xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882'
  xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14882'
  xmlns:rs='urn:schemas-microsoft-com:rowset'
  xmlns:z='#RowsetSchema'>
<s:Schema id='RowsetSchema'>
  <s:ElementType name='row' content='eltOnly' rs:CommandTimeout='30'
  rs:updatable='true'>
    <s:AttributeType name='ShipperID' rs:number='1' rs:writeunknown='true'
    rs:basecatalog='Paint' rs:basetable='Shippers' rs:basecolumn='ShipperID'>
      <s:datatype dt:type='i2' dt:maxLength='2' rs:precision='5'
      rs:fixedlength='true' rs:benull='false'/>
    </s:AttributeType>
    <s:AttributeType name='CompanyName' rs:number='2' rs:writeunknown='true'
    rs:basecatalog='Paint' rs:basetable='Shippers' rs:basecolumn='CompanyName'>
      <s:datatype dt:type='string' rs:dbtype='str' dt:maxLength='40'
      rs:benull='false'/>
    </s:AttributeType>
    <s:AttributeType name='Phone' rs:number='3' rs:nullable='true'
    rs:writeunknown='true' rs:basecatalog='Paint' rs:basetable='Shippers'
    rs:basecolumn='Phone'>
      <s:datatype dt:type='string' rs:dbtype='str' dt:maxLength='24'
      rs:fixedlength='true'/>
    </s:AttributeType>
    <s:extends type='rs:rowbase'/>
  </s:ElementType>
</s:Schema>
<rs:data>
  <z:row ShipperID='1' CompanyName='Speedy Express' Phone='(503)
  555-9831          '/>
  <z:row ShipperID='2' CompanyName='United Package' Phone='(503)
  555-3199          '/>
  <z:row ShipperID='3' CompanyName='Federal Shipping' Phone='(503)
  555-9931          '/>
</rs:data>
</xml>
```

Examples of XML Documents

This section contains XML documents that provide examples of several different situations and explains how the Oracle BI Server XML access method handles those situations.

- The XML documents **83.xml** and **8_sch.xml** (shown in Example 5–4 and Example 5–5) demonstrate the use of the same element declarations in different scope. For example, <p3> could appear within <p2> as well as within <p4>.

Because the element <p3> in the preceding examples appears in two different scopes, each element is given a distinct column name by appending an index number to the second occurrence of the element during the import process. In this case, the second occurrence becomes p3_1. If <p3> occurs in additional contexts, they become p3_2, p3_3.

- The XML documents **83.xml** and **84.xml** (shown in Example 5–4 and Example 5–6) demonstrate that multiple XML files can share the same schema (8_sch.xml).
- Internet Explorer version 5 and higher supports HTML documents containing embedded XML fragments called XML islands. The XML document **island2.htm** (shown in Example 5–7) demonstrates a simple situation where multiple XML data islands, and therefore multiple tables, could be generated from one document. One table is generated for each instance of an XML island. Tables are distinguished by appending an appropriate index to the document name. For island2.htm, the two XML tables generated would be island2_0 and island2_1.

Example 5–4 83.xml

```
===83.xml===
<?xml version="1.0"?>
<test xmlns="x-schema:8_sch.xml">
<row>
<p1>0</p1>
<p2 width="5" height="2">
   <p3>hi</p3>
   <p4>
      <p3>hi</p3>
      <p6>xx0</p6>
      <p7>yy0</p7>
   </p4>
   <p5>zz0</p5>
</p2>
</row>

<row>
<p1>1</p1>
<p2 width="6" height="3">
   <p3>how are you</p3>
   <p4>
      <p3>hi</p3>
      <p6>xx1</p6>
      <p7>yy1</p7>
   </p4>
   <p5>zz1</p5>
</p2>
</row>
</test>
```

Example 5–5 8_sch.xml

```
===8_sch.xml===
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
         <AttributeType name="height" dt:type="int" />
   <ElementType name="test" content="eltOnly" order="many">
      <AttributeType name="height" dt:type="int" />
      <element type="row"/>
   </ElementType>
   <ElementType name="row" content="eltOnly" order="many">
```

```
         <element type="p1"/>
      <element type="p2"/>
   </ElementType>
   <ElementType name="p2" content="eltOnly" order="many">
         <AttributeType name="width" dt:type="int" />
      <AttributeType name="height" dt:type="int" />
         <attribute type="width" />
      <attribute type="height" />
      <element type="p3"/>
      <element type="p4"/>
      <element type="p5"/>
   </ElementType>
   <ElementType name="p4" content="eltOnly" order="many">
      <element type="p3"/>
      <element type="p6"/>
      <element type="p7"/>
   </ElementType>
   <ElementType name="test0" content="eltOnly" order="many">
      <element type="row"/>
   </ElementType>
      <ElementType name="p1" content="textOnly" dt:type="string"/>
      <ElementType name="p3" content="textOnly" dt:type="string"/>
      <ElementType name="p5" content="textOnly" dt:type="string"/>
      <ElementType name="p6" content="textOnly" dt:type="string"/>
      <ElementType name="p7" content="textOnly" dt:type="string"/>
</Schema>
```

Example 5–6 84.xml

```
===84.xml===
<?xml version="1.0"?>
<test0 xmlns="x-schema:8_sch.xml">
<row>
<p1>0</p1>
<p2 width="5" height="2">
   <p3>hi</p3>
   <p4>
      <p3>hi</p3>
      <p6>xx0</p6>
      <p7>yy0</p7>
   </p4>
   <p5>zz0</p5>
</p2>
</row>

<row>
<p1>1</p1>
<p2 width="6" height="3">
   <p3>how are you</p3>
   <p4>
      <p3>hi</p3>
      <p6>xx1</p6>
      <p7>yy1</p7>
   </p4>
   <p5>zz1</p5>
</p2>
</row>
</test0>
```

Example 5–7 Island2.htm

```
===island2.htm===
<HTML>
   <HEAD>
<TITLE>HTML Document with Data Island</TITLE>
</HEAD>
   <BODY>
<p>This is an example of an XML data island in I.E. 5</p>
   <XML ID="12345">
   test>
      <row>
         <field1>00</field1>
         <field2>01</field2>
   </row>
      <row>
         <field1>10</field1>
         <field2>11</field2>
   </row>
      <row>
         <field1>20</field1>
         <field2>21</field2>
      </row>
   </test>
</XML>
<p>End of first example.</p>
<XML ID="12346">
   <test>
      <row>
         <field11>00</field11>
         <field12>01</field12>
      </row>
      <row>
         <field11>10</field11>
         <field12>11</field12>
      </row>
      <row>
         <field11>20</field11>
         <field12>21</field12>
      </row>
   </test>
</XML>
<p>End of second example.</p>
</BODY>
</HTML>
```

About Using a Standby Database with Oracle Business Intelligence

A standby database is used mainly for its high availability and failover functions as a backup for the primary database. In a standby database configuration, there is regularly scheduled replication from the primary database to the secondary database. The latency of this replication must be short enough that writing to the primary database while reading from the secondary database does not cause any synchronization or data integrity problems.

Because a standby database is essentially a read-only database, it can be used as a business intelligence query server, relieving the workload of the primary database and improving query performance.

The following sections explain how to use a standby database with Oracle Business Intelligence:

Configuring a Standby Database with Oracle Business Intelligence

In a standby database configuration, you have two databases: a primary database that handles all write operations and is the source of truth for data integrity, and a secondary database that is exposed as a read-only source. When you use a standby database configuration with Oracle Business Intelligence, all write operations are offloaded to the primary database, and read operations are sent to the standby database.

Write operations that need to be routed to the primary source may include the following:

- Oracle BI Scheduler job and instance data
- Temporary tables for performance enhancements
- Writeback scripts for aggregate persistence
- Usage tracking data, if usage tracking has been enabled
- Event polling table data, if event polling tables are being used

The following list provides an overview of how to configure the Oracle BI Server to use a standby database.

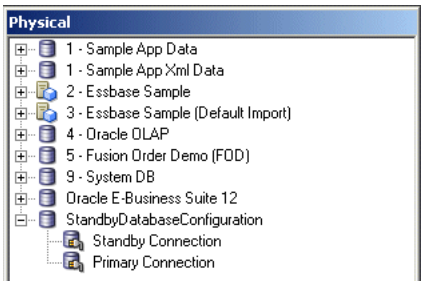
To configure the Oracle BI Server to use a standby database:

1. Create a single database object for the standby database configuration, with temporary table creation disabled.
2. Configure two connection pools for the database object:
   - A read-only connection pool that points to the standby database
   - A second connection pool that points to the primary database for write operations
3. Update any connection scripts that write to the database so that they explicitly specify the primary database connection pool.
4. If usage tracking has been enabled, update the usage tracking configuration to use the primary connection.
5. If event polling tables are being used, update the event polling database configuration to use the primary connection.
6. Ensure that Oracle BI Scheduler is not configured to use any standby sources.

Even though there are two separate physical data sources for the standby database configuration, you create only one database object in the Physical layer. Figure 5–4

shows the database object and connection pools for the standby database configuration in the Physical layer.

Figure 5–4 Standby Database Configuration in the Physical Layer



Creating the Database Object for the Standby Database Configuration

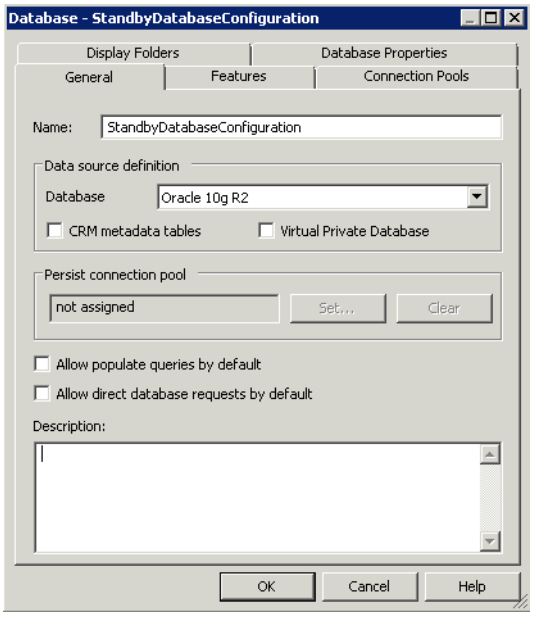
Use the Administration Tool to create a database object in the repository for the standby database configuration. When you create the database object, make sure that the persist connection pool is not assigned, to prevent the Oracle BI Server from creating temporary tables in the standby database.

To create a database object for the standby database configuration:

1. In the Administration Tool, right-click the Physical layer and select **New Database**.
2. Provide a name for the database, and then select the type of database in the Database list.
3. Ensure that the **Persist connection pool** is "not assigned."

Figure 5–5 shows the Database dialog for a standby database configuration.

Figure 5–5 Database Dialog for Standby Database Configuration



Creating Connection Pools for the Standby Database Configuration

After you have created a database object in the repository for the standby database configuration, use the Administration Tool to create two connection pools: one that points to the standby database, and another that points to the primary database.

Because the standby connection pool is used for the majority of connections, make sure that the standby connection pool is listed first.

Note: Connection pools are used in the order listed, until the maximum number of connections is achieved. Ensure that the maximum number of connections is set in accordance with the standby database tuning.

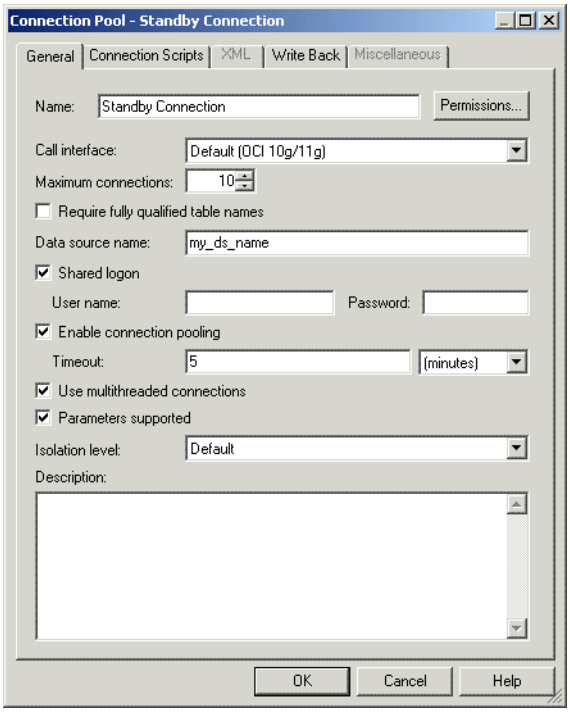
See "Creating or Changing Connection Pools" for more information about setting the maximum number of connections.

To create a standby connection pool for the standby database configuration:

1. In the Administration Tool, in the Physical layer, right-click the database object for the standby database configuration and select **New Object**, then select **Connection Pool**.
2. Provide a name for the connection pool, and ensure that the call interface is appropriate for the standby database type.
3. Provide the **Data source name** for the standby database.
4. Enter a user name and password for the standby database.
5. Click **OK**.

Figure 5–6 shows the Connection Pool dialog for the standby connection pool.

Figure 5–6 Connection Pool Dialog for Standby Connection Pool

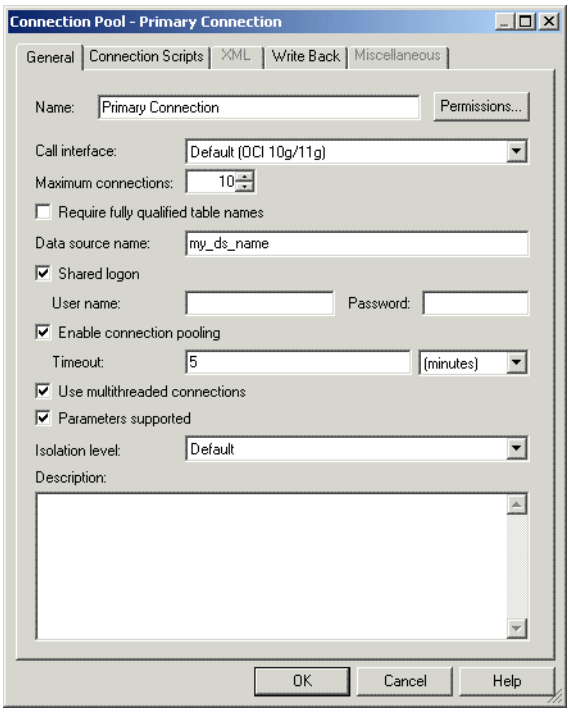


To create the primary connection pool for the standby database configuration:

1. In the Administration Tool, in the Physical layer, right-click the database object for the standby database configuration and select **New Object**, then select **Connection Pool.**
2. Provide a name for the connection pool, and ensure that the call interface is appropriate for the primary database type.
3. Provide the **Data source name** for the primary database.
4. Enter a user name and password for the primary database.
5. Click **OK**.

Figure 5–7 shows the Connection Pool dialog for the primary connection pool.

Figure 5–7 Connection Pool Dialog for Primary Connection Pool



Updating Write-Back Scripts in a Standby Database Configuration

If you use scripts that write to the database, such as scripts for aggregate persistence, you must update the scripts to explicitly refer to the primary connection pool. Information written through the primary connection will be automatically transferred to the standby database (through the regularly scheduled replication between the primary and secondary databases), and will become available through the standby connection pool.

The following example shows a writeback script for aggregate persistence that explicitly specifies the primary connection pool:

```
create aggregates sc_rev_qty_yr_cat for "DimSnowflakeSales"."SalesFacts"
("Revenue", "QtySold") at levels ("DimSnowflakeSales"."Time"."Year",
"DimSnowflakeSales"."Product"."Category") using connection pool
"StandbyDemo"."Primary Connection" in "StandbyDemo"."My_Schema"
```

Setting Up Usage Tracking in a Standby Database Configuration

The Oracle BI Server supports the collection of usage tracking data. When usage tracking is enabled, the Oracle BI Server collects usage tracking data for each query and writes statistics to a usage tracking log file or inserts them directly to a database table.

If you want to enable usage tracking on a standby database configuration using direct insertion, you must create the table used to store the usage tracking data (typically `S_NQ_ACCT`) on the primary database. Then, import the table into the physical layer of the repository using the Administration Tool.

You must ensure that the database object for the usage tracking table is configured with both the standby connection pool and the primary connection pool. Then, ensure that the `CONNECTION_POOL` parameter for usage tracking points to the primary database. For example, in NQSConfig.ini:

```
CONNECTION_POOL = "StandbyDatabaseConfiguration"."Primary Connection";
```

See "Managing Usage Tracking" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for full information about usage tracking, including how to create tables for usage tracking data and how to set parameters for usage tracking.

Setting Up Event Polling in a Standby Database Configuration

You can use an Oracle BI Server event polling table (event table) as a way to notify the Oracle BI Server that one or more physical tables have been updated. The event table is a physical table that resides on a database accessible to the Oracle BI Server. It is normally exposed only in the Physical layer of the Administration Tool, where it is identified in the Physical Table dialog as an Oracle BI Server event table.

The Oracle BI Server requires write access to the event polling table. Because of this, if you are using event polling in a standby database configuration, you must ensure that the database object for the event table only references the primary connection pool.

See "Cache Event Processing with an Event Polling Table" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for full information about event polling, including how to set up, activate, and populate event tables.

Setting Up Oracle BI Scheduler in a Standby Database Configuration

Oracle BI Scheduler is an extensible application and server that manages and schedules jobs, both scripted and unscripted. To use Oracle BI Scheduler in a standby database configuration, you must ensure that the database object for Oracle BI Scheduler only references the primary connection pool.

See "Configuration Tasks for Oracle BI Scheduler" in *Oracle Fusion Middleware Scheduling Jobs Guide for Oracle Business Intelligence Enterprise Edition* for full information about setting up and using Oracle BI Scheduler.

6

Working with ADF Data Sources

This chapter describes how to set up Oracle ADF Business Components for use with Oracle Business Intelligence, and how to import metadata from ADF data sources.

Connecting to ADF data sources enables Oracle Business Intelligence users to query data from any application that is built using the ADF Framework. For example, because Oracle CRM applications are developed using the ADF Framework, Oracle Business Intelligence users can report on CRM data using an ADF data source that implements the required ADF Application Programming Interface (API).

By using the ADF components as a data source to the Oracle BI Server, users can quickly integrate operational reporting with any application that is built on top of the ADF Framework.

This chapter contains the following topics:

What Are ADF Business Components?

Oracle Application Development Framework (Oracle ADF) is an object-relational framework that can be used to create J2EE business services and expose underlying database objects. This framework provides an abstraction layer that enables application developers to build applications quickly and efficiently.

When you use Oracle ADF to build service-oriented Java EE applications, you implement your core business logic as one or more business services. These back-end services provide clients with a way to query, insert, update, and delete business data as required, while enforcing appropriate business rules. ADF Business Components are prebuilt application objects that provide a ready-to-use implementation of Java EE design patterns and best practices.

The ADF model is represented through the ADF Business Component constructs called Entity Objects and View Objects, usually constructed and defined during design time:

- **Entity Objects:** ADF framework components that represent a row in a database table and simplify modifying its data. Importantly, it enables you to encapsulate domain business logic for those rows to ensure your business policies and rules are consistently validated.
- **View Objects:** ADF framework components that encapsulate a SQL query and simplify working with its results. In addition to *read-only view objects*, there are *entity-based view objects* that support updatable rows. The view object queries just the data needed for the client-facing task at hand, then cooperates with one or more entity objects in your business domain layer to automatically validate and save changes made to its view rows. Like the read-only view object, an entity-based view object encapsulates a SQL query, can be linked into master/detail hierarchies using view links, and can be used in the data model of your application modules.

  Applications built using ADF obtain their data by querying the defined View Objects using the ADF APIs.

The ADF model also includes an *application module*, which is the transactional component that UI clients use to work with application data. It defines an updatable data model along with top-level procedures and functions (called service methods) related to a logical unit of work related to an end-user task.

The application module serves as a container for multiple View Objects and Entity Objects, and also contains configuration related to the JDBC data source.

About Operational Reporting with ADF Business Components

You can use Oracle Business Intelligence integration with ADF Business Components to generate reports on data within your applications.

For example, you can generate reports based on expense report data entered into an Expense Application. To do this, you would first import the Expense Application metadata into the Oracle BI repository using the Administration Tool, then map the data from the Physical layer to the Business Model and Mapping layer and Presentation layer. After you restart the Oracle BI Server and reload the metadata into Oracle BI Presentation Services, you can log in to Oracle BI Answers and drag and drop the columns to generate a report on the Expense Application data. For example, you can select columns to view a report of your expenses grouped by category.

About Importing ADF Business Components Into Oracle Business Intelligence

During import, the required physical tables and complex joins are automatically created. The instances (ViewObject and ViewLink) are imported into Oracle Business Intelligence. During query execution, the definitions retrieved from these instances are used to create the CompositeVO in ADF.

These complex joins are 'dummy joins' and are not executed in Oracle Business Intelligence. Instead, they denote ViewLink instances that connect pairs of View Objects in the ADF model. The physical table and complex join names correspond to the fully qualified ViewObject and ViewLink instance names, respectively. This convention allows arbitrary nesting of ApplicationModules in the ADF model.

Note that the **External Expression** field in the Complex Join dialog for ADF data sources is populated with the join condition defined in the view link.

The name of the automatically generated joins follow a naming convention similar to *ViewObjectName1_ViewObjectName2* (for example, AppModuleAM.AP_VO1_ AppModuleAM_BU_VO1). The ViewLink instance name appears in the **ViewLink Name** field of the Complex Join dialog.

The complex joins are only created automatically if a ViewLink instance is available. They are not created for ViewLink definitions. Joins using ViewLink definitions must be created manually. To do this, specify the ViewLink definition name in the **ViewLink Name** field of the Complex Join dialog.

Alternatively, Oracle Business Intelligence joins between view objects in different ApplicationModules are created upon import from ADF if custom properties are defined on the ApplicationModule. Note the following:

- The property name format is BI_VIEW_LINK_*property_name*
- The property value format is *source_view_object_instance_name*, *ViewLink_definition_name*, *destination_view_object_instance_name*

Be sure to use the fully qualified view object instance names for the source and destination view objects, as well as the fully qualified package name for the ViewLink definition.

About Specifying a SQL Bypass Database

The Oracle BI Server can automatically create composite View Objects at run time, so that an ad-hoc BI query can reference multiple View Objects in the ADF layer. For improved performance, a SQL bypass query is generated that incorporates the projection columns, filters, and joins required by the BI query.

The SQL Bypass feature directly queries the database so that aggregations and other transformations are pushed down where possible, reducing the amount of data streamed and worked on in Oracle Business Intelligence. When using a SQL Bypass database, the Oracle BI Server gets the view object query from the ADF data source and then wraps it with the aggregations in the Logical SQL query. The query, including the aggregations, is then executed in the database. Because the database computes the aggregation and fewer rows are streamed back to Oracle Business Intelligence, using a SQL Bypass database can result in significant performance gains.

Multiple View Objects are modeled as separate BI physical tables and are connected with dummy complex joins. These joins only represent the ViewLinks in the ADF model and are not executed by the Oracle BI Server.

You can specify the name of the SQL Bypass database in the connection pool for the ADF data source. The SQL Bypass database must be a physical database in the Physical layer of the repository. The database object for the SQL Bypass database must have a valid connection pool, with connection information that points to the same database that is being used by the JDBC Data source defined in the Oracle WebLogic Server that runs the ADF application.

The SQL Bypass database does not need to have any tables under it. After a valid database name is supplied, the SQL Bypass feature is enabled for all queries against that ADF database.

Setting Up ADF Data Sources

This section explains how to configure your ADF Business Components for use with Oracle Business Intelligence.

See "System Requirements and Certification" for information about supported versions.

This section contains the following topics:

- Creating a WebLogic Domain for ADF Business Components Used with Oracle Business Intelligence
- Deploying OBIEEBroker as a Shared Library in Oracle WebLogic Server
- Deploying the Application EAR File to Oracle WebLogic Server from JDeveloper
- Setting Up a JDBC Data Source in the WebLogic Server
- Setting the Logging Level for the Deployed Application in Oracle WebLogic Server

Creating a WebLogic Domain for ADF Business Components Used with Oracle Business Intelligence

To configure your ADF Business Components for use with Oracle Business Intelligence, you need to create a WebLogic Domain for your ADF Business Components that supports WebLogic Server, Oracle Application Core (Webapp), and Oracle JRF.

To create a WebLogic domain that supports the required components:

1. Start the WebLogic Configuration Wizard. For example, on Windows, run *MW_HOME*\wlserver_10.3\common\bin\config.cmd.
2. Select **Create a new WebLogic domain** and click **Next**.
3. On the Select Domain Source screen, ensure that **Basic WebLogic Server Domain**, **Oracle JRF**, and **Oracle Application Core (Webapp)** are selected.
4. Follow the remaining steps in the wizard, providing values appropriate for your environment.
5. Click **Create** on the Configuration Summary screen to create the domain.

You can start and stop the Oracle WebLogic Server for this domain using command-line scripts in the domain directory. For example, on Windows, use the following:

- *MW_HOME*\user_projects\domains*domain_name*\bin\startWebLogic.cmd
- *MW_HOME*\user_projects\domains*domain_name*\bin\stopWebLogic.cmd

Deploying OBIEEBroker as a Shared Library in Oracle WebLogic Server

To configure your ADF Business Components for use with Oracle Business Intelligence, you need to install OBIEEBroker (making its physical file or directory known to Oracle WebLogic Server) and start it. This process deploys the OBIEEBroker library as a shared library in Oracle WebLogic Server.

After the library has been installed and started, other deployed modules can reference the library. Note that the OBIEEBroker shared library is installed as part of your Oracle Business Intelligence installation.

To deploy OBIEEBroker as a shared library in Oracle WebLogic Server:

1. Ensure that Oracle WebLogic Server is running. If it is not running, start it. For example, on Windows, run *MW_HOME*\user_projects\domains*your_domain*\bin\startWebLogic.cmd.

2. Open the WebLogic Server Administration Console. For example, if your Oracle WebLogic Server is running locally on port 7001, go to `http://localhost:7001/console`.
3. Log in to the WebLogic Server Administration Console with the credentials you created when you set up your WebLogic domain.
4. In the Change Center, click **Lock & Edit**.
5. On the Home Page, in the left pane, click **Deployments**.
6. In the right pane, click **Install**.
7. Using the Install Application Assistant, locate the OBIEEBroker EAR file. You can find this file at:

   ORACLE_HOME`\bifoundation\javahost\lib\obisintegration\adf\oracle.bi.integration.adf.ear`

8. Click **Next**.
9. Select **Install this deployment as a library** and click **Next**.
10. Select the servers and/or clusters to which you want to deploy the OBIEEBroker library. Make sure to select all servers and clusters to which modules or applications that reference the library are deployed.
11. Click **Next**.
12. You can optionally update settings about the deployment. Typically, the default values are adequate. Click **Help** for more information.
13. Click **Next**, then click **Finish** to complete the installation.
14. In the Change Center, click **Activate Changes**.

Deploying the Application EAR File to Oracle WebLogic Server from JDeveloper

To configure your ADF Business Components for use with Oracle Business Intelligence, you need to deploy the application EAR file to Oracle WebLogic Server from JDeveloper.

Before beginning this procedure, ensure that the following conditions are true:

- You have an ADF Model project that contains ApplicationModules and view objects that will be exposed to Oracle Business Intelligence.
- You have deployed OBIEEBroker as a shared library in Oracle WebLogic Server. See "Deploying OBIEEBroker as a Shared Library in Oracle WebLogic Server" for more information.
- Oracle WebLogic Server is running.

To deploy the application EAR file to Oracle WebLogic Server from JDeveloper:

1. Start JDeveloper. For example, on Windows, run *MW_HOME*\jdeveloper\jdev\bin\jdev.exe.
2. Select **File**, then select **Open** to open the project that contains your ADF Business Components in JDeveloper. If prompted, allow JDeveloper to migrate the project to the latest version.
3. Create a new Application Module configuration, as follows:
   a. In the Model project, double-click the application module, then click the Configurations tab for that application module.

b. Create a new configuration with the following characteristics:

   – Select **JDBC DataSource** for **Connection Type**.

   – Keep the default **DataSource Name** (for example, java:comp/env/jdbc/ApplicationDBDS).

   When you set up the JDBC data source in Oracle WebLogic Server in a later step, you use part of this DataSource Name as the JNDI name required by Oracle WebLogic Server. The JNDI name is the DataSource Name without the java:comp/env context prefix (for example, jdbc/ApplicationDBDS).

4. Create a Business Component Archive deployment profile, as follows:

   a. In the Projects window, right-click the Model project and choose **New**.

   b. Select **Deployment Profiles** under **General** in the left pane, then choose **Business Components Archive** in the right pane and click **OK**.

   c. Provide a name for the deployment profile (for example, MyApplication_Archive) and click **OK**.

   d. On the Deployment page, click **OK**.

5. In the Projects window, right-click the Model project and select **Deploy > *your_deployment_profile_name* > Deploy**, or use the deployment wizard by selecting **Deploy to File**.

   After the project has been deployed, two jar files are created in the deploy directory for the Model project (for example, MyApplication_Archive_Common.jar and MyApplication_Archive_MiddleTier.jar).

6. Create a new Web Project for the application, as follows:

   a. Right-click the global application and select **New Project**.

   b. Select **Projects** from the left pane, then select **Web Project** from the right pane.

   c. Provide a project name (for example, **OBIEEBroker**).

   d. Click **Next** until you reach the **Web Project Profile** page.

   e. Modify the **Java EE Context Root** to a name that better represents your application (for example, MyApplication).

      This value determines the URL that you use to connect to the application from Oracle Business Intelligence (for example, http://localhost:7001/MyApplication/obieebroker).

7. Edit the Profile Dependencies of the WAR deployment, as follows:

   a. Right-click the Web Project you just created (for example, **OBIEEBroker**) and select **Project Properties**.

   b. From the left pane, select **Deployment**. Then, open the WAR File deployment profile on the right pane.

   c. Select **Profile Dependencies** from the left pane. Then, on the right pane, select the Common and MiddleTier deployment profiles of your Model project.

      Following this step ensures that the Business Component Archives for the Model project are included in the WAR file.

8. Expand the Web Project and open web.xml. Then, go to the source view of the file.

9. In the web.xml source, replace the content within the <web-app> element with the following:

```
<context-param>
  <description>This holds the Principals (CSV) that a valid end user should
have (at least one) in order to query the ADF layer from BI.</description>
  <param-name>oracle.bi.integration.approle.whitelist</param-name>
  <param-value>Application_Roles_List</param-value>
</context-param>

<filter>
  <filter-name>ServletADFFilter</filter-name>
  <filter-class>oracle.adf.share.http.ServletADFFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>ServletADFFilter</filter-name>
  <servlet-name>OBIEEBroker</servlet-name>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>

<servlet>
  <servlet-name>OBIEEBroker</servlet-name>
  <servlet-class>oracle.bi.integration.adf.v11g.obieebroker.OBIEEBroker
  </servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>OBIEEBroker</servlet-name>
  <url-pattern>/obieebroker</url-pattern>
</servlet-mapping>
```

Following this step ensures that the OBIEEBroker servlet will be used to access your application from Oracle Business Intelligence

For *application_roles_list*, provide a list of application roles in CSV form. For example:

```
<param-value>FBI_TRANSACTION_ANALYSIS_GENERIC_DUTY, OBIA_ANALYSIS_GENERIC_DUTY,
OBIA_EXTRACT_TRANSFORM_LOAD_DUTY, FUSION_APPS_BI_APPID</param-value>
```

If you provide a list of application roles, a user's application role is checked before access is allowed to the application. Note that this run-time check requires the following grant to be present in the *domain_name*/config/fmwconfig/system-jazn-data.xml file for the WebLogic domain:

```
<grant>
  <grantee>
    <codesource>
      <url>file:${domain.home}/servers/${weblogic.Name}/tmp/
      _WL_user/oracle.bi.integration.adf/-</url>
    </codesource>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>IdentityAssertion</name>
      <actions>execute</actions>
    </permission>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>AppSecurityContext.setApplicationID.obi</name>
    </permission>
```

```
  </permissions>
</grant>
```

If you do not want application roles to be checked by the OBIEEBroker servlet, use DISABLE_WHITELIST_ROLE_CHECK as the value for the <context-param> in web.xml. For example:

```
<param-value>DISABLE_BI_WHITELIST_ROLE_CHECK</param-value>
```

10. Create an EAR deployment profile for the application, as follows:
    a. Right-click the global application and select **Application Properties**.
    b. From the left pane, select **Deployment**, then click **New** on the right pane to create a new deployment profile.
    c. For **Archive Type**, select **EAR File**. Then, provide a name for the deployment profile (for example, MyApplication).

       The deployment profile name is used as the name displayed in the list of deployments in Oracle WebLogic Server.
    d. From the left pane, select **Application Assembly**. Then, on the right pane, select the **webapp** deployment profile of your Web Project.

       Following this step ensures that the WAR file from your Web Project is included in the EAR file.
11. Under **Application Resources**, select **Descriptors > META-INF > weblogic-application.xml**.
12. On the left, select the Libraries tab.
13. Create two new Shared Library References, as follows:
    - Create the first Shared Library Reference with the following characteristics:
      - **Library Name:** oracle.bi.integration.adf
      - **Implementation Version:** 11.1.1.2.0
    - Create the second Shared Library Reference with the following characteristics:
      - **Library Name:** oracle.applcore.model
      - **Implementation Version:** 11.1.1.0.0

    These two Shared Library References create the following entries in the weblogic-application.xml file for the application:

```
<library-ref>
  <library-name>oracle.bi.integration.adf</library-name>
  <implementation-version>11.1.1.2.0</implementation-version>
</library-ref>
<library-ref>
  <library-name>oracle.applcore.model</library-name>
  <implementation-version>11.1.1.0.0</implementation-version>
</library-ref>
```

14. Deploy the EAR file to Oracle WebLogic Server by right-clicking the global application, then selecting **Deploy > *EAR_deployment_profile_name***. From the dialog that appears, select **Deploy to Application Server** and then follow the instructions in the wizard.
15. To verify that the application has been deployed, log in to the WebLogic Server Administration Console and click **Deployments** under **Your Deployed Resources**.

Verify that your application appears in the list (for example, obieebroker_*app_name*).

Setting Up a JDBC Data Source in the WebLogic Server

To configure your ADF Business Components for use with Oracle Business Intelligence, you must set up a JDBC data source in Oracle WebLogic Server for your application.

To set up a JDBC data source in Oracle WebLogic Server:

1. Ensure that Oracle WebLogic Server is running. If it is not running, start it. For example, on Windows, run *MW_HOME*\user_projects\domains*your_domain*\bin\startWebLogic.cmd.
2. Open the WebLogic Server Administration Console. For example, if your Oracle WebLogic Server is running locally on port 7001, go to `http://localhost:7001/console`.
3. Log in to the WebLogic Server Administration Console with the credentials you created when you set up your WebLogic domain.
4. On the Home Page, select **JDBC**, then select **Data Sources**.
5. Click **New**.
6. Provide information for your data source. For **Name** and **JNDI Name**, provide the DataSource Name you specified in the Application Module configuration for the application, without the java:comp/env context prefix (for example, jdbc/ApplicationDBDS). In addition, make sure to select the target on which you want to deploy the data source before exiting the wizard.
7. Click **Finish** when you are done providing JDBC data source settings.

Setting the Logging Level for the Deployed Application in Oracle WebLogic Server

The log file for the server to which your application is deployed (*server_name*-diagnostic.log) records information about your deployed application.

You can find this file in the server-specific directory within your domain. For example, on Windows, the log file for the AdminServer is located in:

MW_HOME\user_projects\domains*your_domain*\servers\AdminServer\logs

To set the logging level for your deployed application:

1. Open the Oracle WebLogic Server file logging.xml for editing. You can find this file in:

   MW_HOME\user_projects\domains*your_domain*\config\fmwconfig\servers*server_name*

2. Within the `<loggers>` element, add the following child elements:

   ```
   <logger name="oracle.bi.integration.adf" level="LOG_LEVEL"/>
   <logger name="oracle.bi.integration.adf.v11g.obieebroker" level="LOG_LEVEL"/>
   ```

   Log levels include `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, and `FINEST`. Refer to the Oracle WebLogic Server documentation for information about logger levels.

3. Save and close the file.
4. Restart Oracle WebLogic Server.

Importing Metadata from ADF Data Sources

There are different ways to import metadata from ADF data sources into Oracle Business Intelligence.

Before you can import metadata from ADF sources, you must complete the steps in "Setting Up ADF Data Sources."

This section contains the following topics:

- Performing an Initial Import from ADF Data Sources
- Using Incremental Import to Propagate Flex Object Changes
- Automatically Mapping Flex Object Changes to the Logical Model
- Automatically Mapping Flex Object Changes Using the biserverextender Utility

Performing an Initial Import from ADF Data Sources

You can use the Import Metadata Wizard to perform an initial import from ADF data sources.

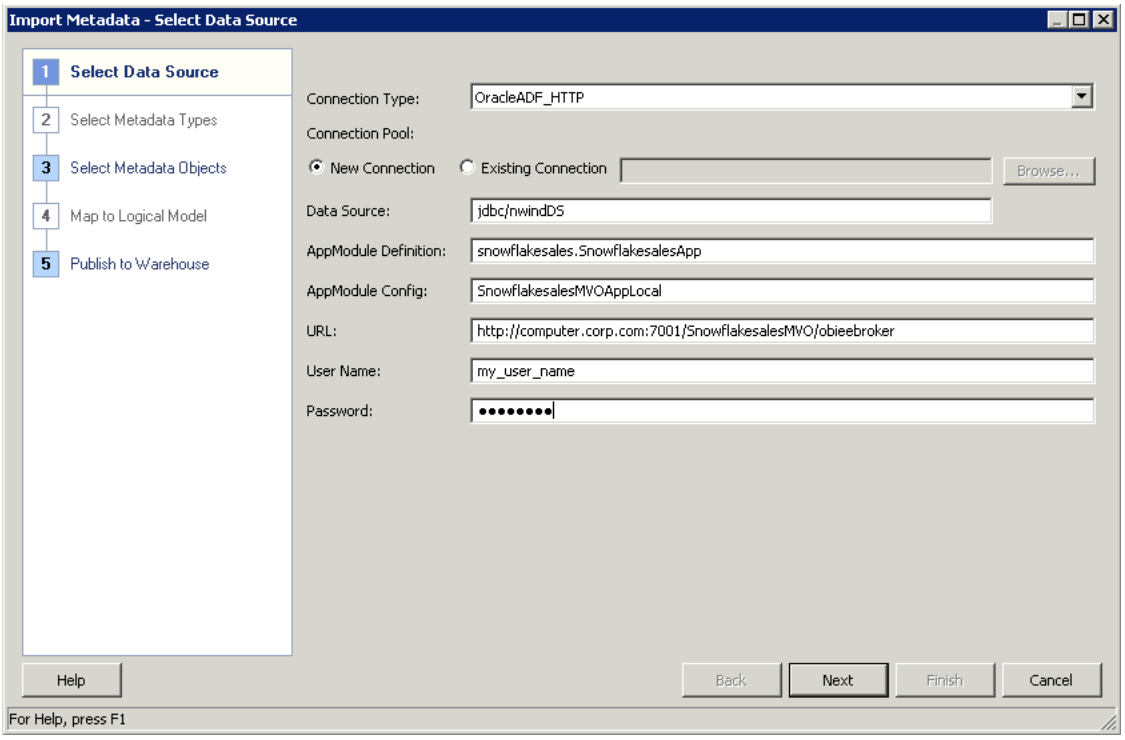
To import metadata from an ADF data source:

1. In the Administration Tool, select **File**, then select **Import Metadata**. The Import Metadata Wizard appears.

   Note: If you have already defined an existing ADF data source and connection pool, you can right-click the connection pool in the Physical layer and select **Import Metadata**. The Import Metadata Wizard appears with the information on the Select Data Source screen pre-filled.

   Figure 6–1 shows the Import Metadata Wizard.

Figure 6–1 Import Metadata Wizard: ADF Data Source



2. In the Select Data Source screen, select **OracleADF_HTTP** for **Connection Type**. Then, provide the following values:

   – Select **New Connection**, or select **Existing Connection** if you already have a connection pool for this data source. Click **Browse** to locate and select an existing connection pool. If you select **Existing Connection**, you do not provide information for **Data Source**, **AppModule Definition**, **AppModule Config**, or **URL**, and the **User Name** and **Password** fields are prefilled.

   – Keep the **Data Source** field blank to use the default JDBC data source configured in the application module. You only need to provide data source information (a JDBC data source name, such as jdbc/nWindORA05) if you want to use a different data source than the one set up in the application module.

   – For **AppModule Definition**, provide the fully qualified Java package name of the Root Application Module to which you want to connect, such as `oracle.apps.fii.receivables.model.RootAppModule`, or `snowflakesales.SnowflakeSalesApp`.

   – For **AppModule Config**, provide the name of the configuration you want to use in your connection, such as `RootAppModuleShared` or `SnowflakeSalesAppLocal`. See step 3 of "Deploying the Application EAR File to Oracle WebLogic Server from JDeveloper" for more information.

   – For **URL**, provide the URL to the Oracle Business Intelligence broker servlet, in the format:

     `http://`*`host`*`:`*`port`*`/`*`APP_DEPLOYMENT_NAME`*`/obieebroker`

     For example:

     `http://localhost:7001/MyApp/obieebroker`

The URL is case-sensitive.

– For **User Name** and **Password**, provide a valid user name and password for the Oracle ADF application. The user name and password must be set up and authenticated in the Oracle WebLogic Server security realm.

When you have finished providing information in the Select Data Source screen, click **Next**. The Select Metadata Objects screen appears.

3. Select the objects you want to import in the **Data source view** and move them to the **Repository View**, using the **>** (Import selected) or **>>** (Import all) buttons. You can also move objects from the **Repository View** back to the **Data source view**, using the **<** (Remove selected) and **<<** (Remove all) buttons.

   To search for a particular item, enter a keyword in the **Find** box and then click **Find Down** or **Find Up**.

   Select **Show complete structure** to view all objects, including those that have already been imported. Deselecting this option shows only the objects that are available for import. When this option is selected, objects that have already been imported appear grayed out.

   If this import is creating a new connection to the data source, when you move the items from the **Data source view** to the **Repository View** list, the Connection Pool dialog opens, showing the values that you provided in the Select Data Source screen of the Import Metadata Wizard. Optionally, click the Miscellaneous tab of the Connection Pool dialog and provide the name of a SQL Bypass database in the **SQL Bypass Database** field. Then, click **OK**. If you do not want to specify a SQL Bypass database, click **Cancel**.

   See "About Specifying a SQL Bypass Database" for more information.

4. Click **Finish** to close the wizard, or click **Next** to continue to the Map to Logical Model screen. See "Automatically Mapping Flex Object Changes to the Logical Model" for more information.

5. To validate that your import was successful, expand the database object for the ADF data source in the Physical layer. Then, right-click a physical table and click **View Data**. If the appropriate data is displayed, the import completed successfully.

Using Incremental Import to Propagate Flex Object Changes

If you make changes to flexfields in your ADF applications, then you can use the Import Metadata Wizard in the Administration Tool to incrementally import the changes to the Physical layer of the Oracle BI repository.

The Import Metadata Wizard includes a synchronization feature for ADF data sources that enables you to import only the changes made to objects. The synchronization feature detects the changed objects, including new joined dimensions (KFF) and new attributes (DFF and EFF), so that they can be added automatically, without you having to search for them.

The synchronization feature detects the following:

- Changes in columns
- Additions or deletions of tables and columns
- Additions of keys and foreign keys
- Newly joined tables

Note that new tables that are joined to any existing table are only imported when you select the option **Automatically include any missing joined objects** on the Select Metadata Objects screen.

After import, the ADF data is modeled as shown in Table 6–1.

Table 6–1 How the ADF Metadata is Modeled in the Oracle BI Repository

ADF Metadata	Imported BI Metadata
Root Application Module	Database
View Objects	Physical Tables
View Object Attribute	Physical Column
View Object Key	Physical Key
View Links	Physical Joins

Note that as data is imported incrementally, modifications to properties of attributes are detected and propagated. For example, if an attribute changes its data type, that change is propagated to the physical layer objects.

If you are importing metadata into an existing database in the Physical layer, then confirm that the COUNT_STAR_SUPPORTED option is selected in the Features tab of the Database properties dialog. If you import metadata without the COUNT_STAR_SUPPORTED option selected, then the **Update Row Count** option will not display in the right-click menu for the database's physical tables.

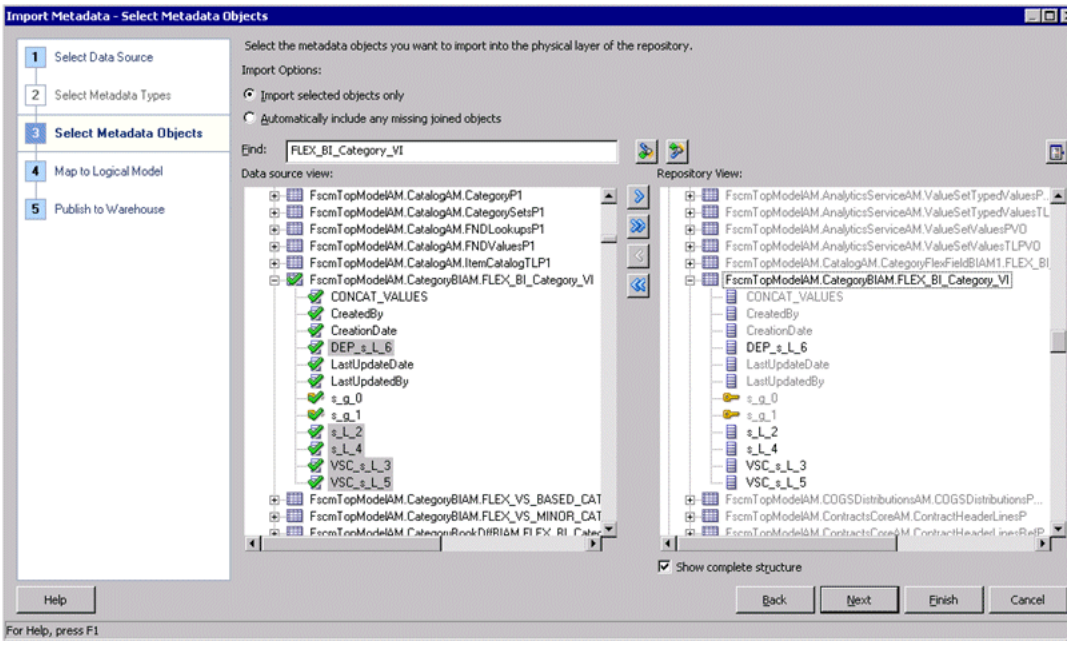
To incrementally import metadata for ADF data sources:

1. In the Administration Tool, in the Physical layer, right-click the connection pool for your ADF OLTP source and select **Import Metadata**.

   The Import Metadata Wizard is displayed, starting at Step 3, Select Metadata Objects.

   Figure 6–2 shows the Select Metadata Objects screen of the Import Metadata Wizard.

Figure 6–2 Select Metadata Objects Screen: ADF Data Source



2. Click **Synchronize** to locate and automatically select all recent changes for import. The Synchronize button is located in the upper right corner of the screen, above the Repository View.
3. Review the selected metadata to locate the new attributes.
4. Click **Finish** to close the wizard, or click **Next** to continue to the Map to Logical Model screen. See "Automatically Mapping Flex Object Changes to the Logical Model" for more information.

Automatically Mapping Flex Object Changes to the Logical Model

After importing changes to flexfields in your ADF application, you can use the Map to Logical Model screen of the Import Metadata Wizard in the Administration Tool to automatically propagate the changes to the Business Model and Mapping layer and Presentation layer.

If needed, you can override the default mapping behavior during this step by renaming logical tables, splitting a view object into multiple tables, combining multiple view objects into a single logical table, and so on.

See also "Customizing the Mapping Behavior" for information about using XML files to automate the mapping behavior displayed in this screen.

To automatically map flex object changes to the logical model:

1. In the Administration Tool, in the Physical layer, right-click the connection pool for your ADF OLTP source and select **Import Metadata**.
2. Complete the Select Metadata Objects screen and click **Next**. See "Using Incremental Import to Propagate Flex Object Changes" for more information about this screen.
3. In the Map to Logical Model screen, the Table Mapping and Column Mapping grids display the results of a default drag-and-drop. You can keep the default behavior, or customize the behavior for your needs. For example, you might want

to rename tables and columns in the Business Model and Mapping layer, map to an existing logical table, or map a logical column to multiple source columns.

Note that the Column Mapping grid shows alias columns as well as regular columns, so that you can handle customized mappings that include alias columns. The Table Mapping grid enables a single physical table to map to multiple logical tables, and the reverse.

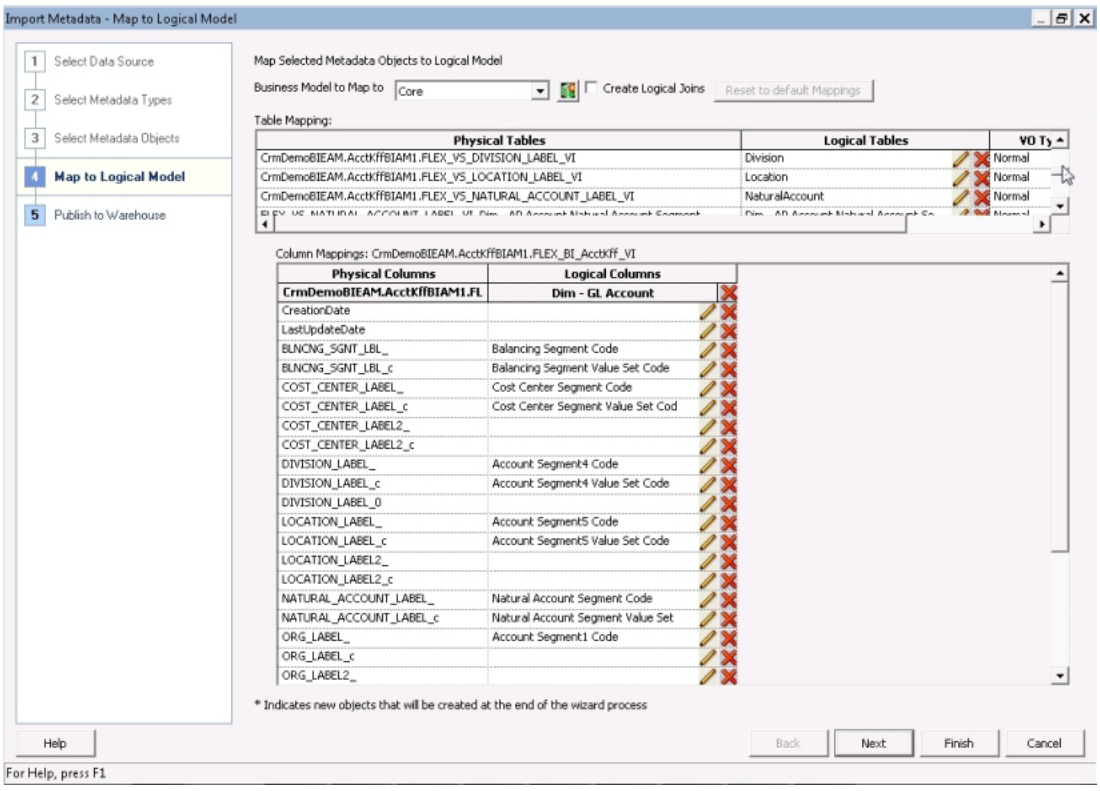
The Table Mapping grid includes a **VO Type** column. Options include **Normal**, **ETL Only**, and **Query Only**. ETL Only view objects exist only to extend the ETL mappings, and are not used for queries. Logical table sources that reference imported view objects of this type are marked as disabled in the Business Model and Mapping layer. Query Only view objects are only used for queries, and are not passed to the BI Extender for extension into the data warehouse.

The Table Mapping grid also includes a **Hierarchy** column. Select this option for objects that are hierarchies.

Select **Create Logical Joins** if the imported tables are being mapped to a new business model that will be created during the Map to Logical Model step. In other words, select this option when the imported logical joins do not already exist. Do *not* select this option for business models that already have the required logical joins in place. Doing so will create erroneous multiple logical joins.

Figure 6–3 shows the Map to Logical Model screen.

Figure 6–3 Map to Logical Model Screen of Import Metadata Wizard



4. Click **Finish** to close the wizard.

Customizing the Mapping Behavior

When setting up automatic mapping to the Logical Model, you can create a set of XML files that specify custom requirements for the mappings displayed in the Map to Logical Model screen.

The Administration Tool reads the XML files and then automatically maps the KFF, DFF, and EFF segments according to the specified logic. Each XML file has a top-level element with an appName attribute that specifies the application to which the file applies.

You must create your XML files according to the logic in the XML schema files app_segment_rule.xsd and mapping_rules.xsd. You can find these files in:

ORACLE_HOME\bifoundation\javahost\lib\obisintegration\biextender

All XML files in this directory with the prefix mapping_rules and app_segment_rules are parsed by the Administration Tool for ADF data sources.

You can use the existing app_segment_rules_*.xml and mapping_rules_*.xml in this directory as examples.

See Appendix H, "XML Schema Files for ADF Mapping Customizations" for more information about the app_segment_rule.xsd and mapping_rules.xsd XML schema files.

Manually Mapping Flex Object Changes to the Logical Model

You can choose to skip the logical mapping step in the Import Metadata Wizard, and instead drag and drop the physical objects to the Business Model and Mapping layer and Presentation layer.

The Administration Tool supports incremental drag-and-drop for ADF data sources, which enables physical database and schema objects to be dragged and dropped into an existing business model, resulting in updates made only for the incremental changes.

Note that the behavior in previous releases was to create new logical objects for every physical object that was dragged and dropped. The current logic includes data source-specific default rules that can enable, for example, logical dimensions and hierarchies to be automatically created.

Automatically Mapping Flex Object Changes Using the biserverextender Utility

You can use the `biserverextender` utility to import flex object changes from your ADF sources and map them to the Business Model and Mapping layer and Presentation layer.

Because this feature does not require the Administration Tool, it is especially useful when you want to map flex object changes on Linux and UNIX systems where the Administration Tool is not available.

To use the `biserverextender` utility, you must first create an XML parameter file that contains the connection pool for an existing ADF data source. The `biserverextender` utility retrieves the existing ADF connection pool name from the parameter file, synchronizes the ADF data source, updates the deployed objects in the source, and then maps physical metadata to the Business Model and Mapping and Presentation layers based on the default rule files in the following directory:

ORACLE_HOME/bifoundation/javahost/lib/obisintegration/biextender

See "Customizing the Mapping Behavior" for more information about the rule files.

Syntax

```
biserverextender -R base_repository_name [-P repository_password]
-O output_repository_name -I input_XML_file [-S]
```

Where:

`-R base_repository_name` is the name and path of the repository into which you want to import and map flex object changes.

`-P repository_password` is the Oracle BI repository password for the base repository.

Note that the *repository_password* argument is optional. If you do not provide the password argument, you are prompted to enter the password when you run the command. To minimize the risk of security breaches, Oracle recommends that you do not provide password arguments either on the command line or in scripts. Note that the password argument is supported for backward compatibility only, and will be removed in a future release. For scripting purposes, you can pass the password through standard input.

`-O output_repository_name` is the name and path of the repository generated by the utility.

`-I input_XML_file` is the name and path of an input XML parameter file that contains the fully-qualified name of a connection pool for an ADF data source.

-S is optional. If -S is not specified, only the changes from the ADF source's DFF, KFF, and EFF objects are synchronized to the Oracle BI Repository. If -S is specified, it reimports all DFF, KFF, and EFF objects from the ADF source based on the ADF source's database properties, and resynchronizes the Oracle BI Repository.

-S also incorporates the following changes in the app_segment_rules.xml rules file:

- New mapping rules segments
- New alias table creation
- New "ADF VO To Be Exposed" subject area or presentation table

Example

```
biserverextender -R /scratch/my_repos.rpd -O /scratch/my_repos_modelled.rpd
-I /scratch/ADFSource.xml -S
Give password: password
```

Sample XML Parameter File

```
<BIExtenderParameters>
 <ConnectionDetails>
  <ConnectionPool>
   <ConnectionPoolName>"oracle.apps.fscm.model.analytics.applicationModule.Fscm
   TopModelAM_FscmTopModelAMLocal"."Connection Pool"</ConnectionPoolName>
  </ConnectionPool>
 </ConnectionDetails>
</BIExtenderParameters>
```

Configuring SSL in Oracle WebLogic Server

You can configure one-way and two-way SSL in Oracle WebLogic Server.

This section contains the following topics:

- Configuring One-Way SSL in Oracle WebLogic Server

Configuring One-Way SSL in Oracle WebLogic Server

One-way SSL is required to properly secure the communication between Oracle Business Intelligence and Oracle WebLogic Server.

To configure one-way SSL in Oracle WebLogic Server:

1. From the WebLogic Server Administration Console home page, click **Servers** under the **Environment** heading.
2. In the Servers table, the name of the server you want to manage. Then, on the General subtab of the Configuration tab, select **SSL Listen Port Enabled**.
3. Use the Administration Tool to update the appropriate connection pool object in the Physical layer so that the URL uses https:// instead of http://. Also, update the port number to use the SSL port (7002 by default).

Configuring Two-Way SSL in Oracle WebLogic Server

Optionally, you can set up two-way SSL to secure the communication between Oracle Business Intelligence and the Oracle WebLogic Server.

To set up and test two-way SSL:

1. Create client certificates in the Oracle BI Server, if they do not already exist. See "Creating Certificates and Keys in Oracle Business Intelligence" in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition* for more information.
2. Modify the ADF Oracle WebLogic Server to accept SSL connections and to perform mutual SSL. To do this, perform the following steps in the Oracle WebLogic Server Administration Console:
   a. Log in to the Administration Console and click **Servers** under the Environment heading, then click the server name (for example, AdminServer).
   b. In the Change Center, click **Lock & Edit** to enable configuration changes.
   c. In the General tab, select **SSL Listen Port Enabled** and record the **SSL Listen Port** number. Then, click **Save**.
   d. Select the SSL tab, then select **Advanced**. For **Two Way Client Cert Behavior**, select **Client Certs Requested and Enforced**. Then, click **Save**.
   e. Select the Keystores tab and record the Trust Keystore that is being used. For example, if the Demo Trust keystore is used, record its location and file name.
   f. Click **Activate Changes**.
3. Ensure that the Certificate Authority (CA) for the Oracle BI Server client certificate is trusted by the ADF Oracle WebLogic Server. To do this, follow these steps:
   a. On the Oracle BI Server computer, find the CA file for the client certificate. If the file was generated using the instructions referenced in Step 1, the file will be "cacert.pem" in:

      ```
      ORACLE_HOME/user_projects/domains/bifoundation_domain/config/fmwconfig
      /biinstances/coreapplication/ssl
      ```

      Copy this file to a known location.

b. On the ADF Oracle WebLogic Server computer, open a command window and go to the location of the trust keystore. You recorded this value in Step 2. For example:

```
/scratch/user_name/view_storage/user_name_fmw/fmwtools/mw_home/wlserver_
10.3/server/lib
```

Copy the client CA file (for example, cacert.pem), stored in the previous step, to this location.

c. Use the JDK keytool utility to import the client CA into the trust keystore for the ADF server, making it a trusted CA. Use the following command:

```
keytool -import -file client_CA_file -keystore keystore_file -keystorepass
keystore_password
```

For example:

```
/scratch/my_name/view_storage/my_name_fmw/jdk6/bin/keytool -import -file
~/Downloads/SSL/cacert.pem -keystore DemoTrust.jks -keystorepass
DemoTrustKeyStorePassPhrase
```

d. Restart the ADF Oracle WebLogic Server.

4. Update the Physical layer of the Oracle BI repository, as follows:

   a. In the Administration Tool, in the Physical layer, open the first ADF connection pool object and select the Miscellaneous tab.

   b. Update the **URL** field to use the https protocol and the SSL port noted in Step 2, and then click **OK**.

   c. Repeat the previous two steps for each additional ADF connection pool object.

   d. Save the repository and restart the Oracle BI Server.

5. Configure the Oracle BI Server ODBC DSN to use SSL. For example, on Windows:

   a. Open the ODBC Data Source Administrator and select the System DSN tab.

   b. Double-click the DSN for the Oracle BI Server. The DSN should start with "coreapplication_OH."

   c. Select **Use SSL**.

   d. Click **Next**, click **Next** again, and then click **Finish**.

6. Perform queries against ADF using your Oracle BI Server client of choice (such as nqcmd). The Oracle BI Server should now be communicating with the ADF Oracle WebLogic Server using mutual SSL / client certs.

Enabling the Ability to Pass Custom Parameters to the ADF Application

Some ADF applications have custom properties defined on the ApplicationModule, such as EFFECTIVE_DATE or TREE_VERSION. You can include these custom properties in your application queries, and the Oracle BI Server will pass them to the ADF application.

You cannot use this feature to pass any custom property to your ADF application. Only certain custom properties, like EFFECTIVE_DATE and TREE_VERSION, are supported.

To enable this feature, use the Administration Tool to register the custom properties as a static repository variable.

To register custom properties:

1. Open your repository in the Administration Tool.
2. Select **Manage**, then select **Variables**.
3. Select **Action > New > Repository > Variable**.
4. For **Name**, enter ADF_PARAM_LIST. Do not enter the name of the custom property as the name of the variable.
5. Ensure that the **Type** is **Static**.
6. For **Default Initializer**, enter the name or names of the custom properties as a character string. If you have multiple custom properties, include them as a comma-delimited list. For example:

   'PARAM_EFFECTIVE_DATE'

   'PARAM_EFFECTIVE_DATE, ApplicationIdBind, KeyFlexfieldCodeBind'
7. Click **OK**.
8. Save and close the repository.

After you register the custom properties as a repository variable, you can include these variables in queries. For example:

```
set variable PARAM_EFFECTIVE_DATE=2001-01-01 : SELECT c1 FROM t1;
```

or

```
set variable ApplicationIdBind = '0', KeyFlexfieldCodeBind = 'KFF1' :
select_physical ApplicationID, KeyFlexfieldCode, DataSecurityObjectName,
SegmentLabelCode from adfdb..."AppModule.KFFHierFilterVO1";
```

Note that when you are including a custom property of type PARAM_EFFECTIVE_DATE, the date format for the property value must be in the format yyyy-mm-dd.

Propagating Labels and Tooltips from ADF Data Sources

You can propagate user interface hints, such as labels and tooltips, from ADF data sources to display when users work with analyses.

When translated labels and tooltips (based on user locale) are maintained within an ADF data source, you can query the data source to access this translated data. You use the Administration Tool to configure presentation columns to use when creating analyses.

This section contains the following topics:

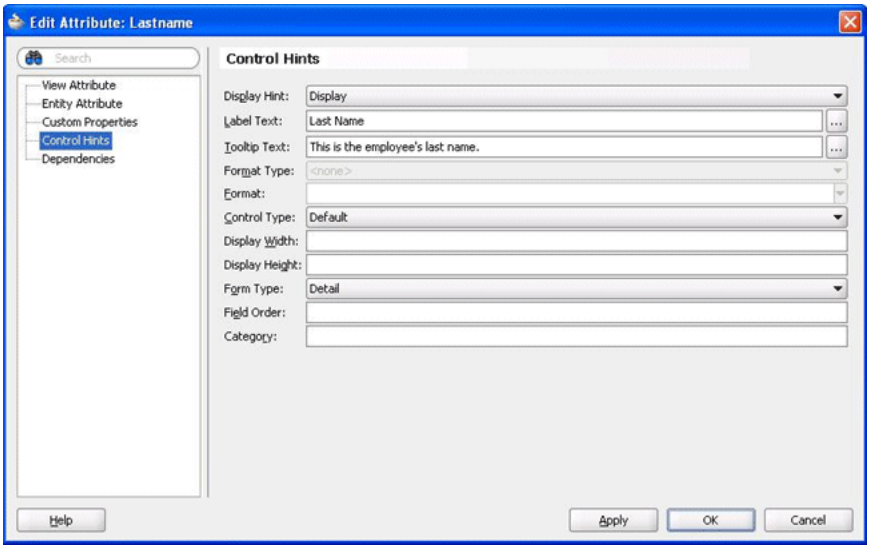
What are Labels and Tooltips?

The propagation of UI hints enables a presentation column in the Administration Tool to use a label and tooltip as its Custom display name and Description respectively.

A label is the text that is used in prompts or table headers that precedes the value of a data item. A tooltip is the text that is displayed when a user hovers over the item. Each attribute of a view object has an associated label and tooltip. A view object is the Oracle Application Development Framework component that enables a developer to work easily with SQL query results.

Figure 6–4 shows the Label Text and Tooltip Text options in the Edit Attribute dialog in Oracle JDeveloper.

Figure 6–4 Edit Attribute Dialog in JDeveloper for Label and Tooltip Options



About the Session Variable Naming Scheme for UI Hints

Session variable names are generated by the Oracle BI Enterprise Edition broker servlet in Oracle WebLogic Server in the following format:

ADF_*UI Hint Type_Database Name_View Object Name_Attribute's Name*

Where:

UI Hint Type is either LABEL or TOOLTIP, depending on the UI hint type that the session variable represents.

Database Name is the value of the "database" attribute of the ADFQuery element in the XML query. Special characters such single quotes ('), double quotes ("), and spaces are replaced by the underscore character.

View Object Name is the name of the view object to which the attribute belongs. Oracle ADF prohibits special characters and spaces in the name.

Attribute's Name is the name of the attribute that the session variable represents. Oracle ADF prohibits special characters and spaces in the name.

Every character in the session variable name is uppercase. For example, the XML query in Example 6–3 generates four session variables with the following names:

ADF_LABEL_MY_ORCLADF_EMPLOYEESVIEW_FIRSTNAME

ADF_TOOLTIP_MY_ORCLADF_EMPLOYEESVIEW_FIRSTNAME

ADF_LABEL_MY_ORCLADF_EMPLOYEESVIEW_LASTNAME

ADF_TOOLTIP_MY_ORCLADF_EMPLOYEESVIEW_LASTNAME

About Determining the Physical Column for a Presentation Column

As required by the naming scheme for session variables, each presentation column must map to a physical column.

When you select **Externalize Display Names > Generate ADF Label** or **Externalize Descriptions > Generate ADF Tooltip** for a presentation layer object, then the physical column is located using the following rules:

1. Examine the presentation column and determine its logical column. If the logical column is derived from an existing logical column, then the physical column cannot be found.
2. If the default aggregation rule for the logical column is not None or Sum, then the physical column cannot be found. It does not make sense semantically to use the ADF UI hints for aggregation rules other than Sum.
3. A logical column can be mapped to physical columns by multiple logical table sources. Only logical table sources that are not disabled are searched.
4. Do not search logical table sources that map the logical column using non-trivial expressions (that is, anything more than a physical column name). If no logical table sources are searched, then the physical column cannot be found.
5. From the remaining ordered list of logical table sources, examine the physical column that is mapped by the first logical table source. The physical column must be mapped to a view object attribute. In other words, the physical column must be part of a physical database of type OracleADF11g.
   - If this condition is satisfied, then the physical column for obtaining UI hints is found.
   - If this condition is not satisfied, then continue to examine the physical column that is mapped by the next logical table source until the physical column that is mapped to a view object attribute is found.

     If all logical table source are searched without satisfying the condition, then the physical column cannot be found.

If the physical column for obtaining UI hints is found using these rules, then the custom display name or description is populated with a session variable that has a name based on a predetermined naming scheme. See "About the Session Variable Naming Scheme for UI Hints" for more information.

If the physical column for obtaining UI hints is not found using these rules, then the **Generate ADF Label** and **Generate ADF Tooltip** options are shown as disabled in the right-click menu.

As an alternative to using the physical column found using these rules, you can use XML code in an initialization block to initialize your own session variables with ADF UI hints. You must then enter these session variable names in the **Custom display name** and **Custom description** fields manually. See "Using XML Code in Initialization Blocks to Query UI Hints" for more information.

About Initializing Session Variables Automatically for Propagating UI Hints

If the **Externalize Display Names > Generate ADF Label** and **Externalize Descriptions > Generate ADF Tooltip** options were used to successfully generate the session variable names for UI hints from Oracle ADF, then the session variables are created and initialized when Oracle BI Presentation Services queries them during the session.

The variables are not created and initialized during the session logon stage for performance reasons. Instead, the variables are created and initialized when they are needed by a specific query within a session, using the **Allow deferred execution** feature.

When Presentation Services queries the custom display names and custom descriptions through ODBC, the Oracle BI Server checks if the associated session variables have been created. If they have not been created, then the Oracle BI Server dynamically generates the appropriate XML query (as described in "Using XML Code in Initialization Blocks to Query UI Hints") to query the UI hints from the Oracle ADF data source. The Oracle BI Server uses the UI hints to create and initialize the session variables. As an optimization, the Oracle BI Server queries UI hints for each view object; that is, if the Oracle BI Server needs the UI hints of a view object's attributes, then the UI hints for all the attributes under the view object are queried and propagated through session variables.

Using UI Hints From an Oracle ADF Data Source When Creating Analyses

You can use UI hints from an Oracle ADF data source when creating analyses.

Before you can perform this task, the following prerequisites must be met:

- UI hints must have been configured in the Oracle ADF data source.
- A working repository must have been configured for the Oracle ADF data source in the Administration Tool.

To use UI hints from an Oracle ADF data source when creating analyses:

1. Suppose that the repository contains a presentation column named "LastName." On the General tab of the Presentation Column dialog, the **Custom display name** and **Custom description** fields are not selected.

   Right-click the column in the Presentation layer and select first **Externalize Display Names > Generate ADF Label**, then **Externalize Descriptions > Generate ADF Tooltip** to generate the strings that populate the **Custom display name** and **Custom description** fields.

   You can also use these options from the right-click menu of a presentation table to generate the strings for all the columns in that table.

2. View the UI hints:

   a. Sign in to Oracle Business Intelligence.

   b. Create a new analysis using the subject area for which you obtained UI hints.

   c. In the Subject Areas pane, expand the Employee folder to see the UI hints that have been propagated from the Oracle ADF data source.

      The LastName column displays as "Last Name" (the label value from the Oracle ADF data source). When you hover the mouse pointer over the column, the tip displays as "This is the employee's last name" (the tooltip value from the Oracle ADF data source).

For information about creating analyses, see *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition*.

Using XML Code in Initialization Blocks to Query UI Hints

As an alternative to using the automated system, you can use specialized XML code in place of SQL statements in initialization blocks to query the data source for UI hints, within a single repository and subject area.

You use the ADFQuery element, which has three attributes that are named mode, database, and locale. The element requires zero or more child elements. The syntax of the element is as follows:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<ADFQuery mode="mode" database="database_name"
locale="VALUEOF(NQ_SESSION.WEBLANGUAGE)">
     <ViewObject><![CDATA[view_object_name]]></ViewObject>
     <Attribute>
     <ViewObject><![CDATA[attribute_view_object_name]]></ViewObject>
     <Name><![CDATA[attribute_name]]></Name>
     </Attribute>
</ADFQuery>
```

Where:

mode specifies what you want to query:

- `label` for querying attributes' label
- `tooltip` for querying attributes' tooltip
- `ui_hints` for querying attributes' label and tooltip

database_name specifies the name of the physical database object in the Administration Tool, which contains the physical columns that correspond to the attributes in the Oracle ADF data source.

view_object_name specifies the name of the view object to obtain the UI hints of all attributes in it.

attribute_view_object_name specifies the name of the view object that contains the attribute.

attribute_name specifies the name of the attribute that belongs to the associated view object to obtain the UI hints of this attribute.

Example 6–1 Querying Labels for All View Objects

No child elements must be included in the ADFQuery element, if the UI hints of all attributes in all View Objects are queried. For example, to query the labels of all attributes in all View Objects under the My_orclADF physical database object, use the following XML code:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<ADFQuery mode="label" database="My_orclADF"
locale="VALUEOF(NQ_SESSION.WEBLANGUAGE)">
</ADFQuery>
```

Example 6–2 Querying Tooltips for Specific View Objects

The ADFQuery element can contain zero or more child elements named ViewObject if UI hints of all attributes in specific View Objects are queried. Each ViewObject element has a text content that contains the View Object's name. The ViewObject element is

used to specify the View Objects from which the UI hints of all attributes are queried. For example, to query the tooltips of all attributes in the View Object that is named EmployeesView and CustomersView under the My_orclADF physical database object, use the following XML code:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<ADFQuery mode="tooltip" database="My_orclADF"
locale="VALUEOF(NQ_SESSION.WEBLANGUAGE)">
   <ViewObject><![CDATA[EmployeesView]]></ViewObject>
   <ViewObject><![CDATA[CustomersView]]></ViewObject>
</ADFQuery>
```

Example 6–3 Querying UI Hints for Specific Attributes

The ADFQuery element can contain zero or more child elements named Attribute. Each Attribute element has two required child elements named ViewObject and Name. The Attribute element is used to specify the attributes from which the UI hints are queried. The ViewObject child element has a text content that contains the View Object's name. This element specifies the View Object that the attribute belongs to. The Name child element has a text content which contains the attribute's name. For example, to query the labels and tooltips of the attributes named Firstname and Lastname in the EmployeesView View Object under the My_orclADF physical database object, use the following XML code:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<ADFQuery mode="ui_hints" database="My_orclADF"
locale="VALUEOF(NQ_SESSION.WEBLANGUAGE)">
   <Attribute>
      <ViewObject><![CDATA[EmployeesView]]></ViewObject>
      <Name><![CDATA[Firstname]]></Name>
   </Attribute>
   <Attribute>
      <ViewObject><![CDATA[EmployeesView]]></ViewObject>
      <Name><![CDATA[Lastname]]></Name>
   </Attribute>
</ADFQuery>
```

After configuring the initialization blocks, you must manually enter the session variable names in the **Custom display name** and **Custom description** text fields for the appropriate presentation column.

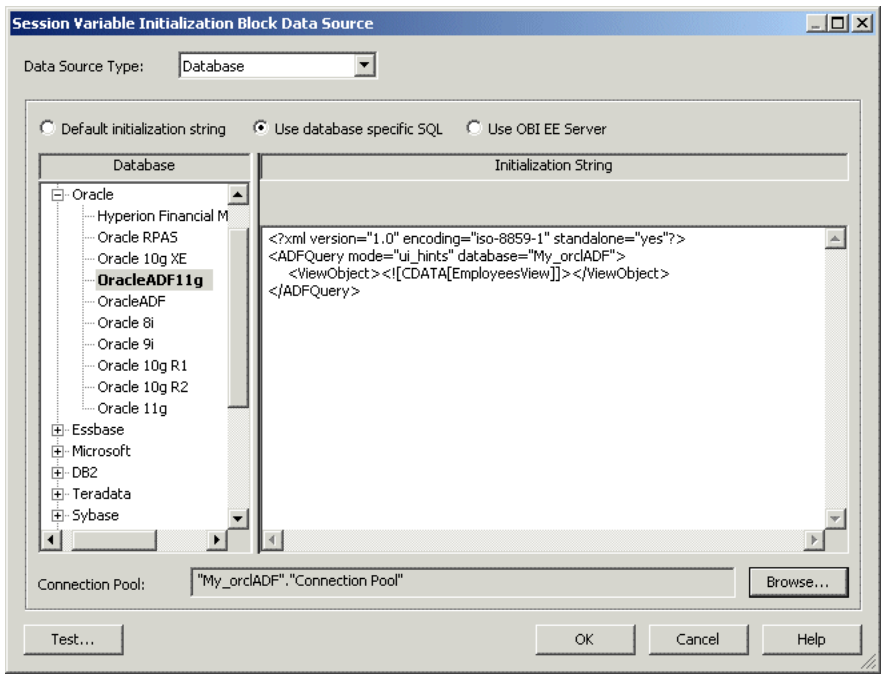
Follow the procedure in the example in "Using UI Hints From an Oracle ADF Data Source When Creating Analyses", but replace the first step with the following ones:

1. Create session initialization blocks in the Administration Tool.

   a. In the Session Variable Initialization Block Data Source dialog, enter the Initialization string.

      In this example, the initialization block queries both the label and tooltip of all attributes in a View Object named EmployeesView. Figure 6–5 shows the setup of a session variable initialization block with an appropriate Oracle ADF UI hint query. "My_orclADF"."Connection Pool" is a connection pool for an Oracle ADF data source.

Figure 6–5 Setting Up a Session Variable Initialization Block Data Source with an Oracle ADF UI Hints Query



b. In the Session Variable Initialization Block dialog, select **Row-wise initialization** as the Variable Target.

c. Click **Test** to test the query against the Oracle ADF data source.

In the results window, the first column contains the session variable names that are generated using the naming scheme. The second column contains the label and tooltip values from the Oracle ADF data source.

See "About the Session Variable Naming Scheme for UI Hints" for a description of the naming scheme.

2. Configure a custom display name and a description in presentation columns.

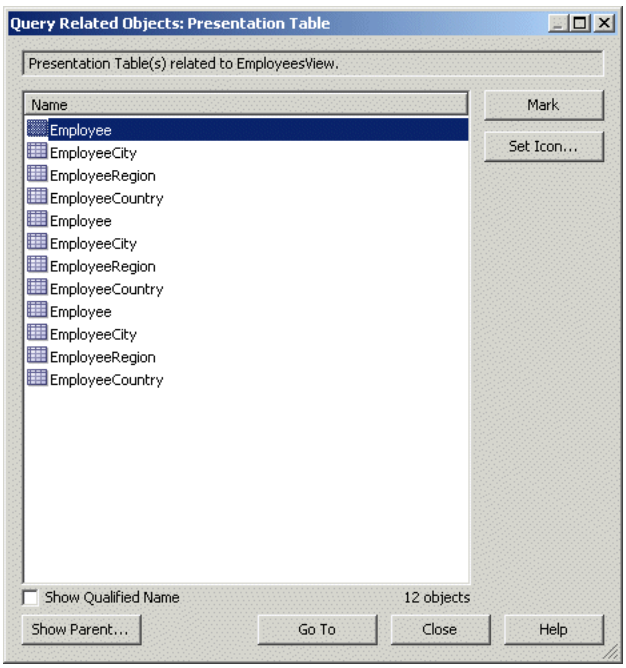
To find the presentation tables that can use the UI hints from the EmployeesView View Object, this example uses the Query Repository feature in the Administration Tool.

a. Right-click a physical table (for example, EmployeesView), then select **Query Related Objects** > **Presentation** > **Presentation Table** from the menu.

The Query Related Objects dialog displays all the related presentation tables.

This example sets up a custom display name and custom description for columns in the Employee presentation table.

Figure 6–6 Using the Query Related Objects Feature to Find the Related Presentation Tables



b. Select the required presentation table and click **Go To**.

This displays the selected presentation table.

c. Expand the presentation table to view the presentation columns.

d. Double-click the LastName presentation column to display the Presentation Column dialog.

e. Select **Custom display name** and enter a value such as the following one:

```
VALUEOF(NQ_SESSION.ADF_LABEL_MY_ORCLADF_EMPLOYEESVIEW_LASTNAME)
```

f. Select **Custom description** and enter a value such as the following one:

```
VALUEOF(NQ_SESSION.ADF_TOOLTIP_MY_ORCLADF_EMPLOYEESVIEW_LASTNAME)
```

g. Click **OK**.

h. Save the changes in the repository and restart the Oracle BI Server.

7

Setting Up Database Objects and Connection Pools

This chapter describes the properties of the database and connection pool objects in the Physical layer.

Properties for database objects and connection pools are typically set automatically when you import metadata from your data sources. However, in some cases you may want to adjust database or connection pool settings, or create a database object or connection pool manually.

This chapter contains the following sections:

Setting Up Database Objects

Importing metadata from a data source automatically creates a database object for the schema, but you may need to adjust or view the database properties.

In addition, you sometimes need to manually create a database object and connection pool for certain situations like configuring usage tracking, setting up Oracle Scorecard and Strategy Management, or configuring aggregate persistence targets.

See "System Requirements and Certification" for information about supported data sources.

The following sections provide information about how to create, edit, or view properties for database objects in the Physical layer:

About Database Types in the Physical Layer

If you import the physical schema into the Physical layer, the Administration tool usually assigns database type automatically.

The following list contains additional information about automatic assignment of database types:

- **Relational data sources.** During the import process, some ODBC drivers provide the Oracle BI Server with the database type. However, if the server cannot determine the database type, an approximate ODBC type is assigned to the database object. Replace the ODBC type with the closest matching entry from the **Database** list.
- **Multidimensional data sources.** Microsoft Analysis Services and SAP/BW are the only supported XMLA-compliant data sources currently available. After you import metadata from a multidimensional data source, check the database object and update the appropriate database type and version if necessary.

Creating a Database Object Manually in the Physical Layer

If you create a database object manually, you also need to manually set up an associated connection pool.

For multidimensional data sources, if you create the physical schema in the Physical layer of the repository, you need to create one database in the physical layer for each cube, or set of cubes, that belong to the same catalog (database) in the data source. A physical database can have more than one cube. However, all of these cubes must be in the same catalog in the data source.

Caution: It is strongly recommended that you import your physical schema.

To create a database object:

1. In the Administration Tool, in the Physical layer, right-click and select **New Database**.

   Make sure that no object is selected when you right-click.

2. In the Database dialog, in the General tab, complete the fields using Table 7–1 as a guide.

Table 7–1 Options in the General Tab of the Database Dialog

Option	Description
Data source definition: Database	The database type for your database. See "Specifying SQL Features Supported by a Data Source" for more information about using the Features tab to examine the SQL features supported by the specified database type.
Data source definition: CRM metadata tables	This property is only available for relational data sources and is for legacy Siebel Systems sources only. When selected, indicates that the definition of physical tables and columns for Siebel CRM tables was derived from the Siebel metadata dictionary.

Table 7–1 (Cont.) Options in the General Tab of the Database Dialog

Option	Description
Data source definition: Virtual Private Database	Identifies the physical database source as a virtual private database (VPD). When a VPD is used, returned data results are contingent on the user's authorization credentials. Therefore, it is important to identify these sources. These data results affect the validity of the query result set that is used with caching. See "Managing Performance Tuning and Query Caching" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*. Always select this option for Essbase, Hyperion Financial Management, and Hyperion Planning data sources that are configured for **SSO** in the corresponding connection pool. **Note:** If you select this option, you also should select the **Security Sensitive** option in the Session Variable dialog. See "Creating Session Variables" for more information.
Persist connection pool	To use a persistent connection pool, you must set up a temporary table first. See "Setting Up Persist Connection Pools" for more information.
Allow populate queries by default	When selected, allows everyone to execute `POPULATE SQL`. If you want most, but not all, users to be able to execute `POPULATE SQL`, select this option and then limit queries for specific users or groups. See "Setting Query Limits" for more information.
Allow direct database requests by default	When selected, allows all users to execute physical queries. The Oracle BI Server sends unprocessed, user-entered, physical SQL directly to an underlying database. The returned results set can be rendered in Oracle BI Presentation Services, and then charted, rendered in a dashboard, and treated as an Oracle BI request. If you want most, but not all, users to be able to execute physical queries, select this option and then limit queries for specific users or groups. See "Setting Query Limits" for more information. **Caution:** If configured incorrectly, this option can expose sensitive data to an unintended audience. See "When to Allow Direct Database Requests by Default" for more information. For more information about executing physical SQL, see "Working with Direct Database Requests" in *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition*.

When to Allow Direct Database Requests by Default

The property **Allow direct database requests by default** lets all users execute physical queries. If configured incorrectly, it can expose sensitive data to an unintended audience.

Use the following recommended guidelines when setting this database property:

- The Oracle BI Server should be configured to accept connection requests only from a computer on which the Oracle BI Server, Oracle BI Presentation Services, or Oracle BI Scheduler are running. This restriction should be established at the TCP/IP level using the Oracle BI Presentation Services IP address. This allows only a TCP/IP connection from the IP address of Oracle BI Presentation Services.
- To prevent users from running `nqcmd` (a utility that executes SQL scripts) by logging in remotely to this computer, you should disallow access by the following to the computer on which you installed Oracle BI Presentation Services:

  - TELNET
  - Remote shells
  - Remote desktops
  - Teleconferencing software (such as Windows NetMeeting)

  If necessary, you might want to make an exception for users with administrator permissions.

- Only users with administrator permissions should be allowed to perform the following tasks:
  - TELNET into the Oracle BI Server and Oracle BI Presentation Services computers to perform tasks such as running `nqcmd` for cache seeding.
  - Access the advanced SQL page of Answers to create requests. For more information, see *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition*.
- Set up group/user-based permissions on Oracle BI Presentation Services to control access to editing (preconfigured to allow access by Oracle BI Presentation Services administrators) and executing (preconfigured to not allow access by anyone) direct database requests. For more information, see *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*.

Specifying SQL Features Supported by a Data Source

When you import metadata or specify a database type in the General tab of the Database dialog, the set of SQL features for that database object is automatically populated with default values appropriate for the database type. The Oracle BI Server uses these SQL features with this data source.

When a feature is marked as supported (checked) in the Features tab of the Database dialog, the Oracle BI Server typically pushes the function or calculation down to the data source for improved performance. When a function or feature is not supported in the data source, the calculation or processing is performed in the Oracle BI Server.

The supported features list in the Features tab uses the feature defaults defined in the DBFeatures.INI file, located in *ORACLE_INSTANCE*\config\OracleBIServerComponent\coreapplication_obis*n*. Although you should not modify this file directly, it can be useful to look at this file to compare the features supported by different data source types.

You can tailor the query features for a data source. For example, a new version of a data source may be released with updated feature support that is not reflected in the Oracle BI Server defaults. In this case, you can update the settings in the Features tab to reflect the actual features supported by the new version of the data source. Or, if a data source supports a particular feature (such as left outer join queries) but you want to prohibit the Oracle BI Server from sending such queries to a particular data source, you can change this default setting in the Features tab. A third situation is when you have federated data sources that execute functions differently. To ensure query results are consistent, you can disable the appropriate functions so that the calculations are performed in a consistent manner in the Oracle BI Server.

Caution: Be very careful when modifying the set of supported features in the Features tab. If you enable SQL features that the data source does not support, your query may return errors and unexpected results. If you disable supported SQL features, the server could issue less efficient SQL to the data source.

In most cases, you should keep the default values. If you do change the defaults to mark a feature as supported in the Features tab, make sure that the feature is actually supported by the data source.

To specify SQL features supported by a data source:

1. In the Administration Tool, in the Physical layer, double-click the database for which you want to specify SQL features.
2. In the Database dialog, click the Features tab.
3. In the Features tab, use the information in Table 7–2 to help you specify properties for each SQL feature.

Table 7–2 Options in the Features Tab of the Database Dialog

Option	Description
Feature	The name of the database feature, such as `COUNT_DISTINCT_SUPPORTED`.
Value	Shows the current value for the given feature. Selected indicates that the feature is supported in the data source, and that the function or feature should be performed in the data source rather than in the Oracle BI Server. Some features show a default value in the **Value** column rather than selected/not selected, such as 10 for `MAX_ENTRIES_PER_IN_LIST`. It is strongly recommended that you keep the default selections and default values.
Default	Shows the default value for the given feature. The defaults listed in this column are specified in the file DBFeatures.INI.
Find	Lets you type a string to help you locate a feature in the list.
Find Again	This option becomes available after you click **Find**. It lets you perform multiple searches for the same string.
Query DBMS	This button is only used if you are installing and querying a data source that has no set of feature defaults in the Oracle BI Server. It lets you query this type of data source for Feature table entries so that you can find out which SQL features it supports. You can then change the entries that appear in the Features tab based on your query results. This button is not available if you are using an XML or a multidimensional data source. **Caution:** Be very careful when using the Query DBMS feature. The results of the features query are not always an accurate reflection of the SQL features actually supported by the data source. When using this feature, you should verify that the list of supported features in the Features tab matches the actual features supported by your data source. Refer to the documentation for your data source for details.
Reset to defaults	This button restores the default values for this data source type from the file DBFeatures.INI.

Note: Do not change the `OPTIMIZE_MDX_FILTER_QUALIFICATION` feature. This parameter is reserved for a future release.

Viewing Database Properties

The Database Properties tab is used for some data sources as a generic mechanism for extending the Physical layer metadata.

For example, for Oracle ADF data sources, you can view custom database properties that are passed to the Administration Tool from Oracle ADF BI view objects. Click the Database Properties tab to view the custom properties. You do not typically need to create or edit these properties.

Table 7–3 shows examples of custom properties.

Table 7–3 Examples of Custom Properties in the Database Properties Tab

Category	Key Name	Value	Description
FscmTopModelAM.AccountBIAM	BIObject_FLEX_TREE_VS_COST_CENTER_LABEL_VI	Dim - Cost Center	FLEX_TREE_VS_COST_CENTER_LABEL_VI view object needs to map to the Dim - Cost Center logical dimension
FscmTopModelAM.AccountBIAM	BIFlexfieldViewUsage	FLEX_BI_AcctKff_VI	FLEX_BI_AcctKff_VI is the CCID view object for FscmTopModelAM.AccountBIAM
FscmTopModelAM.AccountBIAM	EnforceCustomDataType_FscmTopModelAM.AccountBIAM	"Segment 1":"VARCHAR"; "Segment ID":"DOUBLE"	For FscmTopModelAM.AccountBIAM view objects, the data type of some physical columns needs to be overridden with the values passed in the property

About Connection Pools

The connection pool is an object in the Physical layer that describes access to the data source. It contains information about the connection between the Oracle BI Server and that data source.

The Physical layer in the Administration Tool contains at least one connection pool for each database. When you create the Physical layer by importing a schema for a data source, the connection pool is created automatically. You can configure multiple connection pools for a database. Connection pools allow multiple concurrent data source requests (queries) to share a single database connection, reducing the overhead of connecting to a database.

Note: It is recommended that you create a dedicated connection pool for initialization blocks. See "About Connection Pools for Initialization Blocks" for more information.

For each connection pool, you must specify the maximum number of concurrent connections allowed. After this limit is reached, the connection request waits until a connection becomes available.

Increasing the allowed number of concurrent connections can potentially increase the load on the underlying database accessed by the connection pool. Test and consult with your DBA to make sure the data source can handle the number of connections

specified in the connection pool. Also, if the data sources have a charge back system based on the number of connections, you might want to limit the number of concurrent connections to keep the charge-back costs down.

In addition to the potential load and costs associated with the database resources, the Oracle BI Server allocates shared memory for each connection upon server startup. This raises the number of connections and increases Oracle BI Server memory usage.

About Connection Pools for Initialization Blocks

It is recommended that you create a dedicated connection pool for initialization blocks. This connection pool should not be used for queries.

Additionally, it is recommended that you isolate the connections pools for different types of initialization blocks. This also makes sure that authentication and login-specific initialization blocks do not slow down the login process. The following types should have separate connection pools:

- All authentication and login-specific initialization blocks such as language, externalized strings, and group assignments.
- All initialization blocks that set session variables.
- All initialization blocks that set repository variables. These initialization blocks should always be run using credentials with administrator privileges.

  Be aware of the number of these initialization blocks, their scheduled refresh rate, and when they are scheduled to run. Typically, it would take an extreme case for this scenario to affect resources. For example, refresh rates set in minutes, greater than 15 initialization blocks that refresh concurrently, and a situation in which either of these scenarios could occur during prime user access time frames.

Initialization blocks should be designed so that the maximum number of Oracle BI Server variables may be assigned by each block. For example, if you have five variables, it is more efficient and less resource intensive to construct a single initialization block containing all five variables. When using one initialization block, the values are resolved with one call to the back end tables using the initialization string. Constructing five initialization blocks, one for each variable, would result in five calls to the back end tables for assignment.

If an initialization block fails for a particular connection pool during Oracle BI Server start-up, no more initialization blocks using that connection pool are processed. Instead, the connection pool is blacklisted and subsequent initialization blocks for that connection pool are skipped. This behavior ensures that the Oracle BI Server starts in a timely manner, even when a connection pool has a large number of associated initialization blocks or variables.

If this occurs, a message similar to the following appears in the server log:

```
[OracleBIServerComponent] [ERROR:1] [43143] Blacklisted connection pool
name_of_connection_pool
```

If you see this error, check the initialization blocks for the given connection pool to ensure they are correct.

See "Working with Initialization Blocks" for more information about these objects.

Creating or Changing Connection Pools

Typically, database objects and connection pools are created automatically when you import physical schemas, for both relational and multidimensional data sources. If you

did not import physical schemas, you must create a database object before you create a connection pool.

You create or change a connection pool in the Physical layer of the Administration Tool.

To modify more than one connection pool, use the "List Connection Pool Command" and the "Update Connection Pool Command."

If you have already defined an existing database and connection pool, you can right-click the connection pool in the Physical layer and select **Import Metadata** to import metadata for this data source. The Import Metadata Wizard appears with the information on the Select Data Source screen pre-filled. See Chapter 5, "Importing Metadata and Working with Data Sources" for information about the Import Wizard.

To automate connection pool changes for use in a process such as production migration, consider using the Oracle BI Server XML API. See "About the Oracle BI Server XML API" in *Oracle Fusion Middleware XML Schema Reference for Oracle Business Intelligence Enterprise Edition* for more information.

To create or change a connection pool:

1. In the Physical layer of the Administration Tool, right-click a database and select **New Object**, then select **Connection Pool**. Or, double-click an existing connection pool.
2. Specify or adjust the properties as needed, then click **OK**.

The following sections describe how to set properties in the various tabs of the Connection Pool dialog:

- Setting Connection Pool Properties in the General Tab
- Setting Connection Pool Properties in the Connection Scripts Tab
- Setting Connection Pool Properties in the XML Tab
- Setting Connection Pool Properties in the Write Back Tab
- Setting Connection Pool Properties in the Miscellaneous Tab

Setting Connection Pool Properties in the General Tab

This section describes the properties in the General tab of the Connection Pool dialog. The General tab is available for all data sources.

To set general properties for connection pools:

- In the Connection Pool dialog, click the General tab, and then complete the fields using the information in Table 7–4 and Table 7–5.

The properties listed in the General tab vary according to the data source type. For example, XMLA data sources have a connection pool property for **URL**, while relational and XML data sources have the option **Require fully qualified table names**.

This section contains the following topics:

- Common Connection Pool Properties in the General Tab
- Multidimensional Connection Pool Properties in the General Tab

Common Connection Pool Properties in the General Tab

This section describes connection pool properties in the General tab that are common among most data source types.

Figure 7–1 shows the General tab of the Connection Pool dialog, for an OCI data source.

Figure 7–1 General Tab of the Connection Pool Dialog: OCI Data Source

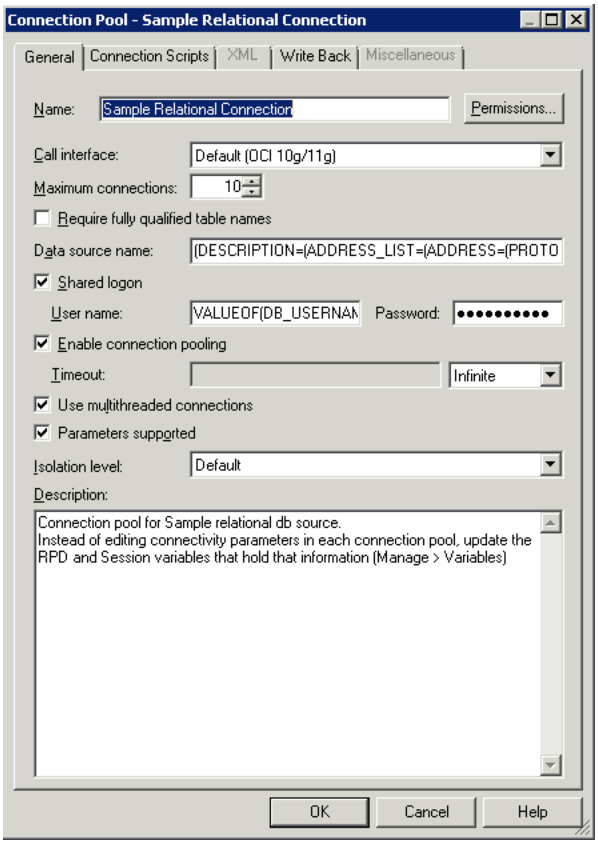


Table 7–4 describes the properties in the General tab of the Connection Pool dialog that are common for different data source types.

Table 7–4 Connection Pool Properties in the General Tab: Common Properties

Property	Description
Name	The name for the connection pool. A name is assigned automatically for connection pools created upon import.
Permissions	Use this option to assign permissions for individual users or application roles to access the connection pool. For example, you can set up a privileged group of users to have its own connection pool. This feature is not intended to be used for data access security. For example, connection pool permissions do not protect cache entries. Refer to Chapter 14 for complete information on data access security in Oracle Business Intelligence.
Call interface	Identifies the application programming interface (API) with which to access the data source. Some databases can be accessed using native APIs, some use ODBC, and some work both ways. Java data sources are accessed using JDBC/JNDI. If the call interface is XML, the XML tab is available but options that do not apply to XML data sources are not available.

Table 7–4 (Cont.) Connection Pool Properties in the General Tab: Common Properties

Property	Description
Maximum connections	The maximum number of connections allowed for this connection pool. The default is 10. This value should be determined by the database make and model and the configuration of the hardware for the computer on which the database runs, as well as the number of concurrent users who require access. For Microsoft Analysis Services data sources, you might encounter 503 Service Not Available errors if the Max Connections setting in the connection pool (default 10) is greater than the XMLA MaxThreadsPerClient setting configured in Analysis Services (default 4). To avoid these errors, increase the MaxThreadsPerClient setting in the msmdpump.ini file, or reduce the Max Connections setting in the repository connection pool. See also "Improving Use of System Memory Resources with TimesTen Data Sources" for related information. **Note:** For deployments with Oracle BI Interactive Dashboards pages, consider estimating this value at 10% to 20% of the number of simultaneous users multiplied by the number of requests on a dashboard. This number can be adjusted based on usage. The total number of all connections in the repository should be less than 800. To estimate the maximum connections needed for a connection pool dedicated to an initialization block, you might use the number of users concurrently logged on during initialization block execution.

Table 7–4 (Cont.) Connection Pool Properties in the General Tab: Common Properties

Property	Description
Require fully qualified table names	Select this option if the database or database configuration requires fully qualified table names. This option is not available for some data source types. When this option is selected, all requests sent from the connection pool use fully qualified names to query the underlying database. The fully qualified names are based on the physical object names in the repository. If you are querying the same tables from which the Physical layer metadata was imported, you can safely select this option. If you have migrated your repository from one physical database to another physical database that has different database and schema names, the fully qualified names would be invalid in the newly migrated database. In this case, if you do not select this option, the queries will succeed against the new database objects. For some data sources, fully qualified names might be safer because they guarantee that the queries are directed to the desired tables in the desired database. For example, if the RDBMS supports a master database concept, a query against a table named Customer first looks for that table in the master database, and then looks for it in the specified database. If the table named Customer exists in the master database, that table is queried, not the table named Customer in the specified database. It is sometimes necessary to select this option when you are using an Oracle Database, and you are accessing the database with a user that is not the owner of the schema containing the tables. When the Oracle Database interprets table names in SQL, it assumes that the user that made the query is the owner if the table name is not fully qualified in the query. This can result in an incorrect qualified name. For example, if the user SAMPLE creates a table called CUSTOMER, the fully qualified table name is SAMPLE.CUSTOMER. When the SAMPLE user references the CUSTOMER table in a query, the Oracle Database assumes the fully qualified table name is SAMPLE.CUSTOMER, and the access is successful. However, if the JANEDOE user references the CUSTOMER table in a query, the Oracle Database assumes the fully qualified table name is JANEDOE.CUSTOMER, and a "Table or view not found" error can result. To enable access for JANEDOE, you must select **Require fully qualified table names** in the connection pool so that the Oracle BI Server specifies SAMPLE.CUSTOMER in all queries.

Table 7–4 (Cont.) Connection Pool Properties in the General Tab: Common Properties

Property	Description
Data source name	The name of the data source to which you want this connection pool to connect and send physical queries. The value you enter in this field depends on the selected call interface: ■ If the call interface is OCI, enter a full connect string or a net service name from the tnsnames.ora file you set up within the Oracle Business Intelligence environment, in *ORACLE_HOME*/network/admin. ■ If you are using a native interface for a different database, enter the name of the database for that system. ■ If the call interface is ODBC, the data source name field displays a list containing all the User and System DSNs defined for ODBC on the local computer. Select the correct one for the data source to which you want connect. If you are using Microsoft SQL Server, then enter an ODBC data source name or a full connect string. The following is the syntax for the full connect string: `Driver={Driver Name};Address=Host Name;Database=Database Name` Where *Driver Name* refers to the Microsoft SQL Server ODBC driver name. This driver name must exist in odbcinst.ini, and the environment variable ODBCINST should be set to point to odbcinst.ini.
Shared logon	Select this option if you want all users whose queries use the connection pool to access the underlying database using the same user name and password. If this option is selected, then all connections to the database that use the connection pool use the user name and password specified in the connection pool, even if the user has specified a database user name and password in the DSN (or in user configuration). If this option is not selected, connections through the connection pool use the database user ID and password specified in the DSN or in the user profile.
Enable connection pooling	When selected, allows a single database connection to remain open for the specified time for use by future query requests. Connection pooling saves the overhead of opening and closing a new connection for every query. If you do not select this option, each query sent to the database opens a new connection.
Timeout	Specify the amount of time and in what increment (such as minutes) that a connection to the data source remains open after a request completes. During this time, new requests use this connection rather than open a new one (up to the number specified for the maximum connections). The time is reset after each completed connection request. If you are using an ADF data source and the call interface is OracleADF_HTTP and the query mode is SQLBypass, then **Timeout** specifies the maximum execution time before the connection is canceled.

Table 7–4 (Cont.) Connection Pool Properties in the General Tab: Common Properties

Property	Description
Use multithreaded connections	When this option is selected, the Oracle BI Server terminates idle physical queries (threads). When not selected, one thread is tied to one database connection (number of threads = maximum connections). Even if threads are idle, they consume memory. The parameter `DB_GATEWAY_THREAD_RANGE` in the Server section of NQSConfig.ini establishes when the Oracle BI Server terminates idle threads. The lower number in the range is the number of threads that are kept open before the Oracle BI Server takes action. If the number of open threads exceeds the low point in the range, the Oracle BI Server terminates idle threads. For example, if `DB_GATEWAY_THREAD_RANGE` is set to 40-200 and 75 threads are open, the Oracle BI Server terminates any idle threads.
Parameters supported	If this option is not selected, and the database features table supports parameters, special code executes that allows the Oracle BI Server to push filters (or calculations) with parameters to the database. The Oracle BI Server does this by simulating parameter support within the gateway/adapter layer by sending extra SQLPrepare calls to the database.
Isolation level	For ODBC and DB2 gateways only. The value sets the transaction isolation level on each connection to the back-end database. The isolation level setting controls the default transaction locking behavior for all statements issued by a connection. Only one option can be set at a time. It remains set for that connection until it is explicitly changed. The following options are available: **Dirty read.** Implements dirty read (isolation level 0 locking). This is the least restrictive isolation level. When this option is set, it is possible to read uncommitted or dirty data, change values in the data, and have rows appear or disappear in the data set before the end of the transaction. *Dirty data* is data that needs to be cleaned before being queried to obtain correct results (for example, duplicate records, records with inconsistent naming conventions, or records with incompatible data types). **Committed read.** Specifies that shared locks are held while the data is read to avoid dirty reads. However, the data can be changed before the end of the transaction, resulting in non repeatable reads or phantom data. **Repeatable read.** Places locks on all data that is used in a query, preventing other users from updating the data. However, new phantom rows can be inserted into the data set by another user and are included in later reads in the current transaction. **Serializable.** Places a range lock on the data set, preventing other users from updating or inserting rows into the data set until the transaction is complete. This is the most restrictive of the four isolation levels. Because concurrency is lower, use this option only if necessary.

Multidimensional Connection Pool Properties in the General Tab

This section describes connection pool properties in the General tab that are specific to multidimensional data sources.

Figure 7–2 shows the General tab of the Connection Pool dialog, for an Essbase data source.

Figure 7–2 General Tab of the Connection Pool Dialog: Essbase Data Source

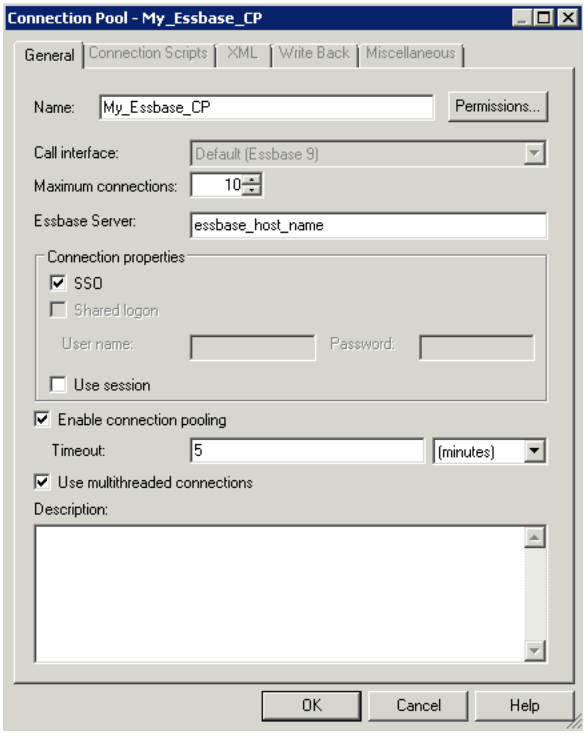


Table 7–5 describes the properties in the General tab of the Connection Pool dialog that are specific to multidimensional data sources. Note that some properties only appear for certain types of multidimensional data sources.

Table 7–5 Connection Pool Properties in the General Tab: Multidimensional Data Source Properties

Property	Description
URL	This property is only displayed for XMLA data sources. Specify the URL to connect to the XMLA provider. This URL points to the XMLA virtual directory of the computer hosting the cube. This virtual directory must be associated with msxisapi.dll (part of the Microsoft XML for Analysis SDK installation). For example, the URL might look like the following: http://SDCDL360i101/xmla/msxisap.dll
Essbase Server	This property is only displayed for Essbase data sources. Specify the host name of the computer where the Essbase Server is running. If the Essbase Server is running on a non-default port, or if it is part of an Essbase Cluster, you must include the port number in the **Essbase Server** field, in the format `hostname:port`. **Note:** You can import metadata from an Essbase Cluster, but you must still specify an individual Essbase Server host name and port number in the **Essbase Server** field.

Table 7–5 (Cont.) Connection Pool Properties in the General Tab: Multidimensional Data Source Properties

Property	Description
SSO	This property is only displayed for Essbase, Hyperion Financial Management, and Hyperion Planning data sources. For Essbase, select this option if you want Essbase to be able to enforce security policies that provide different cube access or member-level access to different users. If you select this option then you must also select the **Shared logon** option. Do not select this option if all users are expected to have the same access to the Essbase cube. In this case all the users will have the same access to the cube based on the shared credentials specified in the connection pool. If you do not select this option then you must also select the **Shared logon** option. For Hyperion Financial Management or Hyperion Planning, select this option and be sure that the **Shared logon** option is *unchecked* to authenticate against Hyperion Financial Management or Hyperion Planning using a shared token, rather than using a set of shared credentials in the connection pool. If you select this option, you should also select **Virtual Private Database** in the corresponding database object to protect cache entries. See Table 7–1 for more information.
Shared logon	This property is only displayed for Essbase, Hyperion Financial Management, and Hyperion Planning data sources. For all Essbase data sources, it is required that you select this option. See "Configuring Essbase to Use a Shared Logon" for more information about using this option in Essbase Connection Pools. For Hyperion Financial Management or Hyperion Planning, you will set this option based on how you set the **SSO** property. ■ If you checked the **SSO** property, then do not check this option. Not checking this option causes authentication against Hyperion Financial Management or Hyperion Planning using a shared token, rather than using a set of shared credentials in the connection pool. ■ If you did not check the **SSO** property, then check this option. This will cause Oracle BI Server to use the same shared logon credentials to connect to the data source for all Oracle BI users. All users will share the same access to the data source.
Data Source Information: Data Source	Specify the vendor-specific information used to connect to the multidimensional data source. Consult your multidimensional data source administrator for setup instructions because specifications can change. For example, if you use v1.0 of the XML for Analysis SDK, then the value should be `Provider-MSOLAP;Data Source-local`. If you use v1.1, then it should be `Local Analysis Server`.
Data Source Information: Catalog	Specify the list of catalogs available, if you imported data from your data source. The cube tables correspond to the catalog you use in the connection pool.
System IP or Hostname	This property is only displayed for SAP/BW data sources. Provide the host name or IP address of the SAP data server. This field corresponds to the parameter `ashost` in the SAP/BW connect string.

Table 7–5 (Cont.) Connection Pool Properties in the General Tab: Multidimensional Data Source Properties

Property	Description
System Number	This property is only displayed for SAP/BW data sources. Provide the SAP system number. This is a two-digit number assigned to an SAP instance, also called Web Application Server, or WAS. This field corresponds to the parameter `sysnr` in the SAP/BW connect string.
Client Number	This property is only displayed for SAP/BW data sources. Provide the SAP client number. This is a three-digit number assigned to the self-contained unit called Client in SAP. A Client can be a training, development, testing, or production client, or it can represent different divisions in a large company. This field corresponds to the parameter `client` in the SAP/BW connect string.
Language	This property is only displayed for SAP/BW data sources. Provide the SAP language code used when logging in to the data source (for example, EN for English or DE for German). This field corresponds to the parameter `lang` in the SAP/BW connect string.
Additional Parameters	This property is only displayed for SAP/BW data sources. Optionally, provide additional connection string parameters in the format param=value. Delimit multiple parameters with a colon.
Use session	An option that controls whether queries go through a common session. Consult your multidimensional data source administrator to determine whether this option should be enabled. Default is Off (not selected).

Setting Connection Pool Properties in the Connection Scripts Tab

You can create connection scripts and set them to be run before the connection is established, before a query is run, after a query is run, or after the connection is disconnected. For example, you can create a connection script that, on connect, inserts the name of the user and the connection time into a table.

This topic describes the properties in the Connection Scripts tab of the Connection Pool dialog. The Connection Scripts tab is available for ODBC, OCI, Oracle OLAP, ADF, and DB2 data sources.

Connection scripts can contain any commands accepted by the database, such as a command to turn on quoted identifiers. In a mainframe environment, a script could be used to set the secondary authorization ID when connecting to DB2 to force a security exit to a mainframe security package such as RACF. This enables mainframe environments to maintain security in one central location.

Because the connection script is sent directly to the data source, the script should use native SQL or another language understood by the data source, not Oracle BI Server Logical SQL.

To create connection scripts for data sources:

- In the Connection Pool dialog, click the Connection Scripts tab, and then complete the fields using the information in Table 7–6.

  To enter a new connection script, click **New** next to the appropriate script type. Then, enter or paste the SQL statements for the script and click **OK**.

You can edit existing scripts by clicking the ellipsis button to launch the Physical SQL window. Use the Up Arrow and Down Arrow buttons to reorder existing scripts.

Click **Delete** to remove a script.

Figure 7–3 shows the Connection Scripts tab of the Connection Pool dialog.

Figure 7–3 Connection Scripts Tab of the Connection Pool Dialog

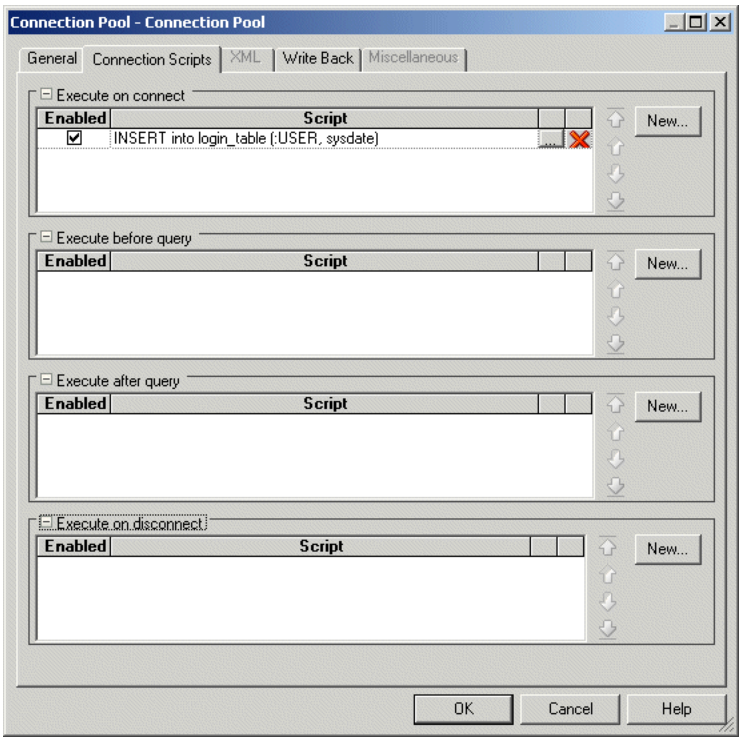


Table 7–6 describes the properties in the Connection Scripts tab of the Connection Pool dialog.

Table 7–6 Connection Pool Properties in the Connection Scripts Tab

Property	Description
Execute on connect	Contains SQL queries that are executed before the connection is established.
Execute before query	Contains SQL queries that are executed before the query is run.
Execute after query	Contains SQL queries that are executed after the query is run.
Execute on disconnect	Contains SQL queries that are executed after the connection is closed.

Setting Connection Pool Properties in the XML Tab

Use the Connection Pool Properties in the XML tab to set properties for XML and XML Server data sources.

Caution: The XML tab of the Connection Pool dialog provides the same functionality as the XML tab of the Physical Table dialog. However, the properties in the XML tab of the Physical Table dialog override the corresponding settings in the Connection Pool dialog.

To set connection pool properties for XML data sources:

- In the Connection Pool dialog, click the XML tab, and then complete the fields using the information in Table 7–7.

Figure 7–4 shows the XML tab of the Connection Pool dialog.

Figure 7–4 XML Tab of the Connection Pool Dialog

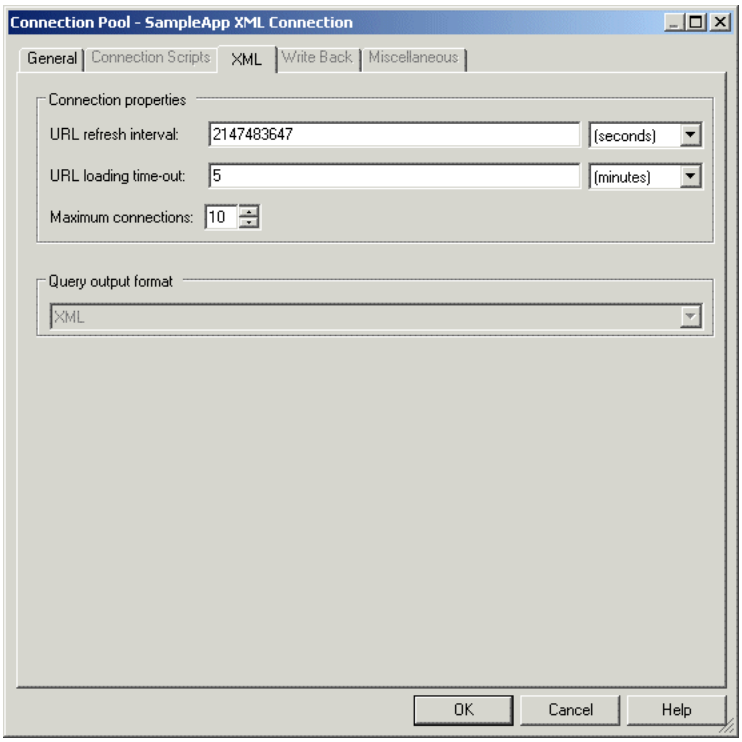


Table 7–7 describes the properties in the XML tab of the Connection Pool dialog.

Table 7–7 Connection Pool Properties in the XML Tab

Property	Description
Connection method: Search script	This property is only displayed for XML Server data sources. Click **Browse** to locate the appropriate search script.

Table 7–7 (Cont.) Connection Pool Properties in the XML Tab

Property	Description
Connection properties: URL refresh interval	This property is used for XML data sources and is not available for XML Server data sources. The refresh interval is analogous to setting cache persistence for database tables. The URL refresh interval is the time interval after which the XML data source is queried again directly rather than using results in cache. The default setting is infinite, meaning the XML data source is never refreshed. If you specified a URL to access the data source, set the URL refresh interval. ■ Select a value from the list (Infinite, Days, Hours, Minutes or Seconds). ■ Specify a whole number as the numeric portion of the interval.
Connection properties: URL loading time-out	The timeout interval for queries. The default is 15 minutes. If you specified a URL to access the data source, set the URL loading time-out as follows: ■ Select a value from the list (Infinite, Days, Hours, Minutes or Seconds). ■ Specify a whole number as the numeric portion of the interval.
Connection properties: Maximum connections	The maximum number of connections. The default is 10.
Query input supplements: Header file/Trailer file	This property is only displayed for XML Server data sources. Click **Browse** to locate the header and trailer files.
Query output format	For XML data sources, choose only **XML**. Other output formats are available for XML Server data sources.

Setting Connection Pool Properties in the Write Back Tab

Use the Write Back tab to set write back properties for ODBC, OCI, Oracle OLAP, ADF, and DB2 data sources.

To set write-back properties for data sources:

- In the Connection Pool dialog, click the Write Back tab, and then complete the fields using the information in Table 7–8.

Figure 7–3 shows the Write Back tab of the Connection Pool dialog.

Figure 7–5 Write Back Tab of the Connection Pool Dialog

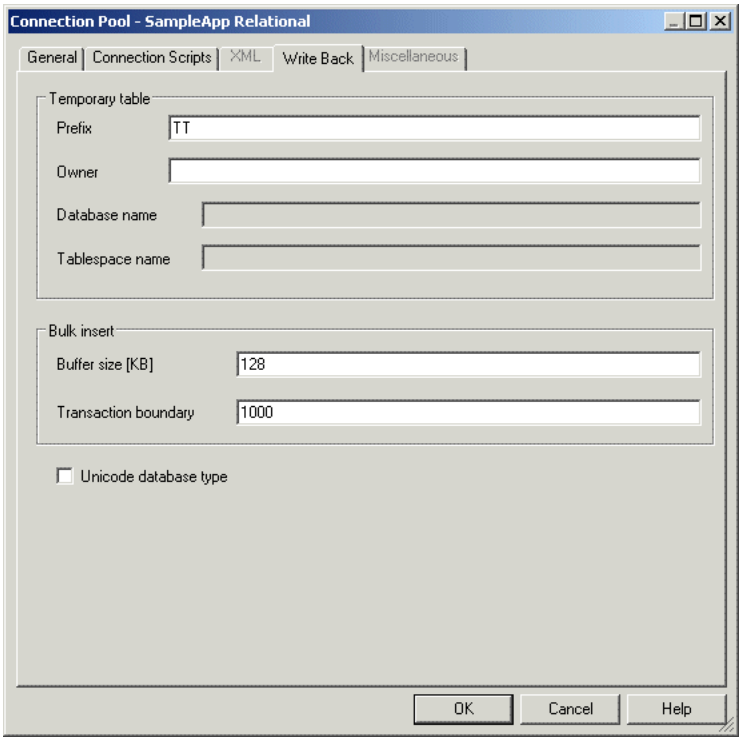


Table 7–8 describes the properties in the Write Back tab of the Connection Pool dialog.

Table 7–8 Connection Pool Properties in the Write Back Tab

Property	Description
Temporary table: Prefix	When the Oracle BI Server creates a temporary table, these are the first two characters in the temporary table name. The default value is `TT`.
Temporary table: Owner	Table owner name used to qualify a temporary table name in a SQL statement, for example to create the table `owner.tablename`. If left blank, the user name specified in the writeable connection pool is used to qualify the table name and the **Shared logon** field on the General tab should also be set.
Temporary table: Database name	Database where the temporary table will be created. This property applies only to IBM OS/390 because IBM OS/390 requires database name qualifier to be part of the `CREATE TABLE` statement. If left blank, OS/390 defaults the target database to a system database for which the users may not have Create Table privileges.
Temporary table: Tablespace name	Tablespace where the temporary table will be created. This property applies to OS/390 only as OS/390 requires tablespace name qualifier to be part of the `CREATE TABLE` statement. If left blank, OS/390 defaults the target database to a system database for which the users may not have Create Table privileges.
Bulk insert: Buffer size (KB)	Used for limiting the number of bytes each time data is inserted in a database table. For optimum performance, consider setting this parameter to 128. See "About Setting the Buffer Size and Transaction Boundary" for additional information.

Table 7–8 (Cont.) Connection Pool Properties in the Write Back Tab

Property	Description
Bulk insert: Transaction boundary	Controls the batch size for an insert in a database table. For optimum performance, consider setting this parameter to 1000. See "About Setting the Buffer Size and Transaction Boundary" for additional information.
Unicode database type	Select this option when working with columns of an explicit Unicode data type, such as NCHAR, in a Unicode database. This makes sure that the binding is correct and that data is inserted correctly. Different database vendors provide different character data types and different levels of Unicode support. Use the following general guidelines to determine when to set this option: ■ On a database where CHAR data type supports Unicode and there is no separate NCHAR data type, do not select this option. ■ On a database where NCHAR data type is available, it is recommended to select this option. ■ On a database where CHAR and NCHAR data type are configured to support Unicode, selecting this option is optional. **Note:** Unicode and non-Unicode data types cannot coexist in a single non-Unicode database. For example, mixing the CHAR and NCHAR data types in a single non-Unicode database environment is not supported.

Setting Connection Pool Properties in the Miscellaneous Tab

Use the Miscellaneous tab of the Connection Pool dialog to set application properties for ADF, JDBC, and JNDI data sources.

This section contains the following topics:

- Application Properties for ADF Data Sources
- Application Properties for JDBC (Direct Driver) or JDBC (JNDI) Data Sources

Application Properties for ADF Data Sources

To set application properties for ADF data sources, in the Connection Pool dialog, click the Miscellaneous tab, and then complete the fields using the information in Table 7–9.

Figure 7–6 shows the Miscellaneous tab of the Connection Pool dialog.

Figure 7–6 Miscellaneous Tab of the Connection Pool Dialog

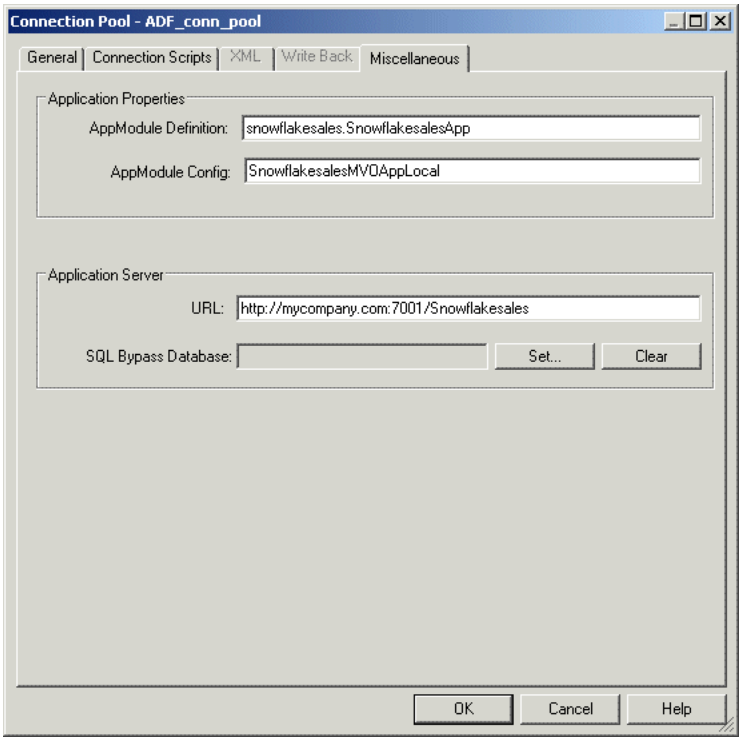


Table 7–9 describes the properties in the Miscellaneous tab of the Connection Pool dialog.

Table 7–9 Connection Pool Properties in the Miscellaneous Tab

Property	Description
AppModule Definition	The fully qualified Java package name of the Root Application Module to which you want to connect, such as `oracle.apps.fii.receivables.model.RootAppModule`.
AppModule Config	Determines which application configuration is used in the connection, such as `RootAppModuleShared`.

URL	The URL to the Oracle Business Intelligence broker servlet, in the format:
	`http://host:port/APP_DEPLOYMENT_NAME/obieebroker`
	
	For example:
	`http://localhost:7001/SnowflakeSalesApp/obieebroker`
	
	The URL is case-sensitive.

Table 7–9 (Cont.) Connection Pool Properties in the Miscellaneous Tab

Property	Description
SQL Bypass Database	(Optional) The name of the SQL Bypass database. The SQL Bypass database must be a physical database in the Physical layer of the repository. The database object for the SQL Bypass database must have a valid connection pool, with connection information that points to the same database that is being used by the JDBC Data source defined in the WebLogic Server. The SQL Bypass database does not need to have any tables under it. After a valid database name is supplied, the SQL Bypass feature is enabled for all queries. The SQL Bypass feature directly queries the database so that aggregations and other transformations are pushed down where possible, reducing the amount of data streamed and worked on in Oracle Business Intelligence. See "About Specifying a SQL Bypass Database" for more information.

Application Properties for JDBC (Direct Driver) or JDBC (JNDI) Data Sources

To set application properties for JDBC (Direct Driver) or JDBC (JNDI) data sources, in the Connection Pool dialog, click the Miscellaneous tab, and then complete the fields using the following information:

- **Required Cartridge Version** - This value defaults to 12.1.
- **Use SQL Over HTTP** - For JDBC (JNDI) call interface, only. If you are using Oracle BI Cloud Service, set this field to **false** to use HTTP to communicate between networks. For example, set this field to **false** if the BI Server and the data source you are accessing reside on different Oracle clouds.
- **Javads Server URL** - For JDBC (Direct Driver) call interface, only. The hostname and port that was specified in the Connect to Java Datasource Server dialog defaults into this field. This is the URL for the Java Datasource server, which supplies the Java metadata into the Physical layer.
- **Driver Class** - For JDBC (Direct Driver) call interface, only. Specify the driver to connect to the database. For example, the DB2 JDBC driver. The driver that you specify must be deployed in Oracle Weblogic Server.

  By default the Oracle JDBC driver (oracle.jdbc.OracleDriver) is available in Oracle Weblogic Server.

Setting Up Persist Connection Pools

A persist connection pool is a database property that is used for specific types of queries (typically used to support Marketing queries). In some queries, all of the logical query cannot be sent to the transactional database because that database might not support all of the functions in the query. This issue might be solved by temporarily constructing a physical table in the database and rewriting the Oracle BI Server query to reference the new temporary physical table.

You can use the persist connection pool in the following situations:

- **Populate stored procedures.** Use to rewrite the Logical SQL result set to a managed table. Typically used by Oracle's Siebel Marketing Server to write segmentation cache result sets.
- **Perform a generalized subquery.** Stores a nonfunction subquery in a temporary table, and then rewrites the original subquery result against this table. Reduces

data movement between the Oracle BI Server and the database, supports unlimited IN list values, and might result in improved performance.

In these situations, the user issuing the Logical SQL query must have been granted the Populate privilege on the target database.

The persist connection pool functionality designates a connection pool with write-back capabilities for processing this type of query. You can assign one connection pool in a single database as a persist connection pool. If this functionality is enabled, the user name specified in the connection pool must have the privileges to create DDL (Data Definition Language) and DML (Data Manipulation Language) in the database.

To assign a persist connection pool:

1. In the Physical layer of the Administration Tool, double-click the database object for which you want to assign a persist connection pool.
2. In the Database dialog, click the General tab.
3. In the **Persist connection pool** area, click **Set**.

   If there is only one connection pool, it appears in the **Persist connection pool** field.
4. If there are multiple connection pools, in the Browse dialog, select the appropriate connection pool, and then click **OK**.

   The selected connection pool name appears in the **Persist connection pool** field.
5. (Optional) To set write-back properties, click the Connection Pools tab.
6. In the connection pool list, double-click the connection pool.
7. In the Connection Pool dialog, click the Write Back tab.
8. Complete the fields using Table 7–8 as a guide. See also "About Setting the Buffer Size and Transaction Boundary" for additional information.
9. Click **OK**, then click **OK** again to save the persist connection pool.

To remove a persist connection pool:

1. In the Physical layer of the Administration Tool, double-click the database object that contains the persist connection pool you want to remove.
2. In the Database dialog, click the General tab.
3. In the **Persist connection pool** area, click **Clear**.

   The database name is replaced by `not assigned` in the **Persist connection pool** field.
4. Click **OK**.

About Setting the Buffer Size and Transaction Boundary

If each row size in a result set is 1 KB and the buffer size is 20 KB, then the maximum array size is 20 KB. If there are 120 rows, there are 6 batches with each batch size limited to 20 rows.

If you set **Transaction boundary** to 3, the server commits twice. The first time, the server commits after row 60 (3 * 20). The second time, the server commits after row 120. If there is a failure when the server commits, the server only rolls back the current transaction. For example, if there are two commits and the first commit succeeds but the second commit fails, the server only rolls back the second commit.

For optimum performance, consider setting the buffer size to 128 and the transaction boundary to 1000.

List Connection Pool Command

Use the list connection pool command `listConnectionpool` to create a list of connection pools in JSON format for a specific service instance. Use this and the `updateConnectionpool` utility when you need to update more than one connection pool.

You execute the utility through a launcher script, data-model-cmd.sh on UNIX and data-model-cmd.cmd on Windows. You can find the launcher script at the following location:

Oracle_Home`/user_projects/domains/bi/bitools/bin`

See "What You Need to Know Before Using the Command" for more information.

Syntax

The `listConnectionpool` command takes the following parameters:

```
listConnectionpool -SI <service_instance> -U <cred_username> [-P <cred_
password>] [-S <hostname>] [-N <port_number>] [-V <true/false>] [-O
<outputFile.json>] [-SSL] [-H]
```

Where

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for Oracle BI EE authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you do not supply the password, then you will be prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you are using automated scripting to run the command.

`S` specifies the Oracle BI EE host name. Only include this option when you are running the command from a client installation.

`N` specifies the Oracle BI EE port number. Only include this option when you are running the command from a client installation.

`V` specifies whether to include repository variables used in the connection pool. Note that the default is `false`.

`O` specifies the output file name with the .json suffix.

`SSL` specifies to use SSL to connect to the WebLogic Server to run the command. Only include this option when you are running the command from a client installation.

`H` displays the usage information and exits the command.

Example

```
data-model-cmd.sh listConnectionpool -SI bi -U weblogic -P password -S
server1.example.com -N 7777  -SSL -V true -O output.json
```

Sample JSON List Connection Pool Output

```
{
    "Title":"List Connection Pools",
   "Conn-Pool-Info":[
```

```
        {
            "uid":"80ca62c5-0bd5-0000-714b-e31d00000000",
            "connPool":"SampleApp_Lite_Xml",
            "parentName":"\"Sample App Lite Data\"",
            "user":"VALUEOF(REPO_STATIC_VAR)_Tushar",
            "password":"B25F85BC2A170AD4349DEF26E4D1295D
7C2E35213306F12832914CBE7A9DD95561D771DED06484112B1FC6F27B6D0D58",
            "dataSource":"VALUEOF(NQ_
SESSION.SERVICEINSTANCEROOT)/data/SampleAppLite"
        }
    ],
    "Variables-In-Conn-Pool":[
        {
            "uid":"40000000-3c25-155b-991a-0af2537d0000",
            "variable":"REPO_STATIC_VAR",
            "value":"'RepoStaticVariable'"
        }
    ]
}
```

Update Connection Pool Command

Use the update connection pool command `updateConnectionpool` to upload a modified JSON file containing updated connection pool values to a specific server instance. Use this and the `listConnectionpool` utility when you need to update more than one connection pool.

Use the `listConnectionpool` command to create a JSON file containing a list of connection pools for a specific service instance. Modify the connection pool information in this file and then upload it to the service instance using the `updateConnectionpool` command. Note that you must not modify the `uid` and `connPool` values in the file. See "List Connection Pool Command" for more information.

You execute the utility through a launcher script, data-model-cmd.sh on UNIX and data-model-cmd.cmd on Windows. You can find the launcher script at the following location:

Oracle_Home`/user_projects/domains/bi/bitools/bin`

See "What You Need to Know Before Using the Command" for more information.

Syntax

The `updateConnectionpool` command takes the following parameters:

```
updateConnectionpool -C <connectionpoolList.json> -SI <service_instance>
-U <cred_username> [-P <cred_password>] [-S <hostname>] [-N <port_number>]
[-SSL] [-H]
```

Where

`C` specifies the name of the modified JSON file that you want to upload. Note this file must not contain modified `uid` and `connPool` values. See "Sample JSON List Connection Pool Output" for information about the correct syntax for the update connection pool input file.

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for Oracle BI EE authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you do not supply the password, then you will be prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you are using automated scripting to run the command.

`S` specifies the Oracle BI EE host name. Only include this option when you are running the command from a client installation.

`N` specifies the Oracle BI EE port number. Only include this option when you are running the command from a client installation.

`SSL` specifies to use SSL to connect to the WebLogic Server to run the command. Only include this option when you are running the command from a client installation.

`H` displays the usage information and exits the command.

Example

```
data-model-cmd.sh updateConnectionpool -C connpool.json -SI bi -U weblogic
-P password -S server1.example.com -N 7777 -SSL
```

Using the BIServerT2PProvisioner.jar Utility to Change Connection Pool Passwords

When moving your Oracle BI repository from one environment to another, you often need to change connection pool information for data sources, because the connection information in one environments is typically different from the connection information in another environments.

> **Note:** Although you can use BIServerT2PProvisioner.jar to update connection pool passwords, Oracle's preferred method is the updateConnectionpool command. See "Update Connection Pool Command"for information.

Connection pool passwords are encrypted and stored inside the encrypted repository file. Because of this, plain-text passwords must first be encrypted before they can be applied to an Oracle BI repository.

You can use the BIServerT2PProvisioner.jar utility to programmatically change and encrypt connection pool passwords in a repository. Note that the utility only works with repositories in RPD format; you cannot use the utility with MDS XML-format repositories. In addition, the utility requires JDK 1.6.

To use the BIServerT2PProvisioner.jar utility to change connection pool passwords:

The location of BIServerT2PProvisioner.jar is:

Oracle_Home/bi/bifoundation/server

1. Run BIServerT2PProvisioner.jar using the `-generate` option to generate a template file where you can input the new passwords, as follows:

   ```
   java -jar ORACLE_HOME/bifoundation/server/bin/BIServerT2PProvisioner.jar
   -generate repository_name -output password_file
   ```

   Where:

   repository_name is the name and path of the Oracle BI repository that contains the connection pools for which you want to change passwords.

password_file is the name and path of the output password text file. This file will contain the connection pool names from the specified repository.

Then, enter the repository password when prompted.

For example:

```
java -jar BIServerT2PProvisioner.jar -generate original.rpd –output
inputpasswords.txt
Enter the repository password: My_Password
```

2. Edit the password file to replace `<Change Password>` with the updated password for each connection pool. A sample password file might appear as follows:

```
"SQLDB_UsageTracking"."UTCP" = <Change Password>
"SQLDB_Data"."Db Authentication Pool" = <Change Password>
```

Tip: Make sure to only edit the text to the right of the equals sign. If you change the text to the left of the equals sign, then the syntax for the connection pool names will be incorrect.

Save and close the password file when your edits are complete.

3. Run BIServerT2PProvisioner.jar again with the `-passwords` option, as follows:

```
java -jar BIServerT2PProvisioner.jar -passwords password_file
-input input_repository -output output_repository
```

Where:

password_file is the name and path of the text file that specifies the connection pools and their corresponding changed passwords.

input_repository is the name and path of the Oracle BI repository where you want to apply the changed passwords.

output_repository is the name and path of the output repository that contains the updated passwords.

Then, enter the repository password when prompted.

For example:

```
java -jar BIServerT2PProvisioner.jar -passwords inputpasswords.txt -input
original.rpd -output updated.rpd
Enter the repository password: My_Password
```

4. Oracle does not recommend leaving clear-text passwords available on the system. Instead, either delete the input password file completely, or encrypt it so that it cannot be viewed.

8

Working with Physical Tables, Cubes, and Joins

This chapter explains how to work with objects in the Physical layer of the Oracle BI repository, and describes Oracle Essbase, Hyperion Financial Management, and Oracle OLAP representations in the Physical layer. It also explains other Physical layer concepts like opaque views, hints, row counts, physical layer folders, and how to use the Physical Diagram.

This chapter contains the following topics:

About Working with the Physical Layer

The Physical layer of the Oracle BI repository contains objects that represent physical data constructs from back-end data sources. The Physical layer defines the objects and relationships available to the Oracle BI Server for writing physical queries. This layer encapsulates data source dependencies to enable portability and federation.

Each data source of the repository model typically has its own discrete physical model in the Physical layer. The top-level object in the Physical layer is a "database," and the type of database determines which features and rules apply to that physical model. For example, a relational database such as "Oracle 11g" has relational objects such as physical tables and joins. In contrast, a multidimensional source such as "Essbase 9"

has cube tables and physical hierarchies. Therefore, some sections of this chapter apply to only certain database types.

Physical tables, cubes, joins, and other objects in the Physical layer are typically created automatically when you import metadata from the data sources. After these objects have been imported, you can perform tasks such as create additional join paths that are not in the data source, create alias tables for physical tables that need to serve in different roles, and adjust properties of physical hierarchies from multidimensional data sources.

Working with the Physical Diagram

In addition to working with Physical layer objects in the right pane of the Administration Tool, you can open the Physical Diagram view to access a graphical model of tables and joins.

> **Note:** The Physical Diagram is typically used with relational and XML sources rather than multidimensional sources. Although the Physical Diagram view for a multidimensional source does display a denormalized table representation of a cube table, the primary means of working with a multidimensional physical model is by working in the physical tree using dimensions, hierarchies, and columns.

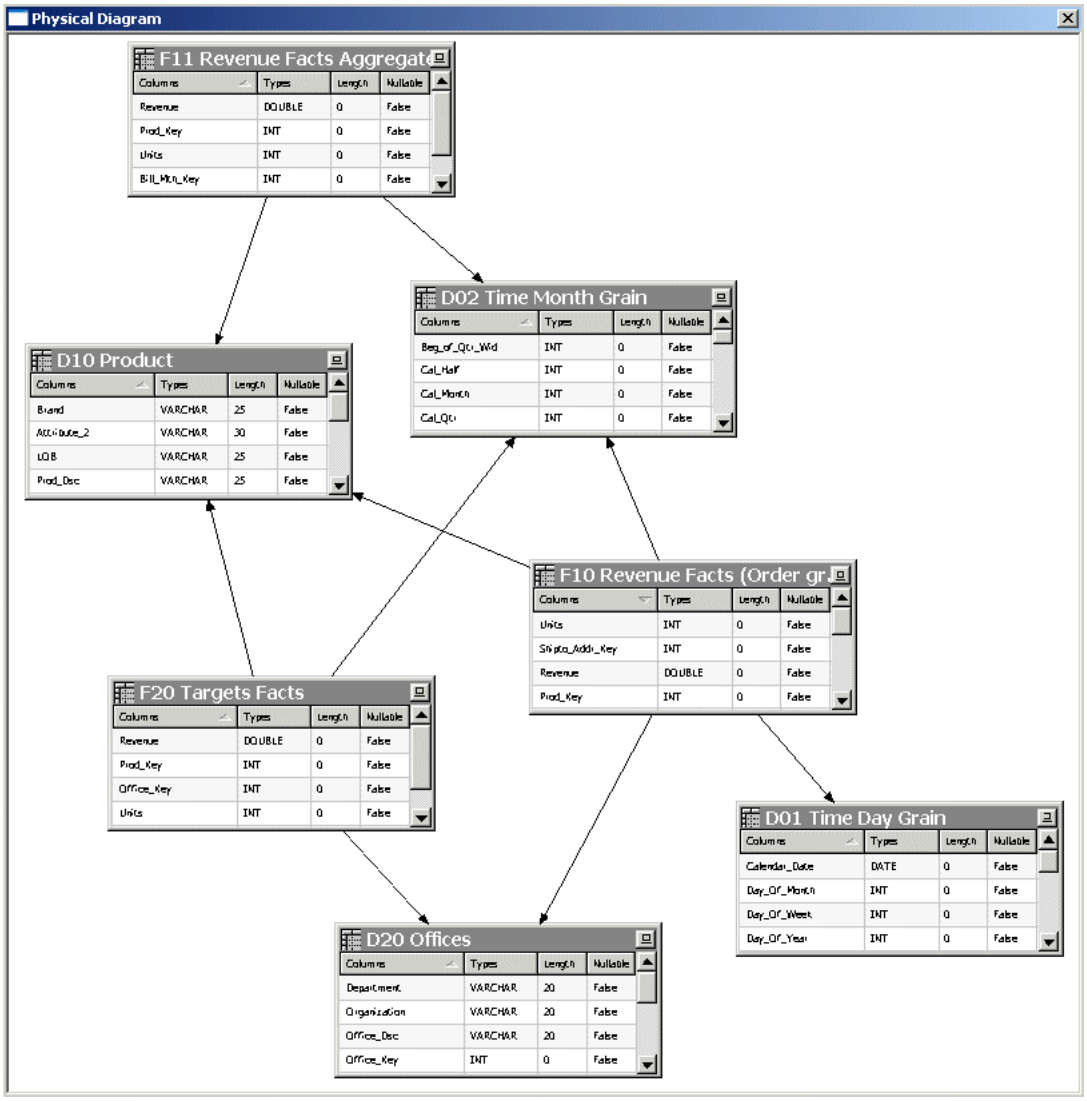
To access the Physical Diagram, right-click an object in the Physical layer tree view (such as a physical database or table) and select **Physical Diagram**. Then, select one of the following options:

- **Selected Object(s) Only.** Displays only the selected objects. Joins appear only if they exist between the objects that you select.
- **Object(s) and Direct Joins.** Displays the selected objects and any tables that join to the objects that you select.
- **Object(s) and All Joins.** Displays the selected objects, and each object that is related directly or indirectly to the selected object through some join path. If all the objects in a schema are related, then using this option diagrams every table, even if you only select one table.

Note that the Physical Diagram displays only physical tables and joins. It does not display other Physical layer objects, such as connection pools, physical hierarchies, or levels.

Figure 8–1 shows the Physical Diagram.

Figure 8–1 Physical Diagram



You can also open the Physical Diagram by selecting one or more objects in the tree view and clicking the **Physical Diagram** button on the toolbar:

Only the objects you selected appear. Joins appear only if they exist between the selected objects. Joins are represented by a line with an arrow at the "one" end of the join.

To help you better understand the model's logical-to-physical mappings, you can view the physical objects that are associated with a particular logical object by selecting one or more business models, logical tables, or logical table sources in the Business Model and Mapping layer tree view and clicking the **Physical Diagram** button on the toolbar. Only physical objects that are related to the objects you selected appear. You can view the same information by right-clicking a logical object and selecting **Objects and Direct Join(s) within Business Model** from the Physical Diagram submenu. You can also choose one of the other Physical Diagram display options.

To add tables to the Physical Diagram, leave the Physical Diagram window open and right-click the table or tables you want to add. Then, select **Physical Diagram** and choose one of the display options.

Additional options are available in the right-click menu for the graphical tables and joins displayed in the Physical Diagram. For example, you can delete objects or view their properties, or you can add related objects using the right-click options **Add Direct Joins**, **Add Tables Joined to Whole Selection**, and **Add All Joins**. You can also select **Find in Tree View** to locate a particular object in the Physical layer tree view in the right pane, or check out objects in online mode.

You can also right-click an object in the Physical Diagram view and select **Hide** to hide particular objects in the diagram. Note that this effect is temporary and does not persist.

Use the **Print** and **Print Preview** options on the File menu to manage printing options for the Physical Diagram. You can also use the **Print** option on the toolbar.

See also the following sections:

- "Using the Physical and Business Model Diagrams" for information about zooming, panning, and controlling the layout of the tables
- "Defining Physical Joins with the Physical Diagram" for information about defining physical joins

Creating Physical Layer Folders

Use folders to organize the contents of the Physical layer.

This section contains the following topics:

Creating Physical Layer Catalogs and Schemas

Physical layer catalogs are optional ways to group different schemas. A catalog contains all the schemas (metadata) for a physical database object. A schema contains only the metadata information for a particular user or application.

Note the following:

- You must create a physical database object before you can create a physical catalog object or a physical schema object.
- After you implement a certain type of grouping, you cannot change it later. For example, if you decide to implement database then schema then table, you cannot add a catalog afterward.

Creating Catalogs

In the Physical layer of a large repository, administrators can create physical catalogs that contain one or more physical schemas.

To create a catalog:

1. In the Physical layer of the Administration Tool, right-click a physical database and select **New Object**, then select **Physical Catalog**.

2. In the Physical Catalog dialog, type a name for the catalog.
3. Type a description for the catalog and click **OK**.

Creating Schemas

The schema object contains tables and columns for a physical schema. Schema objects are optional in the Physical layer of the Administration Tool.

To create a schema:

1. In the Physical layer of the Administration Tool, right-click a physical database or physical catalog and select **New Object**, then select **Physical Schema**.
2. In the Physical Schema dialog, type a name.
3. Type a description for the schema and click **OK**.

Using a Variable to Specify the Name of a Catalog or Schema

You can use a variable to specify the names of catalog and schema objects. For example, you have data for multiple clients and you structured the data source so that data for each client was in a separate catalog. You would initialize a session variable named Client, for example, that could be used to set the name for the catalog object dynamically when a user signs on to the Oracle BI Server.

You specify the session variable to use in the Dynamic Name tab of the Physical Catalog or Physical Schema dialog.

Note: The Dynamic Name tab is not active unless at least one session variable is defined.

To specify the session variable to use in the Dynamic Name tab:

1. In the **Name** column of the Dynamic Name tab, click the name of the session variable that you want to use. The initial value for the variable (if any) is shown in the **Default Initializer** column.
2. To select the highlighted variable, click **Select**.

   The name of the variable is displayed in the dynamic name field, and the **Select** button toggles to the **Clear** button.

To remove assignment for a session variable in the Dynamic Name tab:

- Click **Clear** to remove the assignment for the variable as the dynamic name.

  The value **not assigned** is displayed in the dynamic name field, and the **Clear** button toggles to the **Select** button.

To sort column entries in the Dynamic Name tab:

- Click the **Name** or **Default Initializer** column heading to sort the entries in a column. Clicking a column heading toggles the order of the entries in that column between ascending and descending order, according to the column type.

Setting Up Display Folders in the Physical Layer

You can create display folders to organize table objects in the Physical layer. They have no effect on query processing. After you create a display folder, the selected tables appear in the folder as a shortcut and in the Physical layer tree as an object. You can

hide the objects so that you only view the shortcuts in the display folder. See the information about the Repository tab of the Options dialog in "Setting Administration Tool Options" for more information about hiding these objects.

> **Note:** Deleting a table in a display folder deletes only the shortcut to that object. When you delete a column in a display folder, however, the column is actually deleted.

To set up a physical display folder:

1. In the Physical layer of the Administration Tool, right-click a physical database and select **New Object**, then select **Physical Display Folder**.
2. In the Physical Display Folder dialog, type a name for the folder.
3. To add tables to the display folder, click **Add**. Then, in the Browse dialog, select the fact or physical tables you want to add to the folder and click **Select**.

   Alternatively, you can drag one or more physical tables to the display folder after you close the dialog.
4. Click **OK**.

Working with Physical Tables

This section explains the different things you can do with physical table objects in the Physical layer of the Oracle BI repository. Both physical tables from relational data sources and physical cube tables from multidimensional data sources use the Physical Table table type.

Many of the tasks described in this section apply to relational and multidimensional data sources. See also "Working with Multidimensional Sources in the Physical Layer" for additional information specific to multidimensional data sources.

This section contains the following topics:

- About Tables in the Physical Layer
- About Physical Alias Tables
- Creating and Managing Physical Tables and Physical Cube Tables
- Creating and Managing Columns and Keys for Relational and Cube Tables
- Viewing Data in Physical Tables or Columns

About Tables in the Physical Layer

A physical table is an object in the Physical layer of the Oracle BI repository that corresponds to a table in a data source. Metadata for physical tables is usually imported from the data source. This metadata enables the Oracle BI Server to access the data source tables with SQL requests.

When you delete a physical table, all dependent objects are deleted (for example, columns, keys, and foreign keys). When you delete a physical cube table, hierarchies are also deleted. The deletion fails if an alias exists on the physical table.

In addition to importing data source tables into the Physical layer, you can create virtual physical tables in the Physical layer, using values in the **Table Type** field in the Physical Table dialog. Creating virtual tables can provide the Oracle BI Server and the

underlying data sources with the proper metadata to perform some advanced query requests.

A virtual physical table can be a stored procedure, or a SELECT statement. A virtual physical table created from a SELECT statement is also called an opaque view. You can define an opaque view, and deploy it in the data source to create a deployed view. See "Deploying Opaque Views" for more information.

Use the **Table Type** list in the General tab of the Physical Table dialog to specify the physical table object type. Table 8–1 describes the available object types.

Table 8–1 Table Types for Physical Tables

Table Type	Description
Physical Table	Specifies that the physical table object represents a data source table. This table type is used for both relational physical tables and multidimensional cube tables.

Stored Proc	Specifies that the physical table object is a stored procedure. When you select this option, you type the stored procedure in the text box. Requests for this table will call the stored procedure.
	
	For stored procedures that are data source-specific, select **Use database specific SQL**. When you select this option, the **Database** column displays supported data sources by brand, with Default as the root. You can enter data source-specific initialization strings by selecting the database type on the left and entering the corresponding string on the right. The initialization string for the Default option is run when the queried database type does not have a corresponding database-specific string defined.
	
	Stored procedures within an Oracle Database do not typically return result sets. Therefore, they cannot be initiated from within Oracle Business Intelligence. You need to rewrite the procedure as an Oracle function, use it in a SELECT statement in the Administration Tool initialization block, and associate it with the appropriate Oracle BI Server session variables in the Session Variables dialog.
	
	The following example shows a SQL initialization string using the GET_ROLES function that is associated with the USER, GROUP, and DISPLAYNAME variables. The function takes a user Id as a parameter and returns a semicolon-delimited list of group names:
	
	`SELECT user_id, get_roles(user_id), first_name || ' ' || last_name`
	`FROM csx_security_table`
	`WHERE user_id = ':USER' and password = ':PASSWORD'`

Table 8–1 (Cont.) Table Types for Physical Tables

Table Type	Description
Select	Specifies that the physical table object is a `SELECT` statement. When you select this option, you type the `SELECT` statement in the text field, and you also need to manually create the table columns. The column names must match the ones specified in the `SELECT` statement. Column aliases are required for advanced SQL functions, such as aggregates and `CASE` statements. Requests for this table will execute the `SELECT` statement. For `SELECT` statements that are data source-specific, select **Use database specific SQL**. When you select this option, the **Database** column displays supported data sources by brand, with Default as the root. You can enter data source-specific initialization strings by selecting the database type on the left and entering the corresponding string on the right. The initialization string for the Default option is run when the queried database type does not have a corresponding database-specific string defined. If you are using Physical SQL to deploy an opaque view, then you must use the VALUELISTOF function. This type of table is also called an opaque view. See "Deploying Opaque Views" for more information.

About Physical Alias Tables

An alias table (alias) is a physical table that references a different physical table as its source (called the original table). Alias tables can be an important part of designing a Physical layer because they enable you to reuse an existing table more than once, without having to import it several times.

There are two main reasons to create an alias table:

- To set up multiple tables, each with different keys, names, or joins, when a single data source table needs to serve in different semantic roles. Setting up alias tables in this case is a way to avoid triangular or circular joins.

  For example, an order date and a shipping date in a fact table may both point to the same column in the time dimension data source table, but alias the dimension table so that each role is presented as a separately labeled alias table with a single join. These separate roles carry over into the business model, so that "Order Date" and "Ship Date" are part of two different logical dimensions. If a single logical query contains both columns, the physical query uses aliases in the SQL statement so that it can include both of them.

  You can also use aliases to enable a data source table to play the role of both a fact table, and a dimension table that joins to another fact table (often called a "fan trap").

- To include best practice naming conventions for physical table names. For example, you can prefix the alias table name with the table type (such as fact, dimension, or bridge), and not change the original physical table names. Some organizations create alias tables for all physical tables to enforce best practice naming conventions. In this case, all mappings and joins are based on the alias tables rather than the original tables.

Alias table names appear in physical SQL queries. Using alias tables to provide meaningful table names can make SQL queries referencing those tables easier to read. For example:

```
WITH
SAWITH0 AS (select sum(T835.Dollars) as c1
from
     FactsRevT835/*AllRevenue(Billed Time Join)*/)
select distinct 0 as c1,
     D1.c1 as c2
from
     SAWITH0 D1
order by c1
```

In this query, the meaningful alias table name "A11 Revenue (Billed Time Join)" has been applied to the terse original physical table name "FACTSREV." In this case, the alias table name provides information about which role the table was playing each time it appears in SQL queries.

Alias tables can have cache properties that differ from their original tables. To set different cache properties for an alias table, select the option **Override Source Table Caching Properties** in the Physical Table dialog for the alias table. In alias tables, columns cannot be added, deleted, or modified. Because columns are automatically synchronized, no manual intervention is required.

Synchronization ensures that the original tables and their related alias tables have the same column definitions. For example, if you delete a column in the original table, the column is automatically removed from the alias table.

You cannot delete an original table unless you delete all its alias tables first. Alternatively, you can select the original table and all its alias tables and delete them at the same time.

You can change the original table of an alias table, if the new original table is a superset of the current original table. However, this could result in an inconsistent repository if changing the original table deletes columns that are being used. If you attempt to do this, a warning message appears to let you know that this could cause a problem and lets you cancel the action. Running a consistency check identifies orphaned aliases.

When you edit a physical table or column in online mode, all alias tables and columns must be checked out. The behavior of online checkout uses the following conventions:

- If an original table or column is checked out, all its alias tables and columns are checked out.
- If an alias table or column is checked out, its original table and column are checked out.
- The checkout option is available for online repositories (if not read-only) and for all original and alias tables and columns.

Alias tables inherit some properties from their original tables. A property that is proxied is a value that is always the same as the original table, and cannot be changed. (In other words, the proxied properties are the ones that are dimmed in the alias table dialog.) If the original table changes its value for that particular property, the same change is applied on the alias table.

The following is a list of the properties that are proxied:

- Cacheable (the inherited property can be overridden)
- Cache never expires and Cache persistence time (the inherited properties can be overridden)
- Row Count

- Last Updated
- Table Type
- External Db Specifications

The following is a list of the properties that are not proxied:

- Name
- Description
- Display Folder Containers
- Foreign Keys
- Columns

> **Note:** Alias tables and original tables never share columns. Aliases and original tables have distinctly different columns that alias each other.

- Table Keys
- Complex Joins
- Source Connection Pool
- Polling Frequency
- All XML attributes

Creating and Managing Physical Tables and Physical Cube Tables

Use the General tab of the Physical Table dialog to create or edit physical tables and physical cube tables in the Physical layer of the Administration Tool.

This section contains the following topics:

Creating or Editing Physical Tables

You can create or edit the general properties for a table, including both relational physical tables and physical cube tables.

To create a physical table or edit general properties for tables:

1. In the Physical layer of the Administration Tool, perform one of the following steps:
   - To create a physical table, right-click the physical database or physical catalog and select **New Object**, then select **Physical Table**.

     If the database object has physical schemas defined, right-click the physical schema and select **New Physical Table**.
   - To create a physical cube table for a multidimensional data source, right-click the physical database and select **New Object**, then select **Cube Table**.

Caution: It is strongly recommended that you import cube tables, not create them manually.

– To edit an existing physical table, double-click the physical table object in the Physical layer.

2. In the Physical Table dialog, complete the fields using Table 8–2 as a guide.

Table 8–2 General Properties for Physical Tables

Property	Description
Name	The name of the physical table.
Table Type	Physical Table values: Physical Table, Stored Proc (stored procedure), or Select. Physical Cube Table values: Physical Table or Select. See Table 8–1 for more information.
Use Dynamic Name	Select this option to use a session variable to specify the physical table name, similar to catalog and schema objects. This option is available for non-multidimensional data source tables when you select a table type of **Physical Table**. You might want to choose this option if you have a multi-tenancy implementation and you want to define a separate physical table name for each customer. Another example would be to select between primary and shadow tables that are valid at different times in the ETL cycle. In both cases, you can assign session variables to dynamically select the appropriate table.
Default Initialization String / Use database specific SQL	For non-multidimensional data source tables (not alias tables), this option appears if you choose a Table Type of **Stored Proc** or **Select**. For multidimensional data source tables, this appears if you choose a Table Type of **Select**. When you select this option, you can specify the data source and type the SQL statements. See Table 8–1 for more information.
Cacheable	Select this option to include the table in the Oracle BI Server query cache. Typically, select this option for tables that do not need to be accessed in real time. When you select this option, the Cache persistence time settings become active. Note that there are additional configuration settings that affect the behavior of the query cache. See "Configuring Query Caching" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for full information.
Cache never expires	When you select this option, cache entries do not automatically expire. This could be useful when a table is important to a large number of queries that users might run. For example, if most queries have a reference to an account object, keeping it cached indefinitely could actually improve performance rather than compromise it. Note that selecting this option does not mean that an entry always remains in the cache. Other invalidation techniques, such as manual purging, LRU (Least Recently Used) replacement, metadata changes, or use of the cache polling table can result in entries being removed from the cache.

Table 8–2 (Cont.) General Properties for Physical Tables

Property	Description
Cache persistence time	How long table entries should persist in the query cache, or in other words, the cache expiration time. Setting a cache persistence time is useful for OLTP data sources and other data sources that are updated frequently. For example, you could set this option to refresh the underlying physical tables daily for a particular dashboard. If a query references multiple physical tables with different persistence times, the cache entry for the query exists for the shortest persistence time set for any of the tables referenced in the query. This makes sure that no subsequent query gets a cache hit from an expired cache entry. For more information, see "Troubleshooting Problems with Event Polling Tables" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.
External name	Applies to physical cube tables from multidimensional data sources. The external name is the physical name that is used when referencing the cube table in physical SQL queries. This value must reflect the external name defined in the data source.
Display Column	For Essbase data sources only. See "Working with Essbase Data Sources" for more information.
Hint	Available only for some data sources. See "Using Hints in SQL Statements" for more information.

Creating Alias Tables

To create an alias table, right-click an existing physical table and select **New Object**, then select **Alias**. You can also create aliases on opaque views and stored procedures.

Table 8–3 describes properties that are specific to alias tables. Refer to Table 8–2 for information about other table properties that are shared between physical tables and alias tables.

Table 8–3 Properties Specific to Physical Alias Tables

Property	Description
Source Table	Applies to alias tables. Click **Select** to choose the original physical table from which to create an alias table.
Override Source Table Caching Properties	Applies to alias tables. Click this field to enable the cacheable properties. You can select or clear the appropriate cacheable options.

Viewing Physical Table Properties

The Properties tab for physical tables displays name-value pairs that are used for some data sources as a generic mechanism for extending the Physical layer metadata. Typically, the values are passed up from the data source, but you can edit the values if needed.

See also "Viewing Physical Column Properties" for related information.

Setting Physical Table Properties for XML Data Sources

Use the XML tab to set or edit properties for an XML data source. The XML tab of the Physical Table dialog provides the same functionality as the XML tab of the

Connection Pool dialog. However, setting properties in the Physical Table dialog overrides the corresponding settings in the Connection Pool dialog. See "Setting Connection Pool Properties in the XML Tab" for more information.

Creating and Managing Columns and Keys for Relational and Cube Tables

Each physical table and physical cube table in the Physical layer of the Administration Tool has one or more physical columns. You can use the Columns, Keys, and Foreign Keys tabs in the Physical Table dialog to view, create new, and edit existing columns, keys, and foreign keys that are associated with the table.

The following list describes the buttons that appear in the tabs:

- **New.** Lets you create a new object by opening the dialog that corresponds to the tab.
- **Edit.** When you select an object and click **Edit**, the dialog that corresponds to the tab appears. You can then edit the properties of the object.
- **Delete.** Deletes the selected object.

This section contains the following topics:

Creating and Editing a Column in a Physical Table

An imported column's properties are set automatically. After import, you can modify the column's property, including its type and whether null values are allowed for the column.

The following list contains information about nullable and data type values for columns imported into the Physical layer.

- **Nullable.** Indicates whether null values are allowed for the column. If null values can exist in the underlying table, you need to select this option. This allows null values to be returned to the user, which is expected with certain functions and with outer joins. It is generally safe to change a non-nullable value to a nullable value in a physical column.
- **Type.** Indicates the data type of the column. Use caution when changing the data type. Setting the values to data types that are incorrect in the underlying data source might cause unexpected results. If there are any data type mismatches, correct them in the repository or reimport the columns that have mismatched data types.

  If you reimport columns, you also need to remap any logical column sources that reference the remapped columns. The data type of a logical column in the business model must match the data type of its physical column source. The Oracle BI Server passes these logical column data types to client applications.

  Longvarchar and longvarbinary data types are supported for writing complete Logical SQL statements into usage tracking tables for debugging purposes. They are not supported for general-purpose queries, and cannot be displayed in Presentation Services. Use direct SQL utilities to access columns with these data types.

Except when stated otherwise, the characteristics and behavior of a physical cube column are the same as for other physical columns.

> **Note:** Creating, modifying, or deleting a column in an original physical table also creates, modifies, or deletes the same column on all its alias tables.

To create or edit a physical column:

1. In the Physical layer of the Administration Tool, perform one of the following steps:
   - To create a physical column, right-click a physical table and select **New Object**, then select **Physical Column**.
   - To create a physical cube column for a multidimensional data source, right-click a physical cube table and select **New Object**, then select **Physical Cube Column**.
   - To edit an existing physical column, double-click the physical column object in the Physical layer.
2. In the Physical Column dialog, type a name for the physical column.

   For XML data sources, this field stores and displays the unqualified name of a column (attribute) in an XML document.
3. In the **Type** field, select a data type for the physical column.
4. If applicable, specify the length of the data type.

   For multidimensional data sources, if you select `VARCHAR`, you need to type a value in the **Length** field.
5. Select the **Nullable** option if the column is allowed to have null values.
6. In the **External Name** field, type an external name.
   - Required if the same name (such as `STATE`) is used in multiple hierarchies.
   - Optional for XML documents. The **External Name** field stores and displays the fully qualified name of a column (attribute).
7. (Multidimensional data sources) When the physical cube column is a measure, in the **Aggregation role** list, select the appropriate value.

   A new physical cube column is created as a measure by default. See "Working with Multidimensional Sources in the Physical Layer" for information about changing this behavior.
8. Click **OK**.

Specifying a Primary Key for a Physical Table

Use the Physical Key dialog to specify the column or columns that define the primary key of the physical table.

To specify a primary key for a physical table:

1. In the Physical layer of the Administration Tool, right-click a physical table and select **Properties**.
2. In the Physical Table dialog, click the Keys tab.

3. In the Keys tab, click **New**.
4. In the Physical Key dialog, type a name for the key.
5. Select the column that defines the primary key of the physical table.
6. (Optional) Type a description for the key.
7. Click **OK**.

Deleting Physical Columns for All Data Sources

When you delete a physical column, the following occurs:

- **Multidimensional data sources.** If you delete property or key columns from a level, the association is deleted and the column changes to a measure under the parent cube table.
- **Alias tables.** Deleting a column in an original physical table deletes the same column on all its alias tables.

Viewing Physical Column Properties

The Properties tab for physical columns displays name-value pairs that are used for some data sources as a generic mechanism for extending the Physical layer metadata. Typically, the values are passed up from the data source, but you can edit the values if needed.

Viewing Data in Physical Tables or Columns

You can view the data in a physical table or an individual physical column by right-clicking the object and choosing **View Data**. In online editing mode, you must check in changes before you can use this option.

View Data is not available for physical cube tables or columns. See "Viewing Members in Physical Cube Tables" for more information.

Because the View Data feature issues a row count, it is not available for data sources that do not support row counts. See "Displaying and Updating Row Counts for Physical Tables and Columns" for more information.

Note: View Data does not work in online mode if you set the user name and password for connection pools to `:USER` and `:PASSWORD`. In offline mode, the Set values for variables dialog appears so that you can populate `:USER` and `:PASSWORD` as part of the viewing process.

Working with Multidimensional Sources in the Physical Layer

This section provides information about physical cube tables, dimensions, and hierarchies from multidimensional data sources.

This section contains the following topics:

About Physical Cube Tables

Each cube from a multidimensional data source is structured as a physical cube table, a type of physical table. It has all the capabilities of a table, such as physical cube columns and keys (optional) and foreign keys (optional). It also has cube-specific metadata such as hierarchies and levels.

When you import the physical schema, the Oracle BI Server imports the metadata for the cube, including its metrics, hierarchies, and levels. Expanding the hierarchy object in the Physical layer reveals the levels in the hierarchy. In the Physical Cube Table dialog, the Hierarchies tab lists the dimensional hierarchies in the cube.

Each multidimensional catalog in the data source can contain multiple physical cubes. You can import the metadata for one or more of these cubes into the Oracle BI repository. Although it is possible to create a cube table manually, it is recommended that you import metadata for cube tables and their components.

If you do create cubes manually, you must build each cube one hierarchy at a time and test each one before building another. For example, create the time hierarchy and a measure and test it. When it is correct, create the geography hierarchy and test it. This helps ensure that you have set up each cube correctly, and makes it easier to identify any setup errors.

About Measures in Multidimensional Data Sources

You need to select the aggregation rule for a physical cube column carefully to make sure the measures are correct. Setting it correctly might improve performance.

Always verify aggregation rules after importing cube metadata. Typically, aggregation rules are assigned correctly when you import cube metadata. However, if a measure is a calculated measure, the aggregation rule is reported as None. Therefore, you must examine the aggregation rule for all measures after importing a cube to verify that the aggregation rule has been assigned correctly.

For all measures assigned an aggregation rule value of None, contact the multidimensional data source administrator to verify that the value of the aggregation rule is accurate. If you need to change the aggregation rule, you can change it in the Physical Cube Column dialog.

Use the following guidelines to assign the correct aggregation rule:

- If the generated physical queries to the database should send an aggregation function, such as `SUM(revenue)`, then set that function as the aggregation rule. With this setting, the Oracle BI Server typically sends the aggregation to the database in the query, but might also perform aggregations itself in certain situations.
- If the data for this measure should not be aggregated in the query or by the Oracle BI Server, use the External Aggregation rule. It is important to choose this setting when the measure uses a more complex calculation inside the data source than the Oracle BI Server can replicate with a simple aggregation rule (such as calculations for ratios, consolidations and allocations). This option is also useful when the cube persists a full set of pre-aggregated results.

About Externally Aggregated Measures

In a multidimensional data source, some cubes contain very complex, multi-level based measures. If you assign an aggregation rule of External Aggregation, the Oracle BI Server bypasses its internal aggregation mechanisms and uses the pre-aggregated measures. When imported, these measures are assigned an aggregate value of None.

The following are some guidelines for working with pre-aggregated measures:

- External aggregation only applies to multidimensional data sources (such as Essbase, Hyperion Financial Management, Microsoft Analysis Services, and SAP/BW) that support these complex calculations.
- You cannot assign external aggregation to measures from non-multidimensional data sources. If the required aggregation rule is supported by the Oracle BI Server and can be mapped to a relational data source, then it is not complex and does not require external aggregation.
- There is only one aggregation rule for a logical measure. Therefore, a single logical column cannot federate a noncomplex aggregation rule for a mapping to a non-multidimensional source, with a complex aggregation rule for a mapping to a multidimensional source. Instead, you need to create one logical measure for each source, and create a third logical measure that derives from the first two.
- You can mix noncomplex measures from non-multidimensional data sources with noncomplex measures from multidimensional data sources if they are aggregated through the Oracle BI Server.

About Working with Physical Dimensions and Physical Hierarchies

Most dimensions and hierarchies are imported into the Physical layer from multidimensional data sources, rather than created manually. If a particular hierarchy is not imported, any columns associated with that hierarchy are also not imported. If users need access to columns that are not imported, first add these columns to the Physical layer by manually creating them and associate them with a level in a hierarchy.

Each level in a hierarchy has a level key. The first cube column associated with (added to) the level of a hierarchy is the level key. This must match with the data source definition of the cube. The icon for the column that you select first changes to the key icon after it is associated with the level of a hierarchy.

Oracle Business Intelligence supports unbalanced hierarchies for all multidimensional data sources. In general, you can configure unbalanced hierarchies in the Physical layer by changing the hierarchy type.

You can view and edit properties for physical dimensions and hierarchies by double-clicking physical dimension and physical hierarchy objects in the Physical layer of the Administration Tool. You can also view and edit these objects from the Dimensions and Hierarchies tabs of the Cube Table dialog.

This section contains the following topics:

- Working with Physical Dimension Objects
- Working with Physical Hierarchy Objects

Working with Physical Dimension Objects

In the Physical Dimension dialog, you can view and edit the name and description of the dimension. You can also add, remove, or edit hierarchies for that dimension, and add, remove, or edit columns that represent dimension properties.

Working with Physical Hierarchy Objects

When you select columns to add to a hierarchy, it is recommended that you select them in hierarchical order, starting with the highest level. If you select multiple columns and bring them into the hierarchy at the same time, the order of the selected

group of columns remains the same. After adding columns to the hierarchy, you can change the order of the columns in the Browse dialog.

In the Physical Hierarchy dialog, you can view and edit the name and description of the hierarchy, along with the properties described in Table 8–4. For level-based hierarchies, you can add, remove, edit, or reorder levels. For value-based hierarchies, click the Column tab to add, remove, or edit columns. To specify a key column, double-click a column name.

In the Physical Level dialog, you can view and edit the name, external name, and description of the level. You can also add, remove, or edit columns for that level. To designate a column as a level key, double-click a column name.

Always review the hierarchy type after import to ensure that it is set appropriately. The way this parameter is set upon import depends on the data source. For example, all Essbase hierarchies are initially set to Unbalanced. Review the hierarchy type for each hierarchy and change it as appropriate.

Typically, you always need to manually set the hierarchy type for parent-child (value) hierarchies, except for Hyperion Financial Management hierarchies, which are always set to Value by default upon import. Review the hierarchy type and change the type to Value as appropriate. Parent-child (value) hierarchies are those in which a business transaction, or a cube refresh, can change the number of levels.

For parent-child hierarchies, you must manually set the physical hierarchy type to Value *before* dragging metadata to the Business Model and Mapping layer. The hierarchy type in the Business Model and Mapping layer is set automatically based on the physical hierarchy setting. For all other types, you can determine the hierarchy type later, without needing to rebuild the logical model.

You must also ensure that the corresponding logical dimension properties are correct for queries to work. See Chapter 10, "Working with Logical Dimensions" for more information.

For SAP/BW data sources, all hierarchies default to fully balanced hierarchies on import. The hierarchy type for two-level hierarchies (which typically correspond to characteristic primary hierarchies) should not be changed. Review all SAP/BW multi-level (external) hierarchies to determine whether any are parent-child hierarchies, and set them to Value as needed.

Table 8–4 Options in the Physical Hierarchy Dialog

Property	Description
External Name	The physical name that is used when referencing the hierarchy in physical MDX queries. This value must reflect the external name defined in the data source.
Dimension Name	(Dimension Unique Name) Dimension to which the hierarchy belongs.
Dimension Type	Identifies whether this hierarchy belongs to a time dimension, measure dimension, or other type of dimension.

Table 8–4 (Cont.) Options in the Physical Hierarchy Dialog

Property	Description
Hierarchy Type	Identifies the type of hierarchy, as follows: ■ **Fully balanced:** A level-based hierarchy with no unbalanced or skip characteristics. Corresponds to a level-based hierarchy in the Business Model and Mapping layer. ■ **Unbalanced:** Also called ragged. A hierarchy where the leaves (members with no children) do not necessarily have the same depth. Corresponds to a level-based hierarchy with the **Ragged** option selected in the Business Model and Mapping layer. ■ **Ragged balanced:** Also called skip. A hierarchy where there are members that do not have a value for a particular ancestor level. Corresponds to a level-based hierarchy with the **Skipped Levels** option selected in the Business Model and Mapping layer. ■ **Network:** This hierarchy type is not used. ■ **Value:** Also called parent-child. A hierarchy of members that all have the same type. This contrasts with level-based hierarchies, where members of the same type occur only at a single level of the hierarchy. Corresponds to a parent-child hierarchy in the Business Model and Mapping layer. **Note:** For level-based hierarchies with both unbalanced and skip-level characteristics, choose either **Unbalanced** or **Ragged balanced** as the physical hierarchy type. Then, ensure that both **Ragged** and **Skipped Levels** are selected for the corresponding logical dimension in the Business Model and Mapping layer.
Default member type ALL	This option is not used.
Use unqualified member name for better performance	Select this option when member names (including aliases) are unique in a given hierarchy so that the Oracle BI Server can take advantage of specific MDX syntax to optimize performance.

Adding or Removing Cube Columns in a Hierarchy After importing a hierarchy, you may need to add or remove a column. If you remove a cube column from a hierarchy, it is deleted from the hierarchy but remains in the cube table and is available for selection to add to other levels.

To add or remove a cube column in an existing hierarchy:

1. In the Physical layer of the Administration Tool, double-click the physical hierarchy that you want to change. The Physical Hierarchy dialog appears.
2. For level-based hierarchies, double-click the level for which you want to add or remove columns. Then, in the Physical Level dialog, you can add, remove, or edit columns. When you are finished, click **OK** in the Physical Level dialog.
3. For value-based hierarchies, click the Columns tab. You can add, remove, or edit columns, and designate member key and parent key columns.
4. Click **OK** in the Hierarchy dialog.

Working with Cube Variables for SAP/BW Data Sources

In SAP/BW data sources, cube variables are used as a means of parameterizing queries. Cube variable objects are imported into the Physical layer when metadata is imported from Querycubes/Bex Queries in SAP/BW data sources.

Typically, you do not edit these objects directly except to keep them synchronized with the Bex queries in the data source, and except to specify overrides for key characteristics values.

The Cube Variables tab of the Cube Table dialog lists the cube variables for the given cube table, along with the cube variable caption. Double-click a cube variable for more detailed information, or click the **Add** button to define a new cube variable.

Table 8–5 describes the properties of cube variables for SAP/BW data sources. See the SAP/BW documentation for additional information.

Table 8–5 Cube Variable Properties

Property	Description
Name	Name of the cube variable.
Caption	A description (label or caption) associated with the cube variable, mainly used for display purposes.
Variable Type	The type of cube variable. Variable types include: ■ SAP_VAR_TYPE_MEMBER: A placeholder for a selection for MEMBER_UNIQUE_NAMES. ■ SAP_VAR_TYPE_HIERARCHY: A placeholder for a HIERARCHY_UNIQUE_NAME. ■ SAP_VAR_TYPE_NUMERIC: A placeholder for a numeric value in formulas.
Selection Type	The selection type of the cube variable, for cube variables of type SAP_VAR_TYPE_MEMBER. Selection types include: ■ SAP_VAR_SEL_TYPE_VALUE: The variable is replaced by a single value. Cube variables of type NUMERIC must have this selection type. ■ SAP_VAR_SEL_TYPE_INTERVAL: A placeholder for an interval. ■ SAP_VAR_SEL_TYPE_COMPLEX: A placeholder for a complex selection.
Entry Type	Indicates whether replacing variables is optional or mandatory. Entry types include: ■ SAP_VAR_INPUT_TYPE_OPTIONAL: Specifying a value is optional for this variable. ■ SAP_VAR_INPUT_TYPE_MANDATORY: You must specify a value for this variable. ■ SAP_VAR_INPUT_TYPE_MANDATORY_NOT_INITIAL: You must specify a value for this variable. An initial field is not a valid entry.
Reference Dimension	This column contains a DIMENSION_UNIQUE_NAME for the parameter type SAP_VAR_TYPE_HIERARCHY.
Reference Hierarchy	This column contains a HIERARCHY_UNIQUE_NAME for the variable type SAP_VAR_TYPE_MEMBER.
Default Low	This property contains a default value for the variable or is zero.
Default High	This property contains a default value for the variable or is zero. This property is only important for variables with the selection type SAP_VAR_SEL_TYPE_INTERVAL and SAP_VAR_SEL_TYPE_SELECTION.

Table 8–5 (Cont.) Cube Variable Properties

Property	Description
Override Default Low	Provide a default value for the cube variable in this field if the Default Low is zero. You must specify a value for this property for mandatory variables that do not specify a default value.
Override Default High	Provide a default value for the cube variable in this field if the Default High is zero. You must specify a value for this property for mandatory variables that do not specify a default value.

Viewing Members in Physical Cube Tables

You can view members of hierarchies or levels in the Physical layer of repositories. To view members, the repository must be open in online mode. The list of members by level in the hierarchy can help you determine if the connection pool is set up properly. You might want to reduce the time it takes to return data or the size of the returned data by specifying a starting point (Starting from option) and the number of rows you want returned (Show option).

To view members:

1. Open the Administration Tool in online mode.
2. In the Physical layer, right-click a hierarchy or level.
3. Select **View Members**.

   A window opens showing the number of members in the hierarchy and a list of the levels. You might need to enlarge the window and the columns to view all the returned data.
4. Click **Query** to display results.
5. When finished, click **Close**.

Working with Essbase Data Sources

This section describes how Essbase data is modeled by default in the Physical layer of the Oracle BI repository, and describes the tasks you can perform to model the data in different ways.

This section contains the following topics:

About Using Essbase Data Sources with Oracle Business Intelligence

When you import metadata from Essbase data sources, the cube metadata is mapped to the Physical layer in a way that supports the Oracle Business Intelligence logical model.

Metadata that applies to all members of the dimension, such as aliases, are modeled as dimension properties by default. Level-based properties, such as outline sort/memnor information, are mapped as separate physical cube columns in the dimension.

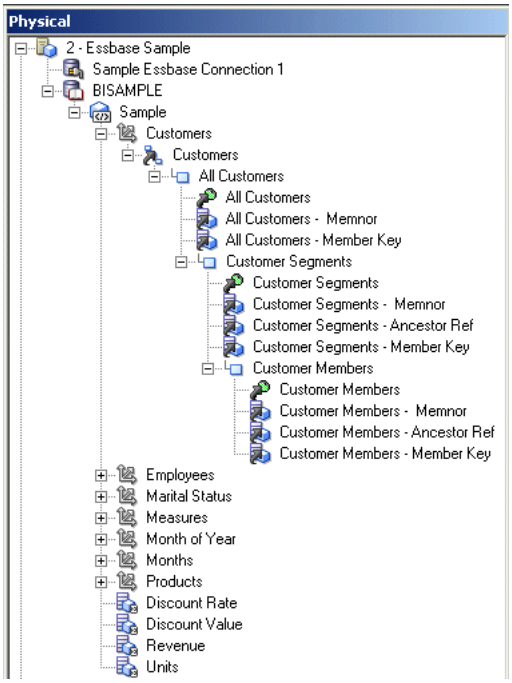
The following physical column types are used for Essbase metadata:

- **Member Alias:** Indicates an Alias column.
- **UDA:** Indicates the column is a User Defined Attribute (UDA).
- **Outline Sort:** Indicates the column is of memnor type, used for outline sorts in the logical layer. Imported at the lowest level of each dimension.
- **Attribute:** Indicates the column is of attribute type, for attribute dimensions.
- **Other:** The type is different than those listed, or unknown.
- **Ancestor Reference:** References the ancestor of a dimension.
- **Member Key:** Indicates the column is a member key.
- **Leaf:** Indicates that the column is the lowest member of the hierarchy.
- **Root:** Indicates that the column is the root member of the hierarchy.
- **Parent Reference:** References the parent of a dimension.

The column types Outline Sort, Ancestor Reference, Member Key, Leaf, Root, and Parent Reference are used internally by the system and should not be changed.

Figure 8–2 shows Essbase data that has been imported into the Physical layer.

Figure 8–2 Essbase Data Modeled in the Physical Layer



There are different options in the Physical layer that let you control how you want to model certain types of metadata. Choose the option that best meets the needs of the user base. For example, many types of Essbase metadata are modeled as dimension properties by default in the Physical layer.

Alternatively, you can choose to flatten the Essbase metadata in the Physical layer for ease of use with the attribute-style reporting supported in previous releases of Oracle Business Intelligence.

The following list summarizes some of these modeling options:

- **Aliases.** Aliases are modeled as dimension properties by default, but you can also choose to flatten them using the **Create Columns for Alias Table** feature. See "Working with Essbase Alias Tables" for more information.
- **UDAs.** UDAs are modeled as dimension properties by default, but you can also choose to flatten them using the **Create Columns for UDA** feature. See "Modeling User-Defined Attributes" for more information.
- **Alternate Hierarchies.** Alternate hierarchies are modeled as separate hierarchies by default, but you can choose to view them in as a single hierarchy using the **Convert to single hierarchy view** feature. See "Modeling Alternate Hierarchies" for more information.
- **Measure Hierarchies.** By default, measures are imported as a single measure column that represents all the measures, but you can also choose to view each measure as an individual column using the **Convert measure dimension to flat measures** feature. See "Modeling Measure Hierarchies" for more information.

Note the following additional information about using Essbase data sources with Oracle Business Intelligence:

- **Substitution variables.** Essbase substitution variables are automatically retrieved and populated into corresponding Oracle BI Server repository variables. Depending on the scope of the Essbase variable, the naming convention for the Oracle BI Server variable is as follows:

  Server instance scope: *server_name*:*var_name*

  Application scope: *server_name*:*app_name*:*var_name*

  Cube scope: *server_name*:*app_name*:*cube_name*:*var_name*

  A single initialization block is also created in the repository for the Essbase variables. Set the appropriate refresh interval in the initialization block to reflect anticipated update cycles for Essbase variables.
- **Essbase Generations.** Essbase Generations are mapped to physical level objects.
- **Time series functions.** The Oracle BI Server time series functions `AGO`, `TODATE`, and `PERIODROLLING` are sent to Essbase to take advantage of the native capabilities of the Essbase server.
- **Database functions.** You can use the database SQL functions `EVALUATE` and `EVALUATE_AGGREGATE` to leverage functions specific to Essbase data sources. See "Examples Using EVALUATE_AGGREGATE and EVALUATE to Leverage Unique Essbase Functions" for more information.

  Note that `EVALUATE_PREDICATE` is not supported for use with Essbase data sources.
- **Gen 1 levels.** By default, Gen 1 levels are included when you drag and drop an Essbase cube or dimension from the Physical layer to the Business Model and Mapping layer. However, because Gen 1 levels are not usually needed for analysis, you can choose to exclude Gen 1 levels when you drag and drop Essbase objects to

the business model. To do this, select **Skip Gen 1 levels in Essbase drag and drop actions** in the General tab of the Options dialog. See "Setting Administration Tool Options" for more information.

- **Hierarchy types.** For Essbase data sources, all hierarchies are imported as Unbalanced by default. Review the **Hierarchy Type** property for each physical hierarchy and change the value if necessary. Supported hierarchy types for Essbase are **Unbalanced**, **Fully balanced**, and **Value**.

About Incremental Import

You can incrementally import Essbase metadata, meaning you can perform an initial import and then import again. You might want to import incrementally when information in the data source has changed, or when the first import only included a subset of the metadata.

Note the following about incremental import:

- When you re-import metadata that already exists in the Physical layer, a message appears, warning you that the Physical objects will be overwritten.
- If you delete data in the source, re-importing the metadata does not automatically perform the deletion in the Physical layer. Instead, you must manually delete the corresponding Physical objects.
- If you rename an object in the source, the renamed object is imported as a new object. In this case, both the old object and the new (renamed) object are displayed in the Physical layer.
- In general, customizations that you have performed on the Physical layer data, such as determining the alias column to use for display, are retained after an incremental import. If you want to revert to the default imported view, you must delete the existing Physical layer objects and re-import the metadata.

Working with Essbase Alias Tables

Essbase cubes support the concept of aliases, which are alternate names for members or shared members. Often, members have separate aliases for each user language to enable users to view member names in their own language.

For example, the member name might be a product code (100), with a default alias for the product name (Cola) and an additional alias for the long name (Cherry Cola).

In the Essbase cube, aliases are stored in alias tables that map a specific set of alias names to member names. Typically, a Default alias table exists for each cube.

This section contains the following topics:

Determining the Value to Use for Display

When you import metadata from Essbase into the Oracle BI repository, the Essbase cube table object in the Physical layer has a property that determines which value to display for members: the member name, the default alias name, or some other alias name. By default, the columns display the default alias name.

To change the value to display for members:

1. In the Physical layer of the Administration Tool, double-click an Essbase cube table.

2. In the General tab of the Cube Table dialog, choose the appropriate value for **Display Column**. You can select **Member Names**, or you can select **Alias** and choose an alias table name from the list, or you can select **Variable** and choose a variable that contains a valid display column name.
3. Click **OK**.

Explicitly Defining Columns for Each Alias

Aliases are modeled as dimension properties in the Physical layer after import. If you want to work with more than one alias, such as when you want to flatten attributes for reporting purposes or externalize strings for translation, you can explicitly define columns for each alias. You can define alias columns at the cube, dimension, or hierarchy level.

To explicitly define columns for each alias:

1. In the Administration Tool, in the Physical layer, right-click the cube table, physical dimension, or physical hierarchy for which you want to define alias columns.
2. Select **Create Columns for Alias Table**. Then, from the sub-list, select the alias table for which you want to create columns.

   Note that the **Fetch** button is not used.
3. Click **Create**.
4. Drag the new alias columns to the appropriate location in the Business Model and Mapping layer.

If you want to externalize strings for translation based on the alias columns, see "Localizing Oracle Business Intelligence" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

Modeling User-Defined Attributes

Essbase supports the concept of user-defined attributes (UDAs). A UDA is essentially any arbitrary textual string that can be associated with any member from a dimension. A member can have multiple strings associated to it.

You can choose whether to import UDAs in the Import Metadata Wizard. If you choose to import UDAs, then by default, each UDA is modeled as a dimension property in the Physical layer of the repository.

You can also choose to model each UDA as a separate physical column. To do this, perform one of the following tasks:

- To model all UDAs in a cube as separate physical columns, right-click the cube table and select **Create columns for UDA**. All UDAs in the cube are modeled as separate physical columns.
- To model all UDAs in a dimension as separate physical columns, right-click the dimension object and select **Create columns for UDA**, then select **All UDAs**. All UDAs in the dimension are modeled as separate physical columns.
- To model a particular UDA in a dimension as a separate physical column in each level, right-click the dimension object and select **Create columns for UDA**, then select the specific UDA you want to model. The selected UDA is modeled as a separate physical column for each level.

Associating Member Attributes to Dimensions and Levels

Member attributes are not automatically associated to corresponding dimensions and levels during the import process. To manually create the association, map the member attribute to the appropriate logical table.

In other words, drag and drop the columns from the attribute dimension in the Physical layer to the appropriate logical tables in the Business Model and Mapping layer.

Modeling Alternate Hierarchies

By default, alternate hierarchies are modeled as separate hierarchies in the Physical layer. You can choose to view them as separate hierarchies (called the multi-hierarchy view), or as a single hierarchy.

To view alternate hierarchies as a single hierarchy, right-click the dimension object containing the alternate hierarchies and select **Convert to single hierarchy view**. To return to the multi-hierarchy view, right-click the dimension object again and select **Convert to multi-hierarchy view**.

For example, Figure 8–3 shows the multi-hierarchy view for an alternate hierarchy.

Figure 8–3 Essbase Alternate Hierarchy Displayed in Multi-Hierarchy View

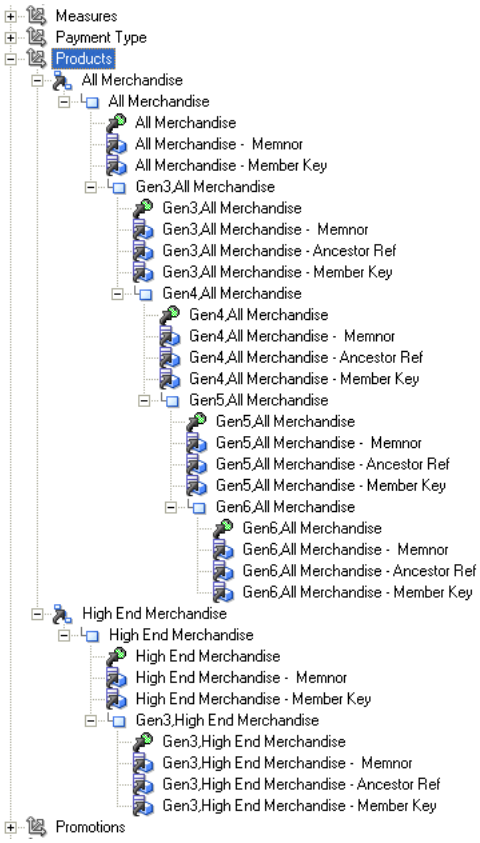
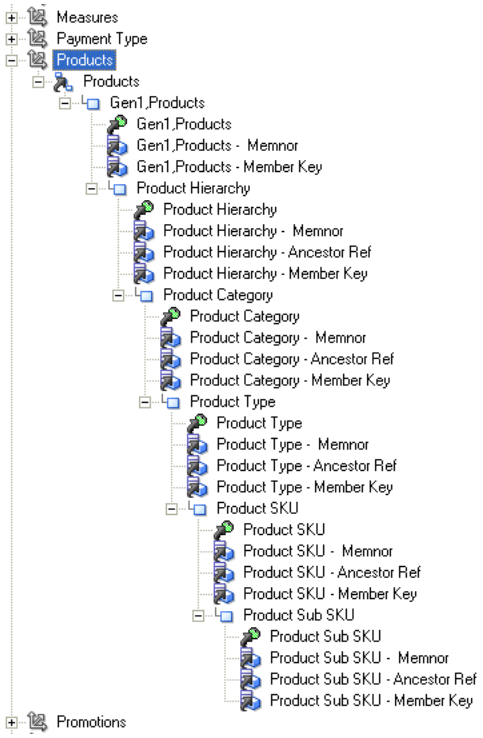


Figure 8–4 shows the single-hierarchy view for the same alternate hierarchy.

Figure 8–4 Essbase Alternate Hierarchy Displayed in Single-Hierarchy View



Modeling Measure Hierarchies

By default, measures are imported as measure hierarchies. In other words, the cube contains a single measure column that represents all the measures.

Alternatively, you can choose to flatten the measure hierarchy to view each measure as an individual column. To do this, right-click the cube object and select **Convert measure dimension to flat measures**.

Improving Performance by Using Unqualified Member Names

When member names (including aliases) are unique in a given hierarchy, the Oracle BI Server can take advantage of specific MDX syntax to optimize performance.

To enable this capability, select **Use unqualified member name for better performance** in the Hierarchy dialog.

The import process cannot identify that member names are unique for a given hierarchy, so it is the responsibility of the administrator to confirm uniqueness. Note that query errors result when a hierarchy is specified as having unique members when it does not.

Note: If you find that the Oracle BI Server is generating incorrect queries for Essbase, check whether there are duplicate member names in a given hierarchy. If there are, ensure that the option **Use unqualified member name for better performance** is not selected for that hierarchy, or perform the following steps:

1. From the Essbase outline, update each offending member variable by adding a prefix or suffix to make the member name unique.
2. Update SQL queries as necessary, if references are made to data within SQL.
3. Reload the data and members in the Essbase outline.

Working with Hyperion Financial Management and Hyperion Planning Data Sources

This topic contains the following sections:

Importing Metadata From Hyperion Financial Management Data Sources

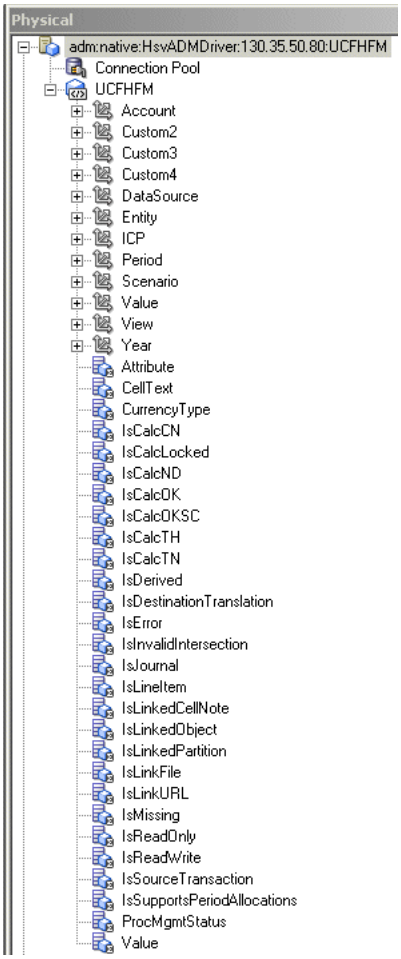
When you import metadata from Hyperion Financial Management data sources, both measures and dimensions are imported into the Physical layer.

The Hyperion Financial Management hypercube model is exposed in the Physical layer in the following ways:

- There is only one measure in Hyperion Financial Management, called Value. This measure is modeled as a single fact column in the Physical layer, also called Value.
- The measure column Value - data type DOUBLE is imported for Hyperion Financial Management.
- The Value measure has three base properties: CellText, CurrencyType, and Attribute. These properties are all represented as additional fact columns.
- The Attribute property has additional sub-properties, such as IsReadOnly. These properties are also exposed as additional columns.

Figure 8–5 shows how Hyperion Financial Management data is modeled in the Physical layer.

Figure 8–5 Hyperion Financial Management Metadata in the Physical Layer



All Hyperion Financial Management dimensions are modeled as parent-child hierarchies in the Physical layer. Shared members, alternate hierarchies, and unbalanced hierarchies are supported.

Dimension member properties are exposed as columns (such as Name, Description, ShortName, and so on). An additional column called Sort Order is also displayed for each dimension. This column contains custom sort information retrieved from the Hyperion Financial Management data source.

Each Hyperion Financial Management dimension has a corresponding Point of View (POV) value that provides customized information for different users. This POV value is exposed as the **Default Member** in the Hierarchies tab of the Dimension dialog. Although the Default Member field is populated upon import, note that you may need to update the default values according to the needs of the user base.

Note: Do not select the **Default member type ALL** option for Hyperion Financial Management hierarchies.

Importing Metadata From Hyperion Planning Data Sources

Hyperion Planning Server version 11.1.2.4 or above is required for importing and querying metadata from Hyperion Planning Data Sources. When you import data

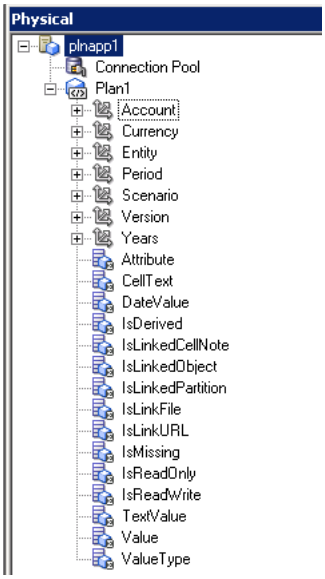
from Hyperion Planning data sources, both measures and dimensions are imported into the Physical layer.

The Hyperion Planning model is exposed in the Physical layer in the following ways:

- The imported Hyperion Planning data source contains multiple Value columns to support measures of different data types.
- Some measures specific to Hyperion Financial Management are not imported for Hyperion Planning data sources.
- The Attribute property has additional sub-properties, such as IsReadOnly. These properties are also exposed as additional columns.

Figure 8–6 shows how Hyperion Planning data is modeled in the Physical layer.

Figure 8–6 Hyperion Planning Metadata in the Physical Layer



All Hyperion Planning dimensions are modeled as parent-child hierarchies in the Physical layer. Shared members, alternate hierarchies, and unbalanced hierarchies are supported.

Dimension member properties are exposed as columns (such as Name, Description, ShortName, and so on). An additional column called Sort Order is also displayed for each dimension. This column contains custom sort information retrieved from the Hyperion Planning data source.

Each Hyperion Planning dimension has a corresponding Point of View (POV) value that provides customized information for different users. This POV value is exposed as the **Default Member** in the Hierarchies tab of the Dimension dialog. Although the Default Member field is populated upon import, note that you may need to update the default values according to the needs of the user base.

Note: Do not select the **Default member type ALL** option for Hyperion Planning hierarchies.

About Query Support for Hyperion Financial Management and Hyperion Planning Data Sources

Both member queries (dimensional browsing) and data queries (measure analysis) are supported for Hyperion Financial Management and Hyperion Planning data sources.

Most Logical SQL functions are performed in the Oracle BI Server. However, use `EVALUATE_PREDICATE` to access the following functions specific to Hyperion Financial Management and Hyperion Planning:

- `PeriodOffset` (used to access prior or future periods through an offset)
- NA Suppression functions specific to Hyperion Financial Management (`SuppressDerived`, `SuppressInvalidIntersection`, `SuppressNoAccess`, `SuppressZero`, `SuppressError`)
- `Base` function (returns the leaf members below a given ancestor member)
- `CommonChildren`
- User-defined functions

See "EVALUATE_PREDICATE" for detailed information about syntax and usage.

Note that in previous releases, you manually added the `SuppressMissing` function to `EVALUATE_PREDICATE` to suppress missing cells. The current release of Oracle BI EE supports the PERF_PREFER_SUPPRESS_EMPTY_TUPLES database feature. This database feature causes Oracle BI EE to insert the `SuppressMissing` function into the Hyperion Financial Management or Hyperion Planning data query to suppress missing cells. PERF_PREFER_SUPPRESS_EMPTY_TUPLES controls whether empty tuples with empty cell values are eliminated. Note that this database feature does not change the null suppression behavior on the final result set.

Note that there is no native support for time series functions. Time series functions are only supported through data modeling.

Working with Oracle OLAP Data Sources

Oracle Database has an OLAP Option that provides an embedded, full-featured online analytical processing server.

The OLAP Option is used in the following roles:

- A summary management solution to SQL-based business intelligence tools and applications.
- A calculation engine that provides SQL-based business intelligence tools with rich analytic content.
- A full-featured multidimensional server, servicing dimensionally oriented business intelligence tools and applications.

Oracle Business Intelligence supports Oracle OLAP as a data source. When you import metadata from an Oracle OLAP source, the Oracle OLAP objects appear in the Physical layer of the Administration Tool. This section provides information about the Oracle OLAP objects in the Physical layer.

This section contains the following topics:

- Working with Oracle OLAP Cubes and Columns

About Importing Metadata from Oracle OLAP Data Sources

This section provides information about using the Administration Tool to import metadata from Oracle OLAP, as follows:

- For Oracle OLAP cubes with multi-language metadata, only the default language is imported.
- Only dimensions that contain at least one hierarchy are imported.
- Multiple hierarchies in a single query are not supported. If a query includes columns from multiple hierarchies in a given dimension, the Oracle BI Server returns an error.
- The default aggregation rule in the Business Model and Mapping layer for Oracle OLAP measures is External Aggregation. The External Aggregation rule means that the Oracle BI Server is not aware of the underlying aggregation rule for the specific measure and will not compute it internally. Instead, the Oracle BI Server will always send the query to the underlying multidimensional data source for aggregation.

  In some cases, you may want to set the aggregation rule for a measure to something other than External Aggregation. For example, you may have federated multiple data sources, or you may want to perform higher-level aggregation along dimension attributes that are not represented by a level in Oracle OLAP. In both of these cases, you can change the default aggregation rule to match the rule in the underlying data source or sources. Note that the aggregation is still performed in the Oracle OLAP data source where possible.

See "System Requirements and Certification" for the latest information about the versions of Oracle OLAP supported by Oracle Business Intelligence.

Working with Oracle OLAP Analytic Workspace (AW) Objects

You can view Oracle OLAP Analytic Workspace (AW) objects in the Physical layer of the Administration Tool. These objects correspond to the analytic workspace object in the Oracle OLAP metadata, and are similar to physical catalog and physical schema objects. Analytic workspaces are containers for storing related cubes. You create dimensions, cubes, and other dimensional objects within the context of an analytic workspace.

Oracle OLAP Analytic Workspace objects have properties for **Name**, **Description**, and **Dynamic Name**. You can use the Dynamic Name tab to provide a variable that specifies the name of the Analytic Workspace object. Note that the Dynamic Name tab is not active unless at least one session variable is defined. See "Using a Variable to Specify the Name of a Catalog or Schema" for more information.

Working with Oracle OLAP Dimensions, Hierarchies, and Levels

Oracle OLAP dimensions are lists of unique values that identify and categorize data. They form the edges of a cube, and thus of the measures within the cube. In a report, the dimension values (or their descriptive attributes) provide labels for the rows and columns.

There are three types of Oracle OLAP dimensions:

- **Level-based dimensions.** Members of level-based dimensions naturally group into levels based on their type, such as "month" and "year." Most dimensions are level-based.
- **Value-based dimensions.** These dimensions have parent-child relationships among their members, but the members are all the same type (like 'Employee' or 'Account'), so these relationships do not form meaningful levels.
- **List or flat dimensions.** These dimensions have no levels or hierarchies.

> **Note:** Oracle Business Intelligence does not support dimensions that have no hierarchies (flat dimensions). Importing flat dimensions from an Oracle OLAP data source will result in an error. If you have flat dimensions, replace them with single-level hierarchies in the data source before importing them into Oracle Business Intelligence.

On the General tab of the Oracle OLAP Dimension dialog, you can view and edit the name and description of the dimension, along with the following dimension properties:

- **Time.** Indicates that this dimension is a time dimension.
- **Ragged.** Indicates that this dimension contains a hierarchy that has at least one member with a different base, creating a "ragged" base level for the hierarchy.
- **Skipped levels.** Indicates that this dimension contains a hierarchy that has at least one member whose parents are more than one level above it, creating a hole in the hierarchy. An example of a skip-level hierarchy is City-State-Country, where at least one city has a country as its parent (for example, Washington D.C. in the United States).
- **External Name.** The physical name that is used when referencing the dimension in physical SQL queries. This value must reflect the external name defined in the data source.
- **Cache properties.** Select **Cacheable** to include this dimension in the Oracle BI Server query cache. To specify that cache entries do not expire, select **Cache never expires**. Alternatively, you can select **Cache persistence time** and enter a value to specify how long entries should persist in the query cache. Note that if a query references multiple physical objects with different persistence times, the cache entry for the query exists for the shortest persistence time set for any of the tables referenced in the query. This makes sure that no subsequent query gets a cache hit from an expired cache entry.

The Columns and Hierarchies tabs of the Oracle OLAP Dimension dialog list the dimension members and hierarchies that belong to the dimension. In the Columns tab, you can add or remove columns, and edit particular columns. In the Hierarchies tab, you can add, remove, or edit hierarchies. You can also use the type (key) button to select the default hierarchy for the dimension.

Dimensions can contain one or more hierarchies. Most hierarchies are level-based and consist of one or more levels of aggregation. Members roll up into the next higher level in a many-to-one relationship, and these members roll up into the next higher level, and so forth to the top level. Ragged and skip-level hierarchies are also supported.

Dimensions can also contain value-based hierarchies, which are parent-child hierarchies that do not support levels. For example, an employee dimension might have a parent-child relationship that identifies each employee's supervisor. However,

levels that group together first-, second-, and third-level supervisors and so forth may not be meaningful for analysis.

For value-based hierarchies, the Nullable option is selected by default for the root member physical cube column. This option must be selected for the root member for value-based hierarchies to work correctly.

Multiple hierarchies for a dimension typically share the base-level dimension members and branch into separate hierarchies. They can share the top level if they use all the same base members and use the same aggregation operators. Otherwise, they need different top levels to store different aggregate values.

In the Oracle OLAP Hierarchy dialog, you can view and edit the name, external name, and description of the hierarchy. For level-based hierarchies, you can add, remove, edit, or reorder levels. For value-based hierarchies, you can add, remove, or edit columns. To specify a key column, double-click a column name.

In the Oracle OLAP Level dialog, you can view and edit the name, external name, and description of the level. You can also add, remove, or edit columns for that level. To designate a column as a level key, double-click a column name.

Working with Oracle OLAP Cubes and Columns

Oracle OLAP cubes are informational objects that identify measures with the exact same dimensions and thus are candidates for being processed together at all stages: data loading, aggregation, storage, and querying.

Cubes define the shape of the business measures. They are defined by a set of ordered dimensions. The dimensions form the edges of a cube, and the measures are the cells in the body of the cube.

Oracle OLAP cubes have properties similar to other cubes. On the General tab of the Oracle OLAP Cube dialog, you can view and edit the name and description of the cube, along with the following cube properties:

- **External Name.** The physical name that is used when referencing the cube in physical SQL queries. This value must reflect the external name defined in the data source.
- **Density and Materialization.** For Oracle OLAP 10*g* cubes that are sparse and fully materialized, specify values for these properties to optimize queries. If you set the **Density** option to **Sparse** and the **Materialization** option to **Fully Materialized**, the Oracle BI Server generates a loop clause to skip empty cells. Note that if you leave the **Density** option blank, the Oracle BI Server assumes the data is sparse.

  If you set these options, make sure that you set them to reflect the actual properties of the data source. Do not specify that the data is sparse and fully materialized unless this is true for the data source.

  You do not need to set these values for Oracle OLAP 11*g* cubes. For these objects, optimization happens automatically.
- **Cache properties.** Select **Cacheable** to include this cube in the Oracle BI Server query cache. To specify that cache entries do not expire, select **Cache never expires**. Alternatively, you can select **Cache persistence time** and enter a value to specify how long entries should persist in the query cache. Note that if a query references multiple physical objects with different persistence times, the cache entry for the query exists for the shortest persistence time set for any of the tables referenced in the query. This makes sure that no subsequent query gets a cache hit from an expired cache entry.

The Columns tab of the Oracle OLAP Cube dialog lists the columns that belong to the cube. You can add or remove columns, and edit particular columns.

Oracle OLAP columns can be measures, calculated measures, attributes, or level keys. Oracle OLAP columns have the same properties as other physical columns. See "Creating and Editing a Column in a Physical Table" for more information about physical column properties like **Type**, **Length**, and **Nullable**.

Working with Physical Foreign Keys and Joins

You can create physical foreign keys and complex joins using either the Physical Diagram, or the Joins Manager. Note that you do not create joins for multidimensional data sources.

This section contains the following topics:

- About Physical Joins
- Defining Physical Joins with the Physical Diagram
- Defining Physical Joins with the Joins Manager

About Physical Joins

When you import keys in a physical schema, the primary key-foreign key joins are automatically defined. Any other joins within each data source or between data sources have to be explicitly defined to express relationships between tables in the Physical layer.

Imported key and foreign key joins do not have to be used in metadata. Joins that are defined to enforce referential integrity constraints can result in incorrect joins being specified in queries. For example, joins between a multipurpose lookup table and several other tables can result in unnecessary or invalid circular joins in the SQL queries issued by the Oracle BI Server.

This section contains the following topics:

- About Primary Key and Foreign Key Relationships
- About Complex Joins
- About Multi-Database Joins
- About Fragmented Data

About Primary Key and Foreign Key Relationships

A primary key and foreign key relationship defines a one-to-many relationship between two tables.

A foreign key is a column or a set of columns in one table that references the primary key columns in another table. The primary key is defined as a column or set of columns where each value is unique and identifies a single row of the table.

Note that there are two cases where multiple foreign key columns in a table point to the same table:

- When the primary key of the foreign table is "concatenated," meaning that it consists of a set of columns. This is a single join between two tables that happens to use multiple columns.
- When you have created an alias to the foreign table, because the foreign table needs to serve in different roles. In this case, each foreign key joins to a primary

key in one role-playing alias or the other. See"About Physical Alias Tables" for more information.

You can specify primary key and foreign keys in the Physical Diagram, or by using the Keys and Foreign Keys tabs of the Physical Table dialog. Also refer to "Defining Physical Joins with the Physical Diagram" and "Creating and Managing Columns and Keys for Relational and Cube Tables" for more information.

About Complex Joins

In the Physical layer of the repository, complex joins are joins over nonforeign key and primary key columns. In other words, physical complex joins are joins that use an expression rather than key column relationships.

When you create a complex join in the Physical layer, you specify the expression for the join.

For most data sources, foreign key joins are preferred for performance reasons. Complex joins are usually not as performant because they do not use key column relationships to form the join. The exception is ADF data sources, which use physical complex joins exclusively to denote ViewLink instances that connect pairs of View Objects in the ADF model.

About Multi-Database Joins

A multi-database join is defined as a table under one metadata database object that joins to a table under a different metadata database object. You need to specify multi-database joins to combine the data from different databases.

Use the Physical Diagram to specify multi-database joins. See "Defining Physical Joins with the Physical Diagram" for more information.

Multi-database joins can be created between tables in most types of databases and are performed within the Oracle BI Server. Note that you cannot create multi-database joins to tables in Oracle OLAP data sources.

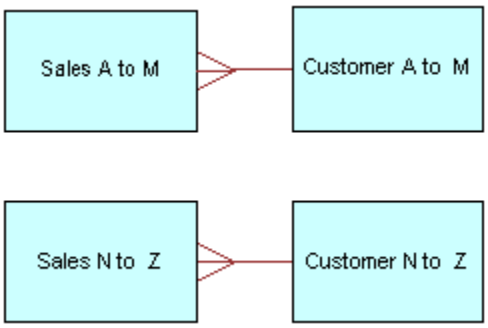
While the Oracle BI Server has several strategies for optimizing the performance of multi-database joins, these joins are significantly slower than joins between tables within the same database. For this reason, avoid them whenever possible.

About Fragmented Data

Fragmented data is data from a single domain that is split between multiple tables.

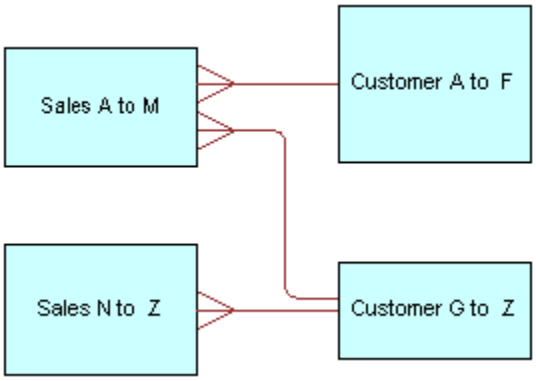
For example, a data source might store sales data for customers with last names beginning with the letter A through M in one table and last names from N through Z in another table. With fragmented tables, you need to define all of the join conditions between each fragment and all the tables to which it relates. Figure 8–7 shows the physical joins with a fragmented sales table and a fragmented customer table where they are fragmented the same way (A through M and N through Z).

Figure 8–7 Fragmented Tables Example



In some cases, you might have a fragmented fact table and a fragmented dimension table, but the fragments might be across different values. In this case, in addition to the joins created in Figure 8–7, you need to define a one-to-many join from Customer A to F and from Customer G to Z to Sales A to M, as shown in Figure 8–8.

Figure 8–8 Joins for Fragmented Tables Example



Note: Avoid adding join conditions where they are not necessary (for example, between Sales A to M and Customer N to Z in Figure 8–7). Extra join conditions can cause performance degradations.

Defining Physical Joins with the Physical Diagram

You can define foreign keys and complex joins between tables, whether or not the tables are in the same data source. When you use the Physical Diagram to create joins, the Administration Tool determines what type of join to create based on the selected object types and the join expression.

If you do not want the Administration Tool to automatically determine what type of join to create, use the Joins manager to explicitly create the join. See "Defining Physical Joins with the Joins Manager" for more information.

To define a physical foreign key join or a complex join with the Physical Diagram:

1. In the Physical layer of the Administration Tool, select one or more tables and choose one of the Physical Diagram commands from the right-click menu.
2. Click the **New Join** button on the Administration Tool toolbar:

3. In the Physical Diagram, left-click the first table in the join (the table representing many in the one-to-many join) to select it.
4. Move the cursor to the table to which you want to join (the table representing one in the one-to-many join) and left-click the second table to select it.

   The Physical Foreign Key dialog appears. Although physical foreign key joins are the default join type, the object type might change to a complex join after you define the join and click **OK**, depending on the join information.
5. Select the joining columns from the left and the right tables.

   The SQL join conditions appear in the expression pane.

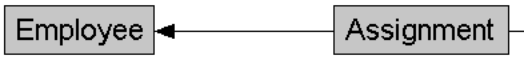
   The driving table option is shown in this dialog, but it is not available for selection because the Oracle BI Server implements driving tables only in the Business Model and Mapping layer. See "Specifying a Driving Table" for more information about driving tables.
6. For complex joins, you can optionally set the cardinality for each side of the join (for example, **N**, **0,1**, **1**, or **Unknown**).

   To set the cardinality to unknown, you only need to select **Unknown** for one side of the join. For example, choosing unknown-to-1 is equivalent to unknown-to-unknown and appears as such the next time you open the dialog for this join.
7. If appropriate, specify a database hint. See "Using Hints in SQL Statements" for more information.
8. If you are creating a complex join for ADF ViewObject or ViewLink instances, specify the ViewLink instance name or the ViewLink definition name in the **ViewLink Name** field.
9. To open Expression Builder, click the button to the right of the Expression pane. The expression displays in the Expression pane.

   The default join expression for ViewObject or ViewLink instances is arbitrary and has no meaning.
10. Click **OK** to apply the selections.

    In the Physical Diagram, the join is represented by a line between the two selected tables, with an arrow at the "one" end of the join. Figure 8–9 shows a join in the Physical Diagram.

Figure 8–9 Join in the Physical Diagram



Defining Physical Joins with the Joins Manager

You can use the Joins Manager to view join relationships and to create physical foreign key joins and complex joins.

To define a physical foreign key join or complex join with the Joins Manager:

1. In the Administration Tool toolbar, select **Manage**, then select **Joins**.
2. In the Joins Manager dialog, perform one of the following tasks:
   - Select **Action > New > Complex Join**.

The Complex Join dialog appears.

– Select **Action > New > Physical Foreign Key**. Then, in the Browse dialog, double-click a table.

3. In the Complex Join or Physical Foreign Key dialog, type a name for the join.
4. Click the **Browse** button for the **Table** field on the left side of the dialog, and locate the table that the foreign key references.
5. Select the columns in the left table that the key references.
6. Select the columns in the right table that comprise the foreign key columns.
7. For complex joins, you can optionally set the cardinality for each side of the join (for example, **N**, **0,1**, **1**, or **Unknown**).

   To set the cardinality to unknown, you only need to select **Unknown** for one side of the join. For example, choosing unknown-to-1 is equivalent to unknown-to-unknown and appears as such the next time you open the dialog for this join.
8. If appropriate, specify a database hint. See "Using Hints in SQL Statements" for more information.
9. If you are creating a complex join for ADF ViewObject or ViewLink instances, specify the ViewLink instance name or the ViewLink definition name in the **ViewLink Name** field.
10. To open Expression Builder, click the button to the right of the Expression pane. The expression displays in the Expression pane.

    The default join expression for ViewObject or ViewLink instances is arbitrary and has no meaning.
11. Click **OK** to save.

Deploying Opaque Views

An opaque view is a Physical layer table that consists of a `SELECT` statement. When you need a new table, you must create a physical table or a materialized view. Use an opaque view only if there is no other solution.

See Appendix G, "Exchanging Metadata with Databases to Enhance Query Performance" for more information about materialized views.

This section contains the following topics:

- About Deploying Opaque Views
- Deploying Opaque View Objects
- Undeploying a Deployed View
- When to Delete Opaque Views or Deployed Views
- When to Redeploy Opaque Views

About Deploying Opaque Views

In the repository, opaque views appear as view tables in the data source, but the view does not actually exist until you deploy it. You deploy an opaque view in the data source using the Deploy Views utility.

After deploying an opaque view, it is called a deployed view. Opaque views can be used without deploying them, but the Oracle BI Server has to generate a more complex query when an opaque view is encountered.

> **Note:** Data sources such as XLS and nonrelational data sources do not support opaque views and cannot run the view deployment utility.

To verify that opaque views are supported by a data source, check whether the `CREATE_VIEW_SUPPORTED` SQL feature is selected in the Database dialog, in the Features tab. See "Specifying SQL Features Supported by a Data Source" for instructions.

Deploying Opaque View Objects

In offline mode, the Deploy Views utility is available when importing from data sources with ODBC and DB2 CLI data sources. Oracle Native (client) drivers are also supported in the offline mode for deploying views. In online mode, view deployment is available for supported data sources using Import through server (the settings on the client are ignored).

Using the Create View SELECT Statement

The SQL statement for deploying opaque views in the Physical layer of the repository is available for supported data sources. To determine which data sources support opaque views, contact your system administrator or consult your data source documentation.

Use only repository variables in the definition. The system generates an error if the view definition contains a session variable.

Syntax

```
CREATE VIEW view_name AS select_statement,
```

Where:

- *select_statement* is the user-entered SQL statement in the opaque view object. If the SQL statement is invalid, the create view statement fails during view deployment.
- *view_name* is one of the two following formats: `schema.viewname`, or `viewname`. The connection pool settings determine if the schema name is added.

Note that if you want your SELECT statement to reference a row-wise initialization variable, then you must use the `VALUELISTOF` function. For example, to get the customers assigned to the user names in the variable `LIST_OF_USERS`, use the following syntax:

```
RW.CUSTOMERS.USER_NAME in (VALUELISTOF(NQ_SESSION.LIST_OF_USERS))
```

To filter by only specific values in the list, then use `ValueNameof` as show in the below example. Note that the first value is 0, not 1.

```
RW.CUSTOMERS.USER_NAME in '(ValueNameOf(0,NQ_SESSION.LIST_OF_USERS))
```

For opaque view objects, the right-click menu contains the **Deploy View(s)** option. When you select **Deploy View(s)**, the Create View SQL statement executes and attempts to create the deployed view objects. The following list describes how to initiate view deployment and the results of each method:

- Right-click a single opaque view object. When you select **Deploy View(s)**, the Create View SQL statement executes and attempts to create a deployed view for the object.
- Right-click several objects. If at least one of the selected objects is an opaque view object, the right-click menu contains the **Deploy View(s)** option. When you select **Deploy View(s)**, the Create View SQL statement executes and attempts to create the deployed views for any qualifying objects.
- Right-click a physical schema or physical catalog. If any opaque view object exists in the schema or catalog, the right-click menu contains the **Deploy View(s)** option. When you select **Deploy View(s)**, the Create View SQL statements for all qualifying objects execute and attempt to create deployed views for the qualifying objects contained in the selected schema or catalog.

During deployment, names are assigned to the views. If you change the preassigned name, the new name must be alphanumeric and no more than 18 characters. If these guidelines are not followed, the object name is automatically transformed to a valid name using the following Name Transform algorithm:

1. All non-alphanumeric characters are removed.
2. If there are 16 or more characters after Step 1, the first 16 characters are kept.
3. Two digits starting from 00 to 99 are appended to the name to make the name unique in the corresponding context.

After the deployment process completes, the following occurs:

- Views that have been successfully and unsuccessfully deployed appear in a list.
- For unsuccessful deployments, a brief reason appears in the list.
- If deployment is successful, the object type of the opaque view changes from Select to None and the deployed view is treated as a regular table.

  If you change the type back to Select, the associated opaque views are dropped from the data source, or an error message appears. See "When to Delete Opaque Views or Deployed Views" for information about deleting deployed views.
- In the Administration Tool, the view icon changes to the deployed view icon for successfully deployed views.

To deploy an opaque view:

1. In the Physical layer of the Administration Tool, right-click the opaque view that you want to deploy.
2. In the right-click menu, select **Deploy View(s)**.
3. In the View Deployment - Deploy View(s) dialog, perform the following steps:
   a. In the **New Table Name** column, optionally change the new deployed view names.

      If the change does not conform to the naming rules, a new name is assigned and the dialog appears again so that you can accept or change it. This action repeats until all names pass validation.

   b. If you do not want to deploy one or more of the views, clear the appropriate rows.
4. If there are multiple connection pools defined for the physical database, in the Select Connection Pool dialog, choose a connection pool and click **Select**.

The SQL statement (`CREATE VIEW`) executes, and the View Deployment Messages dialog appears.

5. In the View Deployment Messages dialog, search for views using **Find** and **Find Again**, or copy the contents.
6. When you are finished, click **OK**.

Undeploying a Deployed View

Running the Undeploy Views utility on a deployed view deletes the view and converts the view table back to an opaque view with its original `SELECT` statement.

To undeploy a deployed view:

1. In the Physical layer of the Administration Tool, right-click a physical database, catalog, schema, or table.

   If a deployed view exists that is related to the selected object, the right-click menu contains the **Undeploy View(s)** option.
2. Select **Undeploy View(s)**.

   A list of views to be undeployed appears.
3. If you do not want to undeploy one or more of the views, clear the appropriate rows.
4. In the View Deployment - Undeploy View(s) dialog, click **OK** to remove the views.

   A message appears if the undeployment was successful.
5. In the View Deployment Messages dialog, search for undeployed views using **Find** and **Find Again**, or copy the contents.
6. When you are finished, click **OK**.

When to Delete Opaque Views or Deployed Views

Use the following guidelines to remove opaque or deployed view objects in the repository:

- **Removing an undeployed opaque view in the repository.** If the opaque view has not been deployed, you can delete it from the repository.
- **Removing a deployed view.** When you deploy an opaque view, a view table is created physically in both the data source and the repository. Therefore, you must undeploy the view before deleting it. You use the Undeploy Views utility in the Administration Tool. This utility removes the opaque view from the back-end data source, changes the Table Type from None to Select, and restores the `SELECT` statement of the object in the Physical layer of repository.

> **Caution:** Do not manually delete the view table in the data source. If this table is deleted, then the Oracle BI Server cannot query the view object. When you undeploy the view, it is removed automatically from the data source.

When to Redeploy Opaque Views

After removing an opaque view, choose to redeploy it. The Administration Tool does not distinguish between a first-time deployment and a redeployment. Make sure that

you remove a deployed view before deploying the opaque view again. The deploy operation will fail and the data source will return error messages if you do not remove the deployed view before deploying the opaque view again.

Using Hints in SQL Statements

Hints are instructions that you place within a SQL statement that tell the data source query optimizer the most efficient way to execute the statement.

Hints override the optimizer's execution plan, so you can use hints to improve performance by forcing the optimizer to use a more efficient plan. Hints are only supported for Oracle Database data sources.

Using the Administration Tool, you can add hints to a repository, in both online and offline modes, to optimize the performance of queries. When you add a hint to the repository, you associate it with Physical layer objects. When the object associated with the hint is queried, the Oracle BI Server inserts the hint into the SQL statement.

Table 8–6 shows the physical objects with which you can associate hints. It also shows the Administration Tool dialog that corresponds to the physical object. Each of these dialogs contains a **Hint** field, into which you can type a hint to add it to the repository.

Table 8–6 Physical Layer Objects That Accept Hints

Database Object	Dialog
Complex join	Complex Join
Physical foreign key	Physical Foreign Key
Physical table	Physical Table - General tab

Hints are only supported when the **Table Type** is set to **Physical Table**. For other table types, the hint text is ignored. For physical tables with a table type of **Select**, you can provide the hint text as part of the SQL statement entered in the **Default Initialization String** field.

How to Use Oracle Hints

This section provides examples of how to use Oracle hints with the Oracle BI Server.

For more information about Oracle hints, see *Oracle Database SQL Language Reference* for the version of the Oracle Database that you use.

This section contains the following topics:

- About the Index Hint
- About the Leading Hint

About the Index Hint

The Index hint instructs the optimizer to scan a specified index rather than a table.

Example 8–1 explains how to use the Index hint.

Example 8–1 Index Hint

You find queries against the `ORDER_ITEMS` table to be slow. You review the execution plan of the query optimizer and find the `FAST_INDEX` index is not being used. You create an Index hint to force the optimizer to scan the `FAST_INDEX` index rather than the `ORDER_ITEMS` table. The syntax for the Index hint is `index(table_name, index_name)`.

To add this hint to the repository, go to the Physical Table dialog in the Administration Tool and type the following text in the **Hint** field:

```
index(ORDER_ITEMS, FAST_INDEX)
```

About the Leading Hint

The Leading hint forces the optimizer to build the join order of a query with a specified table.

The syntax for the Leading hint is `leading(table_name)`. If you were creating a foreign key join between the Products table and the Sales Fact table and wanted to force the optimizer to begin the join with the Products table, you would go to the Physical Foreign Key dialog in the Administration Tool and type the following text in the **Hint** field:

```
leading(Products)
```

About Performance Considerations for Hints

Hints that are well researched and planned can result in significantly better query performance. However, hints can also negatively affect performance if they result in a suboptimal execution plan.

Follow these guidelines to create hints to optimize query performance:

- Only add hints to a repository after you have tried to improve performance in the following ways:
  - Added physical indexes (or other physical changes) to the Oracle Database.
  - Made modeling changes within the server.
- Avoid creating hints for physical table and join objects that are queried often. If you drop or rename a physical object that is associated with a hint, you must also alter the hints accordingly.

Creating Hints

You can add hints to the repository using the Administration Tool.

To create a hint:

1. In the Administration Tool, go to one of the following dialogs:
   - Physical Table—General tab
   - Physical Foreign Key
   - Complex Join
2. Type the text of the hint in the **Hint** field and click **OK**.

   For a description of available Oracle hints and hint syntax, see *Oracle Database SQL Language Reference* for the version of the Oracle Database that you use.

   Note: Although hints are identified using SQL comment markers (/* or --), do not type SQL comment markers when you type the text of the hint. The Oracle BI Server inserts the comment markers when the hint is executed.

Displaying and Updating Row Counts for Physical Tables and Columns

When you request row counts, the Administration Tool retrieves the number of rows from the data source for all or selected tables and columns (distinct values are retrieved for columns) and stores those values in the repository. The time this process takes depends upon the number of row counts retrieved.

When updating all row counts, the Updating Row Counts window appears while row counts are retrieved and stored. If you click **Cancel**, the retrieve process stops after the in-process table (and its columns) have been retrieved. Row counts include all tables and columns for which values were retrieved before the cancel operation.

Updating all row counts for a large repository might take a long time to complete. Therefore, you sometimes might want to update only selected table and column counts.

Row counts are not available for the following:

- Stored Procedure object types
- XML data sources and XML Server data sources
- Multidimensional data sources
- Data sources that do not support the `COUNTDISTINCT` function, such as Microsoft Access and Microsoft Excel, or data sources for which `COUNT_STAR_SUPPORTED` has been disabled in the database features table
- In online mode, Update Row Count does not work with connection pools in which the session variables `:USER` and `:PASSWORD` are set as the user name and password.

  In offline mode, the Set values for variables dialog appears so that you can populate the session variables `:USER` and `:PASSWORD`.
- In online mode, after importing or manually creating a physical table or column, the Oracle BI Server does not recognize the new objects until you check them in. Therefore, **Update Row Count** is not available in the menu until you check in these objects.

To display row counts in the Physical layer:

1. In the Administration Tool, select **Tools**, then select **Options**.
2. In the General tab of the Options dialog, select **Show row count in physical view**, and click **OK**.

To update selected row counts in the Physical layer:

1. In the Physical layer of the Administration Tool, right-click a single table or column. You can select multiple objects and right-click.
2. In the shortcut menu, select **Update Row Count**.

To update all row counts in the Physical layer:

1. In the Administration Tool, select **Tools**, then select **Update All Row Counts**. If the repository is open in online mode, the Check Out Objects window might open.
2. Click **Yes** to check out the objects.

   Any row counts that have changed since the last update are refreshed.

9

Working with Logical Tables, Joins, and Columns

This chapter explains how to work with objects in the Business Model and Mapping layer of the Oracle BI repository, such as logical tables, joins, and columns. It also explains other Business Model and Mapping layer concepts like display folders, bridge tables, the Business Model Diagram, and how to enable write back on columns.

This chapter contains the following sections:

About Working with the Business Model and Mapping Layer

The Business Model and Mapping layer of the Oracle BI repository defines the business, or logical, model of the data and specifies the mapping between the business model and the Physical layer schemas.

Business models are always dimensional, unlike objects in the Physical layer, which reflect the organization of the data sources. The Business Model and Mapping layer can contain one or more business models. Each business model contains logical tables, columns, and joins.

Even though similar terminology is used for logical table and physical table objects, such as the concept of keys, logical tables and joins in the Business Model and Mapping layer have their own set of rules that differ from those of relational models. For example, logical fact tables are not required to have keys, and logical joins can represent many possible physical joins.

Logical tables, joins, mappings, and other objects in the Business Model and Mapping layer are typically created automatically when you drag and drop objects from the Physical layer to a particular business model. After these objects have been created, you can perform tasks like creating additional logical joins, performing calculations and transformations on columns, and adding and removing keys from dimension and fact tables.

Creating the Business Model and Mapping Layer

After creating all of the elements of the Physical layer, you can drag tables or columns from the Physical layer to a business model in the Business Model and Mapping layer to create logical objects in the metadata.

This section contains the following topics:

Creating Business Models

The Business Model and Mapping layer of the Administration Tool can contain one or more business models. A business model contains the business model definitions and the mappings from logical to physical tables for the business model.

When you work in a repository in offline mode, remember to save your repository from time to time. You can save a repository in offline mode even though the business models may be inconsistent.

To create a business model:

1. In the Administration Tool, right-click in the Business Model and Mapping layer below any existing objects.
2. Select the option **New Business Model** from the shortcut menu.
3. Specify a name for the business model.
4. New business models are disabled by default. If you want to make the corresponding Presentation layer available for queries, deselect **Disabled**.

   Note: The business model should be consistent before you deselect this option.

5. Optionally, type a description of the business model.
6. Click **OK**.

After you create a business model, you can create business model objects by dragging and dropping objects from the Physical layer. See the next section for more information.

Automatically Creating Business Model Objects

To automatically map objects in the Business Model and Mapping layer to sources in the Physical layer, you can drag and drop Physical layer objects to a particular business model in the logical layer.

When you drag a physical table to the Business Model and Mapping layer, a corresponding logical table is created. For each physical column in the table, a corresponding logical column is created. If you drag multiple tables at once, a logical join is created for each physical join, but only the first time the tables are dragged onto a new business model.

Automatically Creating Business Model Objects for Multidimensional Data Sources

Setting up objects in the Business Model and Mapping layer for multidimensional data sources is similar to setting up logical layer objects for a relational data source. To create the business model layer, you can drag and drop the Physical layer cube to the logical layer. Oracle Business Intelligence automatically creates a fully configured and consistent business model that retains metrics, attributes and dimensions.

Note: For Essbase data sources, it is recommended that you create a separate business model for each Essbase cube. To do this, drag each cube individually to the Business Model and Mapping layer.

Duplicating a Business Model and Subject Area

This feature lets you select a business model and its corresponding subject area (or a subject area and its corresponding business model), make a copy, and assign new names to the duplicates. Note that aliases are not copied.

To copy a business model and subject area:

1. Perform one of the following steps:
   - In the Business Model and Mapping layer of the Administration Tool, right-click a business model and select **Duplicate with Subject Area**.
   - In the Presentation layer of the Administration Tool, right-click a subject area and select **Duplicate with Business Model**.
2. In the Copy Business Model and Subject Area dialog, select the business model and corresponding subject area you want to copy.
3. Specify new names for the business model and subject area in the appropriate name fields, and then click **OK**.

   The copied business model appears in the Business Model and Mapping layer, and the copied subject area appears in the Presentation layer.

About Working with the Business Model Diagram

In addition to working with Business Model and Mapping layer objects in the middle pane of the Administration Tool, you can open the Business Model Diagram to see a graphical model of logical tables and joins.

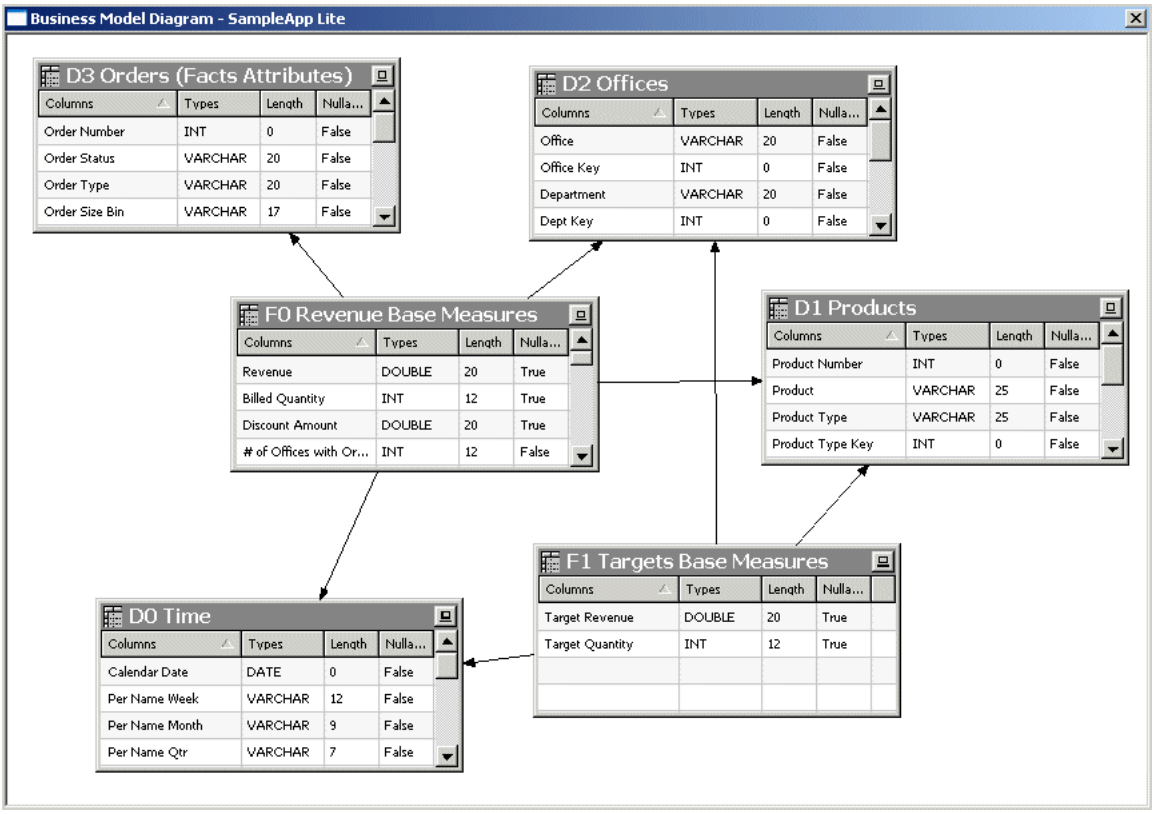
To access the Business Model Diagram, right-click an object in the Business Model and Mapping layer (such as a dimension or fact table) and select **Business Model Diagram**. Then, select one of the following options:

- **Whole Diagram.** Displays all logical tables and joins in the business model.
- **Selected Tables Only.** Displays only the selected logical tables. Logical joins appear only if they exist between the objects that you selected. This option is only available when you select one or more logical tables.

- **Selected Tables and Direct Joins.** Displays the selected logical tables and any logical tables that join to the tables that you selected. This option is only available when you select one or more logical tables.
- **Selected Fact Tables and Dimensions.** Displays the selected logical tables and their associated logical dimensions. This option is only available when your selection includes at least one fact table.

Note that the Business Model Diagram displays only logical tables and joins. It does not display other Business Model and Mapping layer objects, such as business models, dimensions, or hierarchies. Joins are represented by a line with an arrow at the "one" end of the join.

Figure 9–1 Business Model Diagram



To add additional tables to the Business Model Diagram, leave the Business Model Diagram window open and then right-click the table or tables you want to add. Then, select **Business Model Diagram** and choose one of the display options.

Additional options are available in the right-click menu for the graphical tables and joins displayed in the Business Model Diagram. For example, you can delete objects or view their properties, or you can add additional related objects using the right-click options **Add Direct Joins**, **Add Tables Joined to Whole Selection**, and **Add All Joins**. You can also select **Find in Tree View** to locate a particular object in the Business Model and Mapping layer view in the middle pane, or check out objects in online mode.

You can also right-click an object in the Business Model Diagram view and select **Hide** to hide particular objects in the diagram. Note that this effect is temporary and does not persist.

Use the **Print** and **Print Preview** options on the File menu to manage printing options for the Business Model Diagram. You can also use the **Print** option on the toolbar.

See also the following sections:

- "Using the Physical and Business Model Diagrams" for information about zooming, panning, and controlling the layout of the tables
- "Defining Logical Joins with the Business Model Diagram" for information about defining logical joins

Creating and Managing Logical Tables

Logical tables exist in the Business Model and Mapping layer. The logical schema defined in each business model must contain at least two logical tables, and you must define relationships between them.

Each logical table has one or more logical columns and one or more logical table sources associated with it. You can change the logical table name, reorder the logical table sources, and configure the logical keys, both primary and foreign.

This section contains the following topics:

Creating Logical Tables

Typically, you create logical tables by dragging and dropping a physical table from the Physical layer to a business model in the Business Model and Mapping layer. If a table does not exist in your physical schema, you need to create the logical table manually.

Drag and drop operations are usually the fastest method for creating objects in the Business Model and Mapping layer. If you drag and drop physical tables from the Physical layer to the Business Model and Mapping layer, the columns belonging to the table are also copied. After you drag and drop objects into the Business Model and Mapping layer, you can modify them in any way necessary without affecting the objects in the Physical layer.

When you drag physical tables (with key and foreign key relationships defined) to a business model, logical keys and joins are created that mirror the keys and joins in the Physical layer. This occurs only if the tables that you drag include the table with the foreign keys. Additionally, if you create new tables or subsequently drag additional tables from the Physical layer to the Business Model and Mapping layer, the logical mappings between the new or newly dragged tables and the previously dragged tables must be created manually.

See "Defining Logical Joins with the Joins Manager" and "Defining Logical Joins with the Business Model Diagram" for more information about joins.

To create a logical table by dragging and dropping:

1. In the Administration Tool, select one or more table objects in the Physical layer.

   You must include the table with the foreign keys if you want to preserve the keys and joins from the Physical layer.

2. Drag and drop the table objects to a business model in the Business Model and Mapping layer.

When you drop them, the table objects, including the physical source mappings, are created automatically in the Business Model and Mapping layer.

To create a logical table manually:

1. In the Business Model and Mapping layer of the Administration Tool, right-click the business model in which you want to create the table and select **New Object > Logical Table**.

   The Logical Table dialog appears.

2. In the General tab, type a name for the logical table.
3. If this is a lookup table, select the option **Lookup table**. A lookup table stores multilingual data corresponding to rows in the base tables. See "Localizing Oracle Business Intelligence" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about localization and lookup tables.
4. Optionally, type a description of the table.
5. Click **OK**.

After creating a logical table manually, you must create all keys and joins manually.

Creating and Managing Logical Table Sources

You can add a new logical table source, edit or delete an existing table source, create or change mappings to the table source, and define when to use logical tables sources and how content is aggregated.

See Chapter 11, "Managing Logical Table Sources (Mappings)" for instructions about how to perform these tasks.

Specifying a Primary Key in a Logical Table

After creating tables in the Business Model and Mapping layer, you specify a primary key for each dimension table. Logical dimension tables must have a logical primary key. Logical keys can be composed of one or more logical columns.

Note: Oracle recommends that you do not specify logical keys for logical fact tables.

To specify a primary key in a logical table:

1. In the Business Model and Mapping layer of the Administration Tool, double-click a table.
2. In the Logical Table dialog, select the Keys tab and then click **New**.
3. In the Logical Key dialog, type a name for the key and select the column that defines the key of the logical table.
4. Click **OK**.

Reviewing Foreign Keys for a Logical Table

Oracle recommends that you do not use foreign key joins in logical tables. If you must create these joins, you must first enable the option **Allow logical foreign key join creation** in the Options dialog.

See "Creating Logical Foreign Key Joins with the Joins Manager" for more information.

The Foreign Keys tab of the Logical Table dialog exists so that you can view logical foreign keys you might have had in a previous release of Oracle Business Intelligence.

Defining Logical Joins

Relationships between logical tables are expressed by logical joins. Logical joins are *conceptual*, rather than physical, joins. In other words, they do not join to particular keys or columns. A single logical join can correspond to many possible physical joins.

A key property of a logical join is cardinality. Cardinality expresses how rows in one table are related to rows in the table to which it is joined. A one-to-many cardinality means that for every row in the first logical dimension table, there are 0, 1, or many rows in the second logical table. The Administration Tool considers a table to be a logical fact table if it is at the Many end of all logical joins that connect it to other logical tables.

Specifying the logical table joins is required so that the Oracle BI Server can have the necessary metadata to translate a logical request against the business model to SQL queries against the physical data sources. The logical join information provides the Oracle BI Server with the many-to-one relationships between the logical tables. This logical join information is used when the Oracle BI Server generates queries against the underlying databases.

You do not need to create logical joins in the Business Model and Mapping layer if both of the following statements are true:

- You create the logical tables by simultaneously dragging and dropping all required physical tables to the Business Model and Mapping layer.
- The logical joins are the same as the joins in the Physical layer.

However, you will probably have to create some logical joins in the Business Model and Mapping layer, because you will rarely drag and drop all physical tables simultaneously except in very simple models.

You can create logical joins using either the Joins Manager or the Business Model Diagram. When you create a complex join in the Physical layer, you can specify expressions and the specific columns on which to create the join. When you create a logical join in the Business Model and Mapping layer, you cannot specify expressions or columns on which to create the join. The existence of a join in the Physical layer does not require a matching join in the Business Model and Mapping layer.

Note: It is recommended that you do not have foreign keys for logical tables. However, for backward compatibility, you can create logical foreign key joins using the Joins Manager if you select **Allow logical foreign key join creation** in the Options dialog.

A logical key for a fact table must be made up of the key columns that join to the attribute tables. Logical foreign key joins may be needed if the Oracle BI Server is to be used as an ODBC data source for certain third-party query and reporting tools.

This section contains the following topics:

- Defining Logical Joins with the Business Model Diagram
- Defining Logical Joins with the Joins Manager

- Specifying a Driving Table
- Factors That Determine Join Trimming
- Identifying Physical Tables That Map to Logical Objects

Defining Logical Joins with the Business Model Diagram

The Business Model Diagram shows logical tables and any defined joins between them. You can use the Business Model Diagram to define logical joins between tables.

To define a logical join with the Business Model Diagram:

1. In the Administration Tool, right-click a business model and select **Business Model Diagram**, then select **Whole Diagram**.
2. Click the **New Join** button on the Administration Tool toolbar:

3. In the Business Model Diagram, left-click the first table in the join (the table representing many in the one-to-many join) to select it.
4. Move the cursor to the table to which you want to join (the table representing one in the one-to-many join), and then left-click the second table to select it.

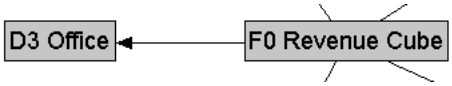
   The Logical Join dialog appears.
5. (Optional) To specify a driving table for the key, select a table from the **Driving table** list, and an applicable cardinality.

   This option is useful for optimizing the manner in which the Oracle BI Server processes multi-database inner joins when one table is very small and the other table is very large. Do not select a driving table unless multi-database joins are going to occur. See "Specifying a Driving Table" for more information about driving tables.

   Caution: Use extreme caution in deciding whether to specify a driving table. Driving tables are used for query optimization only under rare circumstances and when the driving table is extremely small (fewer than 1000 rows). Choosing a driving table incorrectly can lead to severe performance degradation.
6. Select the join type from the **Type** list, or keep the default value.
7. Set the **Cardinality** for each side of the join, or keep the default values.
8. Click **OK** to save your work.

   In the Business Model Diagram, the join is represented by a line between the two selected tables, with an arrow at the "one" end of the join. Figure 9–2 shows a join in the Business Model Diagram.

Figure 9–2 Join in the Business Model Diagram



Defining Logical Joins with the Joins Manager

You can use the Joins Manager to view logical join relationships and to create logical joins. You can also use the Joins Manager to create logical foreign key joins if you select **Allow logical foreign key join creation** in the Options dialog, although this is not recommended.

This section contains the following topics:

Creating Logical Joins with the Joins Manager

Logical joins are recommended over logical foreign key joins in the Business Model and Mapping layer.

To create a logical join with the Joins Manager:

1. In the Administration Tool, select **Manage**, then select **Joins**.

   The Joins Manager dialog appears.

2. Select **Action > New > Logical Join**.

   The Logical Join dialog appears.

3. Type a name for the logical join.

4. In the **Table** lists on the left and right side of the dialog, select the tables that the logical join references.

5. (Optional) To specify a driving table for the key, select a table from the **Driving** list, and an applicable cardinality.

   This option is useful for optimizing the manner in which the Oracle BI Server processes multi-database inner joins when one table is very small and the other table is very large. Do not select a driving table unless multi-database joins are going to occur. See "Specifying a Driving Table" for more information about driving tables.

   Caution: Use extreme caution in deciding whether to specify a driving table. Driving tables are used for query optimization only under rare circumstances and when the driving table is extremely small, that is, less than 1000 rows. Choosing a driving table incorrectly can lead to severe performance degradation.

6. Select the join type from the **Type** list, or keep the default value.

7. Set the **Cardinality** for each side of the join, or keep the default values.

8. Click **OK**.

Creating Logical Foreign Key Joins with the Joins Manager

Logical foreign key joins might be needed if the Oracle BI Server is to be used as an ODBC data source for certain third-party query and reporting tools. Typically, you should not create logical foreign keys.

To create a logical foreign key join with the Joins Manager:

1. In the Administration Tool, select **Tools**, then select **Options**.

2. In the General tab of the Options dialog, select **Allow logical foreign key join creation**.
3. Click **OK**.
4. Select **Manage**, then select **Joins** to display the Joins Manager.
5. Select **Action > New > Logical Foreign Key**.
6. In the Browse dialog, double-click a table to display the Logical Foreign Key dialog.
7. Type a name for the foreign key.
8. In the **Table** list on the left side of the dialog, select the table that the foreign key references.
9. Select the columns in the left table that the foreign key references.
10. Select the columns in the right table that make up the foreign key columns.
11. (Optional) To specify a driving table for the key, select a table from the **Driving** list, and an applicable cardinality.

    This option is useful for optimizing the manner in which the Oracle BI Server processes multi-database inner joins when one table is very small and the other table is very large. Do not select a driving table unless multi-database joins are going to occur. See "Specifying a Driving Table" for more information about driving tables.

    Caution: Use extreme caution in deciding whether to specify a driving table. Driving tables are used for query optimization only under rare circumstances and when the driving table is extremely small, that is, less than 1000 rows. Choosing a driving table incorrectly can lead to severe performance degradation.

12. Select the join type from the **Type** list, or keep the default value.
13. Set the **Cardinality** for each side of the join, or keep the default values.
14. Enter an expression for the join, or click the **Expression Builder** button to define the expression in Expression Builder.
15. Click **OK** to save your work.

Specifying a Driving Table

Driving tables are useful for optimizing how the Oracle BI Server processes cross-database joins when one table is very small and the other table is very large. Specifying driving tables leads to query optimization only when the number of rows being selected from the driving table is much smaller than the number of rows in the table to which it is being joined.

Caution: To avoid problems, only specify driving tables when the driving table is extremely small - less than 1000 rows.

You can specify a driving table for logical joins from the Logical Joins window. When you specify a driving table, the Oracle BI Server uses it if the query plan determines that its use will optimize query processing. The small table (the driving table) is

scanned, and parameterized queries are issued to the large table to select matching rows. The other tables, including other driving tables, are then joined together.

> **Caution:** If large numbers of rows are being selected from the driving table, specifying a driving table could lead to significant performance degradation or, if the `MAX_QUERIES_PER_DRIVE_JOIN` limit is exceeded, the query terminates.

In general, driving tables can be used with inner joins, and for outer joins when the driving table is the left table for a left outer join, or the right table for a right outer join. Driving tables are not used for full outer joins. See "Defining Logical Joins" for instructions on specifying a driving table.

There are two entries in the database features table that control and tune driving table performance.

- `MAX_PARAMETERS_PER_DRIVE_JOIN`

  This is a performance tuning parameter. In general, the larger its value, the fewer parameterized queries need to be generated. Values that are too large can result in parameterized queries that fail due to back-end database limitations. Setting the value to 0 (zero) turns off drive table joins.

- `MAX_QUERIES_PER_DRIVE_JOIN`

  This is used to prevent runaway drive table joins. If the number of parameterized queries exceeds its value, the query is terminated and an error message is returned to the user.

Factors That Determine Join Trimming

When determining which joins can be trimmed from a physical query, the Oracle BI Server considers the factors described in this section.

The following join trimming rules are enforced for tables within a logical table source:

- Join Outerness (Inner, Left Outer, Right Outer, or Full Outer).
- Join Cardinality ({0..1, 1, N, Unknown} to {0..1, 1, N, Unknown}; for example, 0..1 to N represents a zero or one to many join). There are nine join cardinality combinations excluding those with Unknown cardinality on at least one side of the join.
- Whether the logical table source contains a WHERE clause filter.
- Whether the physical join is a complex join or a foreign key join.

For the Oracle BI Server to trim a join, the following criteria must be met.

- The trimmed table must not be referenced anywhere in the query, such as in the projected list of columns or in the WHERE clause.
- The trimmed table must not cause the cardinality of the result set to change. If removing a join could potentially change the number of rows selected, then the Oracle BI Server will not trim it.

  A join is considered to have the potential to change the number of rows in the result set if any of the following conditions are true. If any of these conditions are true, then the join will not be trimmed from the query:

- The join is a full outer join (only inner joins, left outer joins, and right outer joins are candidates for trimming)
- The join cardinality is unknown on either side
- The table to be trimmed is on the many side of a join (in other words, the detail table can never be trimmed in a master-detail relationship)
- The table to be trimmed has a 0..1 cardinality and the join is an inner join. 0..1 cardinality implies that there might or might not be a matching row in the table. So, a join with 0..1 cardinality on one side is effectively like a filter. Therefore, it cannot be trimmed without changing the number of rows selected.
- The table to be trimmed is on the left side of a left outer join or on the right side of a right outer join (in other words, the row-preserving table is never trimmed). There is an exception to this rule for queries that select only attributes in which a DISTINCT clause is added to the query. Because of the DISTINCT clause, trimming the row-preserving table does not affect the number of rows returned from the null-supplying table. So, in the special case of distinct queries on attributes, the row-preserving table from an outer join can be trimmed.

Table 9–1 provides examples of when the Oracle BI Server can trim joins from the query.

Table 9–1 Join Trimming Examples

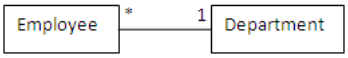
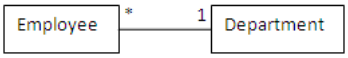
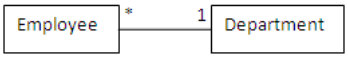
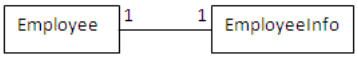
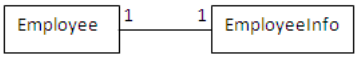
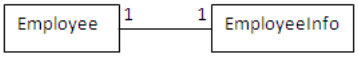
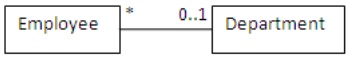
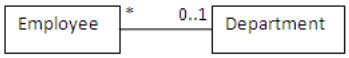
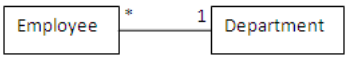
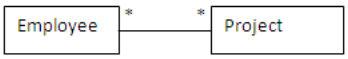
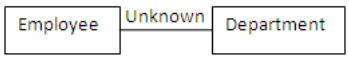
Scenario	Result
 Employee INNER JOIN Department	Department **can** be trimmed because it is on the "one" side of an inner join. Employee **cannot** be trimmed because it is on the many side of an inner join.
 Employee LEFT OUTER JOIN Department	Department **can** be trimmed because it is on the "one" side of the join and it is on the right side of a LEFT OUTER JOIN (in other words, the null supplying table). Employee **cannot** be trimmed because it is on the many side, and because it is on the left side of a LEFT OUTER JOIN (in other words, the row preserving table).
 Employee RIGHT OUTER JOIN Department	Department **cannot** be trimmed because it is on the right side of a RIGHT OUTER JOIN (in other words, the row preserving table). Employee **cannot** be trimmed because it is on the many side of the join.
 Employee INNER JOIN EmployeeInfo	**Either side can be trimmed** because both tables are on the "one" side of an inner join.
 Employee LEFT OUTER JOIN EmployeeInfo	EmployeeInfo **can** be trimmed since it is on the "one" side of the join, and it is on the right side of a LEFT OUTER JOIN (in other words, the null supplying table). Employee **cannot** be trimmed because it is on the left side of a LEFT OUTER JOIN (in other words, the row preserving table).

Table 9–1 (Cont.) Join Trimming Examples

Scenario	Result
 Employee RIGHT OUTER JOIN EmployeeInfo	EmployeeInfo **cannot** be trimmed because it is on the right side of a RIGHT OUTER JOIN (in other words, the row preserving table.) Employee **can** be trimmed because it is on the "one" side of the join, and it is on the left side of a RIGHT OUTER JOIN (in other words, the null supplying table).
 Employee INNER JOIN Department	Department **cannot** be trimmed because it is on the 0..1 side of an inner join. Employee **cannot** be trimmed because it is on the many side of an inner join.
 Employee LEFT OUTER JOIN Department	Department **can** be trimmed because it is on the 0..1 side of an outer join, and it is on the right side of a LEFT OUTER JOIN (in other words, the null supplying table). The Oracle BI Server allows the null supplying table on the 0..1 side of an outer join to be trimmed, because in this case, trimming Department from the query would not change the number of rows selected from the Employee table. Employee **cannot** be trimmed since it is on the many side of an outer join.
 Employee FULL OUTER JOIN Department	**Neither side can be trimmed** because the join is a FULL OUTER JOIN.
 Employee MANY TO MANY Project	**Neither side can be trimmed** because the join is many to many.
 Employee UNKNOWN Department	**Neither side can be trimmed** because the join has unknown cardinality.

Identifying Physical Tables That Map to Logical Objects

The Physical Diagram shows the physical tables that map to the selected logical object and the physical joins between each table.

One of the joins options, **Object(s) and Direct Joins within Business Model**, is unique to the logical layer. It creates a physical diagram of the tables that meet both of the following conditions:

- Tables in the selected objects and tables that join directly
- Tables that are mapped (exist in logical table sources in the business model) in the business model

To open the Physical Diagram for a logical object:

1. In the Business Model and Mapping layer of the Administration Tool, right-click a business model, logical table, or logical table source.

2. Select **Physical Diagram** and then one of the joins options.
3. Click and drag any object to more clearly view the relationship lines, such as one-to-many.

Creating and Managing Logical Columns

Many logical columns are automatically created by dragging tables from the Physical layer to the Business Model and Mapping layer. Other logical columns, especially ones that involve calculations based on other logical columns, can be created later.

Logical columns are displayed in a tree structure expanded out from the logical table to which they belong. If the column is a primary key column or participates in a primary key, the column is displayed with a key icon. If the column has an aggregation rule, it is displayed with a ruler icon. You can also reorder logical columns in the Business Model and Mapping layer.

This section contains the following topics:

Creating Logical Columns

The following procedure explains how to create logical columns in the Business Model and Mapping layer.

To create a logical column:

1. In the Business Model and Mapping layer, right-click a logical table.
2. From the shortcut menu, select **New Object**, then select **Logical Column**.
3. In the General tab, type a name for the logical column.

   The name of the business model and the associated logical table appear in the **Belongs to Table** field.
4. Select **Writeable** to enable write back for this column. See "Enabling Write Back On Columns" for more information.
5. Optionally, you can assign a different column on which to base the sort order for a column. See "Basing the Sort for a Logical Column on a Different Column" for details.
6. Optionally, you can assign a descriptor ID column for this column. See "Enabling Double Column Support by Assigning a Descriptor ID Column" for details.
7. Optionally, on the Column Source tab, you can specify that this logical column is derived from other logical columns. See "Creating Derived Columns" for details.
8. Optionally, on the Aggregation tab, you can set column aggregation. See "Setting Default Levels of Aggregation for Measure Columns" for details.

9. Optionally, on the Levels tab, you can associate attributes with a logical level. Measures can be associated with levels from multiple dimensions and always aggregate to the levels specified. See "Associating an Attribute with a Logical Level in Dimension Tables" for details.
10. Click **OK**.

Basing the Sort for a Logical Column on a Different Column

For a logical column, you can specify a different column on which to base the sort. This changes the sort order of a column when you do not want to order the values lexicographically.

Lexicographical sort arranges the results in alphabetic order such as in a dictionary. In this type of sort, numbers are ordered by their alphabetic spelling and not divided into a separate group.

For example, if you sorted on month (using a column such as `MONTH_NAME`), the results would be returned as February, January, March, and so on, in lexicographical sort order. However, you might want months to be sorted in chronological order. Therefore, your table should have a month key (such as `MONTH_KEY`) with values of 1 (January), 2 (February), 3 (March), and so on. To achieve the desired sort, you set the Sort order column field for the `MONTH_NAME` column to be `MONTH_KEY`. Then, a request to order by `MONTH_NAME` would return January, February, March, and so on.

Note that the sort column is automatically defined for Essbase data sources when business models are created by dragging and dropping cubes from the Physical layer.

To assign a different column on which to base the sort order for a column:

1. In the Logical Column dialog, in the General tab, click **Set** next to the **Sort order column** field.
2. In the Browse dialog, select a column.
3. To view the column details, click **View** to open the Logical Column dialog for that column, and then click **Cancel**.

   You can make some changes in this dialog. If you make changes, click **OK** to accept the changes instead of **Cancel**.
4. In the Browse dialog, click **OK**.

Enabling Double Column Support by Assigning a Descriptor ID Column

When multilingual columns are based on a lookup function, it is common to specify the non-translated lookup key column as the descriptor ID column of the translated column.

Assigning a descriptor ID column enables Double Column Support, a feature which helps in defining language-independent filters. For example, in Answers, users see the display column, but the query filters on the hidden descriptor ID column.

For more information, see "Supporting Multilingual Data" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

Note that double columns are also used for other purposes, like modeling spatial columns.

To assign a Descriptor ID column to a display column:

1. In the Logical Column dialog, in the General tab, click **Set** next to the **Descriptor ID column** field.

2. In the Browse dialog, select a key column.
3. To view the column details, click **View** to open the Logical Column dialog for that column, and then click **Cancel**.

   You can make some changes in this dialog. If you make changes, click **OK** to accept the changes instead of **Cancel**.
4. In the Browse dialog, click **OK**.

Creating Derived Columns

Some columns are derived from other logical columns as a way to apply post-aggregation calculations to measures.

To do this, you specify the derived column expression in the Column Source tab of the Logical Column dialog.

You can also create a set of derived columns using the Calculation Wizard. See "Using the Calculation Wizard" for more information.

Note that if the parameter `PREVENT_DIVIDE_BY_ZERO` is set to `YES` in NQSConfig.INI, the Oracle BI Server prevents errors in divide-by-zero situations, even for Answers column calculations. The Oracle BI Server creates a divide-by-zero prevention expression using `nullif()` or a similar function when it writes the physical SQL. Because of this, you do not have to use `CASE` statements to avoid divide-by-zero errors, as long as `PREVENT_DIVIDE_BY_ZERO` is set to `YES` (the default value).

See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about NQSConfig.INI settings.

You can also apply calculations pre-aggregation. See "Defining Physical to Logical Table Source Mappings and Creating Calculated Items" for more information.

To specify a derived column:

1. In the Logical Column dialog, select the Column Source tab.
2. Select the option **Derived from existing columns using an expression**.
3. Click the **Expression Builder** button to open Expression Builder.
4. In the Expression Builder - Derived logical column dialog, specify the expression from which the logical column should be derived.

   Note: To optimize performance, do not define aggregations in Expression Builder. Instead, use the Aggregation tab of the Logical Column dialog. See "Setting Default Levels of Aggregation for Measure Columns" for more information.
5. Click **OK**.

Note that you can display data from multilingual database schemas by using Expression Builder to create a lookup function. For more information, see "Supporting Multilingual Data" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

Configuring Logical Columns for Multicurrency Support

You can configure logical columns so that Oracle Business Intelligence users can select the currency in which they prefer to view currency columns in analyses and dashboards.

You can set up this feature so that all users see the same static list of currency options, or you can provide a dynamic list of currency options that changes based on a Logical SQL statement you specify.

To configure logical columns for multicurrency support:

1. Create a session variable named `PREFERRED_CURRENCY`, along with an initialization block to use in the variable. Make sure to select **Enable any user to set the value** when you create the session variable. Note that when you use session variables in an expression for Oracle BI Presentation Services, you must preface their names with `NQ_SESSION`.

   See "Creating Session Variables" and "Creating Initialization Blocks" for detailed information about setting up session variables and initialization blocks.

2. Edit any logical columns that display currency values to use the appropriate conversion factor using the `PREFERRED_CURRENCY` session variable. To do this, double-click the appropriate logical column in the Business Model and Mapping layer, select the Column Source tab, and create a derived expression that uses the `PREFERRED_CURRENCY` variable.

   For example, the following logical column expression uses the value of the `NQ_SESSION.PREFERRED_CURRENCY` variable to switch between different currency columns. Note that the currency columns are expected to have the appropriate converted values.

   ```
   INDEXCOL( CASE VALUEOF(NQ_SESSION.PREFERRED_CURRENCY) WHEN 'gc1' THEN 0
   WHEN 'gc2' THEN 1 WHEN 'orgc' THEN 2 WHEN 'lc1' THEN 3 ELSE 4 END,
   "Paint"."Sales Facts"."USDCurrency",
   "Paint"."Sales Facts"."DEMCurrency" ,
   "Paint"."Sales Facts"."EuroCurrency" ,
   "Paint"."Sales Facts"."JapCurrency" ,
   "Paint"."Sales Facts"."USDCurrency" )
   ```

3. If you want to provide a dynamic list of currency options, create a table in your data source that provides the entries you want to display for the user-preferred currency. This table must include the following columns:

   - The first column contains the values used to set the session variable `PREFERRED_CURRENCY`. Each value in this column is a string that uniquely identifies the currency (for example, gc2).
   - The second column contains currency tags from the file currencies.xml. The displayMessage values for each tag are used to populate the Currency box and currency prompts (for example, int:euro-1). The currencies.xml file is located in *ORACLE_HOME*\bifoundation\web\display.
   - You can optionally provide a third column that contains the values used to set the presentation variable `currency.userPreference`. Each value in this column is a string that identifies the currency (for example, Global Currency 2). If you omit this column, then the values for the displayMessage attributes for the corresponding currency tags in the currencies.xml file are used.

   Table 9–2 shows a sample table with user-preferred currency entries.

Table 9–2 Sample Table for Dynamically Displaying the Preferred Currency

UserPreference	CurrencyTag	UserPreferenceName
char	char	char
orgc1	loc:en-BZ	Org currency
gc2	int:euro-1	Global currency 2
lc1	int:DEM	Ledger currency
gc1	int:USD	Global Currency 1

Additional configuration is required in Oracle BI Presentation Services to enable this feature. For full information about the Oracle BI Presentation Services configuration, see "Defining User-Preferred Currency Options" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

Setting Default Levels of Aggregation for Measure Columns

You need to specify aggregation rules for mapped logical columns that are measures. Aggregation should only be performed on measure columns, with the possible exception of the aggregation `COUNT` and `COUNTDISTINCT`. Measure columns should exist only in logical fact tables.

You can optionally select different aggregation rules for different dimensions that are associated with this logical column. For example, if someone queries the aggregate column along with one dimension, you may want to use one type of aggregation rule, whereas with another dimension, you may want to use a different aggregation rule.

When the default aggregation rule is Count Distinct, you can optionally specify an override aggregation expression for specific logical table sources. For example, you may want to specify override aggregation expressions when you are querying different aggregate table sources that already contain some level of aggregation. If you do not specify any override, then the default rule prevails.

You can choose the aggregation rule **Evaluate_Aggr** to enable queries to call custom functions in the data source. For information about this function and other aggregation rules, see Appendix C. See also "Defining Aggregation Rules for Multidimensional Data Sources" for additional information about setting aggregation for multidimensional sources.

By default, data is considered sparse. However, on rare occasions you might have a logical table source with dense data. A logical table source is considered to have dense data if it has a row for every combination of its associated dimension levels. When setting up aggregate rules for a measure column, you can specify that data is dense only if *all* the logical table sources to which it is mapped are dense.

Note: The default aggregation rule set for a column in the Oracle BI repository can be overridden in Answers. See "Aggregation Rules and Functions" in *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

To specify a default aggregation rule for a measure column:

1. In the Business Model and Mapping layer, double-click a logical column.
2. In the Logical Column dialog, click the Aggregation tab.

3. In the Aggregation tab, choose one of the following options:

   - For measures in which the additivity is the same in all dimensions (in other words, for fully-additive or non-additive measures), select one of the aggregate functions from the **Default Aggregation Rule** list.

     The function you select is always applied when a user or an application requests the column in a query, unless an override aggregation expression has been specified.

     When you select Count Distinct as the default aggregation rule, you can specify an override aggregation expression for specific logical table sources. Choose this option when you have more than one logical table source mapped to a logical column and you want to apply a different aggregation rule to each source.

     Click the **Add** button to select logical table sources for which you want to specify individual aggregation rules. In the Browse dialog, select the logical table source you want to add, and click **OK**. Then, in the **Formula** list for that logical table source, select the aggregation rule you want to use.

   - Select **Based on dimensions** if your measure has different additivity for different dimensions (in other words, for semi-additive measures). For example, select this option for inventory units that sum in all dimensions except time. See "Setting Up Dimension-Specific Aggregate Rules for Logical Columns" for more information about this feature.

     Click the **Add** button to select additional dimensions for which you want to specify aggregation rules. In the Browse dialog, select the dimension you want to add, and then click **OK**. Then, in the **Formula** list for that dimension, select the aggregation rule you want to use, or click the **Expression Builder** button to build the aggregation rule using Expression Builder.

     The **Data is dense** option appears when you select **Based on dimensions**. Select this option only if *all* the logical table sources to which this column is mapped are dense.

     Caution: Selecting **Data is dense** indicates that all sources to which this column is mapped have a row for every combination of dimension levels that they represent. Selecting this option when any table source that is used by this column does not contain dense data will return incorrect results.

     ---

4. Click **OK**.

Setting Up Dimension-Specific Aggregate Rules for Logical Columns

The majority of measures have the same aggregation rule for each dimension. However, some measures can have different aggregation rules for different dimensions.

For example, bank balances might be averaged over time but summed over the individual accounts. The Oracle BI Server lets you configure dimension-specific aggregation rules. You can specify one aggregation rule for a given dimension and specify other rules to apply to other dimensions.

You need to configure dimensions in the Business Model and Mapping layer to set up dimension-specific aggregation. For more information about setting up aggregate navigation, see Chapter 11.

To specify dimension-specific aggregation rules for a single logical column:

1. In the Business Model and Mapping layer, double-click a logical column.
2. In the Logical Column dialog, click the Aggregation tab.
3. In the Aggregation tab, select **Based on dimensions**.
4. In the Browse dialog, select a dimension over which you want to aggregate, and then click **OK**.
5. In the Aggregation tab, from the **Formula** list, select a rule.

   After selecting rules for specified dimensions, set the aggregation rule for any remaining dimensions by using the dimension labeled Other.
6. If you need to create more complex formulas, click the **Expression Builder** button to the right of the **Formula** column to open Expression Builder.
7. If you have multiple dimensions, you can click **Up** or **Down** to change the order in which the dimension-specific rules are performed.

   When calculating the measure, aggregation rules are applied in the order (top to bottom) established in the dialog.
8. Click **OK**.

To specify dimension-specific aggregation rules for multiple logical fact columns:

1. In the Business Model and Mapping layer, select multiple logical fact columns.
2. Right-click and select **Set Aggregation**.

   You must select more than one column to see the **Set Aggregation** menu item. Also note that **Set Aggregation** does not appear if one or more of the columns you select is a derived column.
3. In the Aggregation dialog, select or clear **All columns the same**.

   This option is selected by default. When selected, you can set aggregation rules that apply to all selected columns. If you clear this option, you can set aggregation rules separately for each selected column.
4. In the Aggregation tab, select **Based on dimensions**.
5. In the Browse dialog, select a dimension over which you want to perform aggregation, and then click **OK**.

   After setting up the rule for a dimension, specify aggregation rules for any other dimensions in the entry labeled Other.
6. Click the **Expression Builder** button to the right of the **Formula** column.
7. In the Expression Builder - Aggregate dialog, from the **Formula** list, select the aggregation to perform over the dimension.
8. To change the order in which the dimension-specific rules are performed, click **Up** or **Down**, and then click **OK**.

   When calculating the measure, aggregation rules are applied in the order (top to bottom) established in the dialog.

Defining Aggregation Rules for Multidimensional Data Sources

By default, when you import Essbase and some other multidimensional cubes into the Physical layer, Oracle Business Intelligence cannot read the aggregation rules set

within the data source. Because of this, the measures are imported automatically with the default aggregation rule of **External Aggregation**.

This section describes best practices for defining aggregation rules for logical measures sourced from Essbase, Oracle OLAP, and other multidimensional data sources, like Microsoft Analysis Services and SAP/BW.

External Aggregation means that the Oracle BI Server is not aware of the underlying aggregation rule for the specific measure and will not compute it internally. Instead, the Oracle BI Server will always ship the query to the underlying multidimensional data source for aggregation.

Because the underlying data sources are extremely efficient, pushing the aggregation rules down to the data source ensures that the Oracle BI Server returns the results without adding any additional overhead in processing. However, it is recommended that you update the aggregation rule for each measure in Oracle Business Intelligence, both in the Physical layer and Business Model and Mapping layer, with the corresponding aggregation rule defined in the data source. Doing so ensures that the Oracle BI Server can do additional computations when needed. There is no query performance impact, since the Oracle BI Server still pushes down optimized queries wherever possible.

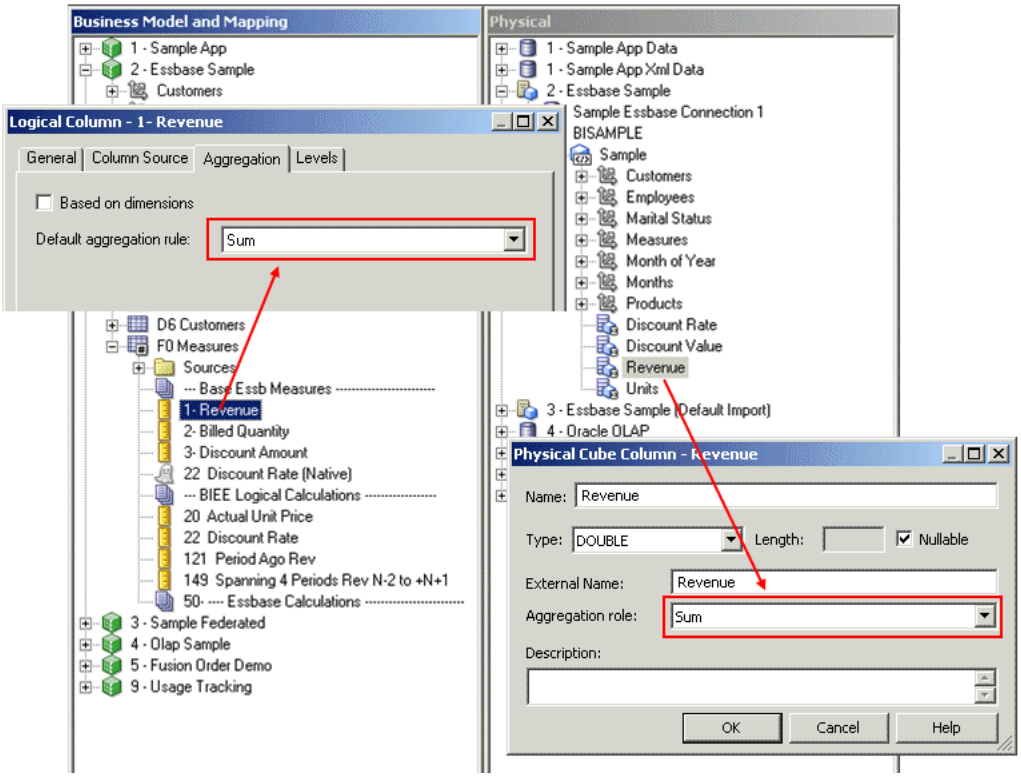
Note: If the Oracle BI Server needs to do additional aggregation for a particular query, and the aggregation rule is set to the default of External Aggregation, the server returns the following error:

An external aggregate is found in an outer query block.

This error occurs because the Oracle BI Server cannot read the aggregation rule in the underlying data source. To ensure that correct results are returned for these queries, you should change the aggregation rules set in the Oracle BI repository to match the aggregation rules set in the underlying data source.

You must ensure that the aggregation rule defined in Oracle Business Intelligence matches the rule in the underlying data source. Also, you must set the appropriate aggregation rule in both the Physical layer and Business Model and Mapping layer, as shown in Figure 9–3.

Figure 9–3 Setting Aggregation Rules in the Physical and Business Model and Mapping Layers for Multidimensional Sources



For custom aggregations or aggregations which do not have a corresponding function within the Oracle BI Server, it is recommended to leave the aggregation as External Aggregation for both the physical measure column and its corresponding logical measure column.

Note: For Oracle OLAP data sources, you do not explicitly set Physical layer aggregation rules for Oracle OLAP columns. Because of this, you only need to set the aggregation rule for Oracle OLAP columns in the Business Model and Mapping layer.

In addition, if a query requests an aggregate that does not exist in the Oracle OLAP data source, and the aggregation rule is set to External Aggregation, then the Oracle BI Server returns an error. To avoid this error, make sure to explicitly set the aggregation rule for the Oracle OLAP column in the Business Model and Mapping layer.

If you do not explicitly set the aggregation rule for Oracle OLAP columns to something other than External Aggregation, requests from Oracle BI Presentation Services custom groups will fail, because custom groups always request aggregates that do not exist in the data source.

Associating an Attribute with a Logical Level in Dimension Tables

Attributes can be associated with a logical level by selecting the dimensional level on the Levels tab. Measures can be associated with levels from multiple dimensions and always aggregate to the levels specified. When a measure is associated to a level it is called a level based measure and it is computed at that grain, even when the query

context has a lower grain. For example, if yearlySales is associated to year level, it will be computed at the yearly level in the following query: Select month, yearlySales.

Dimensions appear in the Dimensions list. If this attribute is associated with a logical level, the level appears in the Levels list.

Another way to associate a measure with a level in a dimension is to expand the dimension tree in the Business Model and Mapping layer, and then use drag-and-drop to drop the column on the target level. For more information about level-based measures, see Example 10–1.

To associate a measure with a logical level in a dimension:

1. In the Business Model and Mapping layer of the Administration Tool, double-click a logical column.
2. In the Logical Column dialog, click the Levels tab.
3. In the Levels tab, click the **Logical Level** field for the dimension from which you want to select a logical level.

   In the Levels tab, in the levels list, you can sort the rows (toggle between ascending order and descending order) by clicking a column heading.
4. In the **Logical Level** list, select the level.
5. Repeat this process to associate this measure with other logical levels in other dimensions.

To remove the association between a dimension and a measure:

1. In the Business Model and Mapping layer of the Administration Tool, double-click a logical column.
2. In the Logical Column dialog, click the Levels tab.
3. In the Levels tab, select the row for the association you want to remove and click **Delete**.
4. Click **OK**.

Moving or Copying Logical Columns

By default, dragging and dropping a logical column from one table to another moves the logical column. If a column with the same name already exists, the new column is renamed (for example, mycolumn#1).

You can also choose the option **Prompt when moving logical columns** in the Options dialog to cause the Sources for moved columns dialog to be displayed when you drag and drop a logical column. This dialog gives you options about the drag and drop behavior.

See "Setting Administration Tool Options" for more information about selecting the **Prompt when moving logical columns** option.

To move or copy logical columns using the Sources for moved columns dialog:

1. In the Business Model and Mapping layer, drag and drop a logical column to a different logical table. You can select multiple columns to move.
2. In the Sources for moved columns dialog, in the **Action** area, select an action.
3. If you select **Ignore**, no logical source is added in the Sources folder of the destination table.

4. If you select **Create new**, a copy of the logical source associated with the logical column is created in the Sources folder of the destination table.
5. If you select **Use existing**, in the **Use existing** list, you must select a logical source from the Sources folder of the destination table.

   The column that you moved or copied is associated with this logical source.

Enabling Write Back On Columns

You can configure individual logical columns so that users in Oracle BI Presentation Services can update column data and write the changes back to the data source.

To enable write back on a particular column, you must select the **Writeable** option for the logical column, and enable the **Read/Write** permission for the corresponding presentation column. You must also disable caching on the corresponding physical table.

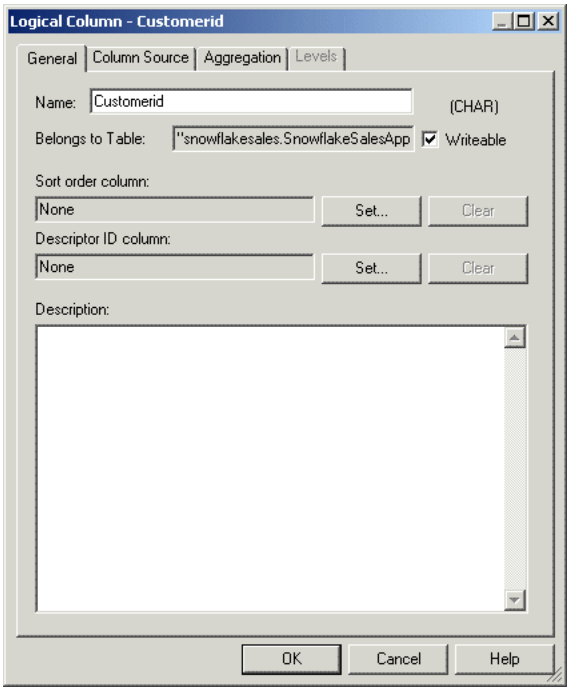
Additional tasks to enable write back need to be performed in Oracle BI Presentation Services. See "Configuring for Write Back in Analyses and Dashboards" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for full information.

To enable write back for a particular column:

1. In the Administration Tool, in the Physical layer, double-click the physical table that contains the column for which you want to enable write back.
2. On the General tab of the Physical Table dialog, ensure that **Cacheable** is not selected. Deselecting this option ensures that Oracle BI Presentation Services users can see updates immediately.
3. In the Business Model and Mapping layer, double-click the corresponding logical column. The Logical Column dialog opens.

   Figure 9–4 shows the Logical Column dialog.

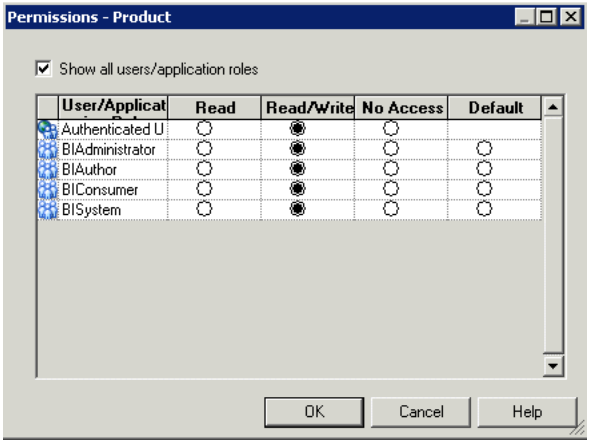
Figure 9–4 Logical Column Dialog with Writeable Option Selected



4. Select **Writeable**, then click **OK**.
5. In the Presentation layer, double-click the column that corresponds to the logical column for which you enabled write back. The Presentation Column dialog opens.
6. Click **Permissions**.
7. Select the **Read/Write** permission for the appropriate users and application roles.

   Figure 9–5 shows the Permissions dialog.

Figure 9–5 Permissions Dialog for Presentation Layer Column with Read/Write Option Selected



8. Click **OK** in the Permissions dialog.
9. Click **OK** in the Presentation Column dialog.

Setting Up Display Folders in the Business Model and Mapping Layer

You can create display folders to organize objects in the Business Model and Mapping layer. Display folders have no effect on query processing.

After you create a display folder, the selected tables and dimensions appear in the folder as a shortcut and in the business model tree as the object. You can hide the objects so that you only view the shortcuts in the display folder. See the information about the Repository tab of the Options dialog in "Setting Administration Tool Options" for more information about hiding these objects.

Note: Deleting a table in a display folder deletes only the shortcut to that object. When you delete a column in a display folder, however, the column is actually deleted.

To set up a logical display folder:

1. In the Business Model and Mapping layer of the Administration Tool, right-click a business model and select **New Object**, then select **Logical Display Folder**.
2. In the Logical Display Folder dialog, in the Tables tab, type a name for the folder.
3. To add tables to the display folder, click **Add**. In the Browse dialog, select the fact or dimension tables you want to add to the folder and click **Select**.

   Alternatively, you can drag one or more logical tables to the display folder after you close the dialog.
4. To add dimensions to the display folder, click the Dimensions tab and click **Add**. In the Browse dialog, select the dimensions that you want to add to the folder and click **Select**.

   Alternatively, you can drag one or more dimensions to the display folder after you close the dialog.
5. Click **OK**.

Modeling Bridge Tables

A bridge table enables you to resolve many-to-many relationships between tables.

For example, you might hold information about employees in an Employees table, and information about the jobs they do in a Jobs table. However, an organization's employees can have multiple jobs, and the same job can be performed by multiple employees. This situation would result in a many-to-many relationship between the Employees table and the Jobs table.

To resolve the many-to-many relationship, you can create a bridge table (or intermediate table) called Assignments. Each row in the Assignments table is unique, representing one employee doing one job. If an employee has several jobs, there are several rows in the Assignments table for that employee. If a job is done by several employees, there are several rows in the Assignments table for that job. The primary key of the Assignments table is a composite key, made up of a column containing the employee ID and a column containing the job ID.

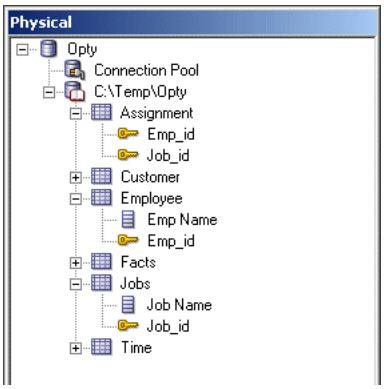
By acting as a bridge table between the Job and Employee tables, the Assignments table enables you to resolve the many-to-many relationship between Employees and Jobs into:

- A one-to-many relationship between Employees and Assignments

- A one-to-many relationship between Assignments and Jobs

Figure 9–6 shows a Physical layer view of the example bridge and associated dimension tables described in the preceding paragraphs.

Figure 9–6 Example Bridge and Associated Tables in the Physical Layer



Note that "Weight Factor" should be included as an additional column in the bridge table and calculated during ETL for efficient query processing.

The following sections explain how to model bridge tables in the Physical and Business Model and Mapping layers:

- Creating Joins in the Physical Layer for Bridge and Associated Dimension Tables
- Modeling the Associated Dimension Tables in a Single Dimension
- Modeling the Associated Dimension Tables in Separate Dimensions

Creating Joins in the Physical Layer for Bridge and Associated Dimension Tables

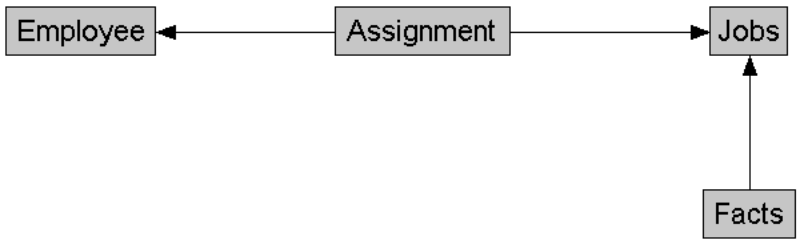
To model bridge tables in the Physical layer, create joins between the bridge table and the associated dimension tables.

To create physical joins for a bridge table and its associated tables:

1. In the Administration Tool, in the Physical layer, select the fact, bridge, and associated dimension tables. Then, right-click the objects and select **Physical Diagram**, and then choose **Selected Object(s) Only**.
2. With the Physical Diagram displayed, click **New Join** on the toolbar. Then, select the bridge table, and then select one of the dimension tables.
3. Click **OK** in the Physical Foreign Key dialog.
4. Repeat steps **2** and **3** for the other associated dimension table.
5. Ensure that one of the associated dimension tables is joined to the fact table.

   Figure 9–7 shows joins between the example Physical tables in the Physical Diagram.

Figure 9–7 Joins Between the Example Tables in the Physical Diagram



Modeling the Associated Dimension Tables in a Single Dimension

In the Business Model and Mapping layer, you can choose to model the two dimension tables associated with a bridge table in a single dimension, or in two separate dimensions.

To model the associated dimension tables in one dimension, create a second logical table source that maps to the bridge table and the other dimension table, and then add columns from the other dimension table.

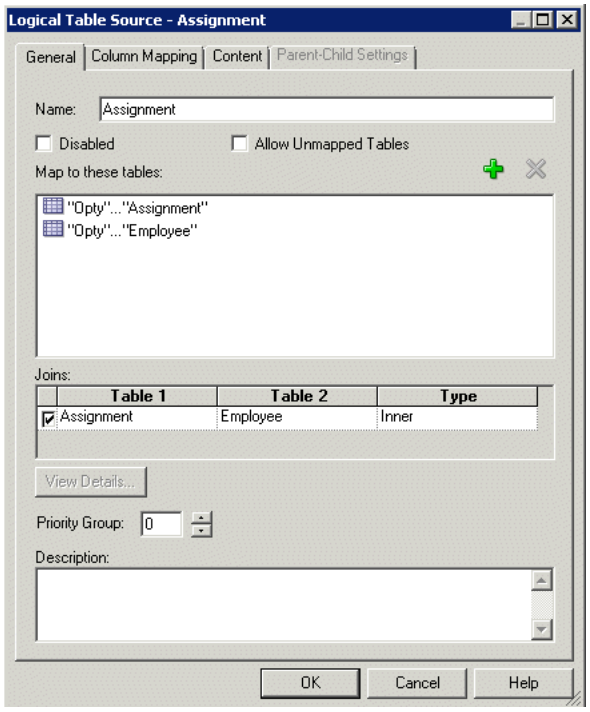
Providing two separate logical table sources makes queries more efficient, because it ensures that queries against a single dimension table do not involve the bridge table.

To model the dimension tables associated with a bridge table in a single dimension:

1. Drag objects from the Physical layer to the Business Model and Mapping layer, *except* the bridge table and the associated dimension table that is not joined to the fact table. For the example described in the previous sections, you would drag all objects except for the Assignment and Employee tables.
2. In the Business Model and Mapping layer, right-click the dimension table that is joined to the fact table (Jobs in our example) and select **New Object**, then select **Logical Table Source**.
3. In the Logical Table Source dialog, provide a name for the new bridge table source. It is a good practice to use the bridge table name as the name of the source (for example, Assignment).
4. Click the **Add** button in the upper right corner of the Logical Table Source dialog. Then, select the bridge table from the **Name** list (**Assignment** in our example) and then click **Select**.
5. Click the **Add** button again and select the associated dimension table that is not joined to the fact table (**Employee** in our example) and then click **Select**.
6. Click **OK** in the Logical Table Source diaog.

   Figure 9–8 shows the Logical Table Source dialog for the bridge table source.

Figure 9–8 Logical Table Source Dialog for Bridge Table Source



7. Drag columns from the dimension table that is not joined to the fact table (Employees in our example) from the Physical layer to the logical table source that you just created.

You can now create dimensions based on your logical tables, including the logical table with the bridge table source.

Modeling the Associated Dimension Tables in Separate Dimensions

As an alternative to modeling the two dimension tables associated with a bridge table in a single dimension, you can choose to model them in separate dimensions.

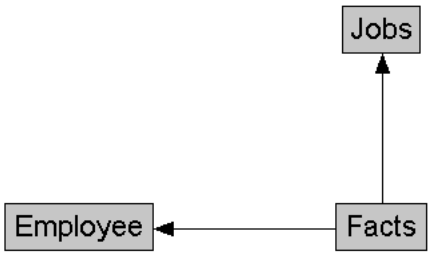
To do this, create a logical join between the fact table and the dimension table that is not physically joined to the fact table, and then modify the logical table source for that same dimension table to add the other table mappings.

To model the dimension tables associated with a bridge table in separate dimensions:

1. Drag objects from the Physical layer to the Business Model and Mapping layer. Because you want to model the dimension tables in separate dimensions, drag both of the dimension tables associated with the bridge table. You do not need to drag and drop the bridge table object.
2. In the Business Model and Mapping layer, select the fact table and the two dimension tables that are associated with the bridge table (Facts, Employee, and Jobs in our example). Then, right-click the objects and select **Business Model Diagram**, and then choose **Selected Tables Only**.
3. With the Business Model Diagram displayed, click **New Join** on the toolbar. Then, select the fact table, and then select the dimension table not currently joined to the fact table.
4. Click **OK** in the Logical Join dialog.

Figure 9–9 shows joins between the example logical tables in the Business Model Diagram.

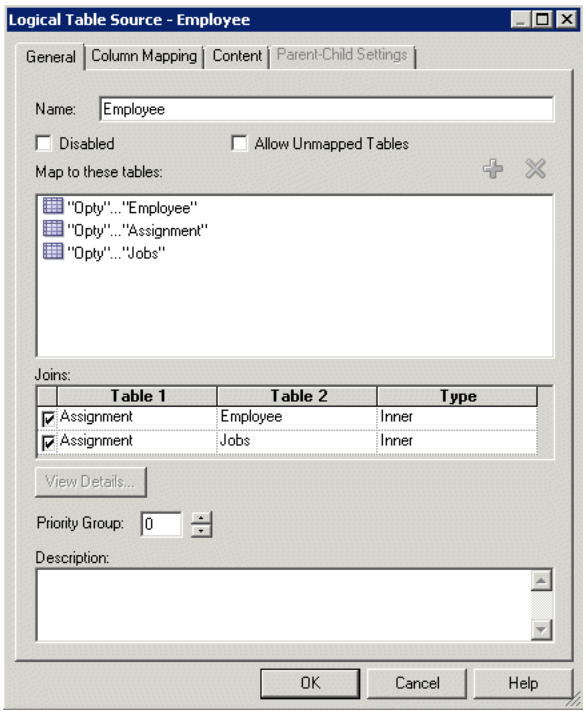
Figure 9–9 Joins Between the Example Tables in the Business Model Diagram



5. Double-click the logical table source for the logical table for which you created the logical join (Employee in our example).
6. Click the **Add** button in the upper right corner of the Logical Table Source dialog. Then, select the bridge table from the **Name** list (**Assignment** in our example) and then click **Select**.
7. Click the **Add** button again and select the other associated dimension table (**Jobs** in our example) and then click **Select**.
8. Click **OK** in the Logical Table Source diaog.

   Figure 9–10 shows the Logical Table Source dialog for the modified dimension table source.

Figure 9–10 Logical Table Source Dialog for Dimension Table Source



You can now create dimensions based on your logical tables, including both logical tables associated with the bridge table.

Modeling Binary Large Object (BLOB) Data and Character Large Object (CLOB) Data

This section describes how to model binary large object (BLOB) data and character large object (CLOB) data in the Oracle BI repository.

CLOB data is a large plain text document in any character set. The supported BLOB image types are: GIF, PNG, TIFF, JPEG, and BMP. BLOB formats not supported are: PDF, audio, or video.

To model BLOB and CLOB data:

1. Import the physical table containing the BLOB or CLOB data from the data source using the Import Metadata Wizard. The default data type for BLOB columns after the import is LongVarBinary, while for CLOB columns it is LongVarChar. After import, open the Physical Column dialog for the BLOB or CLOB column and change the **Length** field to be the desired length, but ensure that it does not exceed the current Oracle BI Server MaxFieldSize limit of 32 KB. Note that this 32 KB limit is also a limitation of the Microsoft Internet Explorer browser.
2. Configure physical joins as necessary. For example, assume there is one table called employees, and another table called Employee_Images that has two columns, employeeid and employee_image (the BLOB column), where employeeid is the primary key. In this situation, you would create a physical join between the employees and Employee_Images tables.
3. Drag the BLOB or CLOB column to the Business Model and Mapping layer. For example, in the situation described in the previous step, you would drag the employee_image physical column to the employee logical table source. Doing this generates a logical column Employee_Image.
4. To ensure that the Oracle BI Server does not generate a group by or order by on the resulting logical column, configure a physical lookup for the logical column. For example:

   ```
   lookup(DENSE "MYDB"."MySchema"."EMPLOYEE_IMAGES"."EMPLOYEE_IMAGE",
   "MYDB"."MySchema"."Employees"."EmployeeID")
   ```

5. In the General tab of the Logical Column dialog for the logical BLOB or CLOB column, configure the **Descriptor ID column** as needed. For our example column Employee_Image, you would configure the descriptor ID column to use EmployeeID. Doing so ensures that Presentation Services uses EmployeeID instead of Employee_Image when generating filters.
6. Still in the General tab of the Logical Column dialog for the logical BLOB or CLOB column, configure the **Sort order column** as needed. For our example column Employee_Image, configure the sort order column to use employeeID. Doing so ensures that the Oracle BI Server orders by EmployeeID when the order by clause contains Employee_Image.
7. Save the changes.

10

Working with Logical Dimensions

This chapter explains how to work with logical dimension objects in the Business Model and Mapping layer of the Oracle BI repository.

This chapter contains the following topics:

About Working with Logical Dimensions

In the Business Model and Mapping layer, a dimension object represents a hierarchical organization of logical columns (attributes). One or more logical dimension tables can be associated with at most one dimension object.

Common dimensions might be time periods, products, markets, customers, suppliers, promotion conditions, raw materials, manufacturing plants, transportation methods, media types, and time of day. Note that dimensions exist in the Business Model and Mapping (logical) layer and in the Presentation layer.

In each dimension, you organize logical columns into the structure of the hierarchy. This structure represents the organization rules and reporting needs required by your business. It also provide the metadata that the Oracle BI Server uses to drill into and across dimensions to get more detailed views of the data.

There are two types of logical dimensions: dimensions with level-based hierarchies (structure hierarchies), and dimensions with parent-child hierarchies (value hierarchies). Level-based hierarchies are those in which members are of several types, and members of the same type occur only at a single level. In parent-child hierarchies, members all have the same type. Oracle Business Intelligence also supports a special type of level-based dimension, called a time dimension, that provides special functionality for modeling time series data.

Because dimensions for multidimensional data sources are defined in the source, they do not require as much work compared with dimensions in other data sources. For example, you do not create dimension level keys. A dimension is specific to a particular multidimensional data source (it cannot be used by more than one) and cannot be created and manipulated individually. Additionally, each cube in the data source should have at least one dimension and one measure in the Business Model and Mapping layer.

You can expose logical dimensions to Oracle BI Answers users by creating presentation hierarchy objects that are based on particular logical dimensions. Creating hierarchies in the Presentation layer enables users to create hierarchy-based queries. See "Working with Presentation Hierarchies and Levels" for more information.

Note that you can also expose dimension hierarchies by adding one or more columns from each hierarchy level to a subject area in the Presentation layer. Oracle BI Answers supports drill-down on these hierarchical columns.

Creating and Managing Dimensions with Level-Based Hierarchies

Each business model can have one or more dimensions, each dimension can have one or more logical levels, and each logical level has one or more attributes (columns) associated with it.

The following sections explain how to create dimensions:

- About Level-Based Hierarchies
- Manually Creating Dimensions, Levels, and Keys with Level-Based Hierarchies
- Automatically Creating Dimensions with Level-Based Hierarchies
- Populating Logical Level Counts Automatically

About Level-Based Hierarchies

A dimension contains two or more logical levels. The recommended sequence for creating logical levels is to create a Grand Total level and then create child levels, working down to the lowest level.

The following are the parts of a dimension:

- **Grand Total level.** A special level representing the grand total for a dimension. Each dimension can have just one Grand Total level. A Grand Total level does not contain dimensional attributes and does not have a level key. However, you can associate measures with a Grand Total level. The aggregation level for those measures will always be the grand total for the dimension.
- **Level.** All levels, except the Grand Total level, need to have at least one column. However, it is not necessary to explicitly associate all of the columns from a table with logical levels. Any column that you do not associate with a logical level is automatically associated with the lowest level in the dimension that corresponds to that dimension table. All logical columns in the same dimension table have to be associated with the same dimension.

  There is no limit to the number of levels you can have in a dimension. The total number of levels is not by itself a determining factor in query performance. However, be aware that for extremely complex SQL queries, even a few levels in a dimension can impact performance.
- **Hierarchy.** Each dimension contains one or more hierarchies. All hierarchies must have a common leaf level.

  For example, a time dimension might contain a fiscal hierarchy and a calendar hierarchy, with a common leaf level of Day. Day has two named parent levels called Fiscal Year and Calendar Year, which are both children of the All root level.

  In the Business Model and Mapping layer, logical hierarchies are not defined as independent metadata objects, unlike hierarchies in the Presentation layer. Rather, logical hierarchies exist implicitly through the relationships between levels.

Tip: It is a good idea to define intermediate levels in your hierarchies to avoid having very large numbers of members at one level. For example, if you are creating a Product dimension for an automotive company that tracks data on 500 different car models, you might want to create some finer-grained hierarchical levels (such as Minivans, Subcompacts, and Midsize Sedans) rather than having all 500 car models directly under Automobiles. Following this practice can improve query performance and can make reports and diagrams easier to read and navigate.

- **Level keys.** Each logical level (except the topmost level defined as a Grand Total level) must have one or more attributes that compose a level key. The level key defines the unique elements in each logical level. The dimension table logical key has to be associated with the lowest level of a dimension and has to be the level key for that level.

  A logical level can have multiple level keys. When that is the case, specify the key that is the primary key of that level. All dimension sources which have an aggregate content at a specified level need to contain the column that is the primary key of that level. Each logical level should have one level key that is displayed when an Oracle BI Presentation Services user clicks to drill down. This may or may not be the primary key of the level. To set the level key to display, select the **Use for display** option in the Level Key dialog.

  Be careful using level keys such as Month whose domain includes the values January, February, and so on (or in other words, values that are not unique to a particular month, repeating every year). To define Month as a level key, you also need to include an attribute from a higher level (for example, Year). To add Year, click **Add** in this dialog and select the logical column from the dialog that is presented.

  If you do not ensure that your level key is unique by including higher-level attributes, then queries might return unexpected results. For example, when the Oracle BI Server needs to combine result sets from multiple physical queries, some expected rows might be dropped because they are not considered unique according to the level key definition.

  Level keys should be meaningful business keys (like Month_name='2010 July') rather than generated surrogate keys (like time_key='1023793'), because surrogate keys are physical artifacts that only apply to a single instance of a source table. The business name, in contrast, can map to any physical instance for that logical column. For example, month_name might map to a detailed table, an aggregate table from an aggregate star, and a column in a federated spreadsheet. Note that the Physical layer still uses the surrogate keys in the joins, so there is no performance or flexibility penalty for using business keys in the business model.

- **Time dimensions and chronological keys.** You can identify a dimension as a time dimension. At least one level of a time dimension must have a chronological key. The following is a list of some guidelines you should use when setting up and using time dimensions:

  – At least one level of a time dimension must have a chronological key. See "Selecting and Sorting Chronological Keys in a Time Dimension" for more information.

  – All time series measures using the `AGO`, `TODATE`, and `PERIODROLLING` functions must be on time levels. `AGO`, `TODATE`, and `PERIODROLLING` aggregates are created as derived logical columns. See "About Time Series Functions" for more

information.

  - AGO, TODATE, and PERIODROLLING functionality is not supported either on fragmented dimensional logical table sources, or on fact sources fragmented on the same time dimension. Fact sources may be fragmented on other dimensions.See "About Time Series Functions" for more information.

- **Unbalanced (or ragged) hierarchy.** An unbalanced (or ragged) hierarchy is a hierarchy where the leaves (members with no children) do not necessarily have the same depth. For example, a site can choose to have data for the current month at the day level, previous months data at the month level, and the previous 5 years data at the quarter level.

  User applications can use the ISLEAF function to determine whether to allow drilldown from any particular member. See "ISLEAF" for more information.

  A missing member is implemented in the data source with a null value for the member value. All computations treat the null value as a unique child within its parent. Level-based measures and aggregate-by calculations group all missing nodes together.

  Note that unbalanced hierarchies are not necessarily the same as parent-child hierarchies. Parent-child hierarchies are unbalanced by nature, but level-based hierarchies can be unbalanced also.

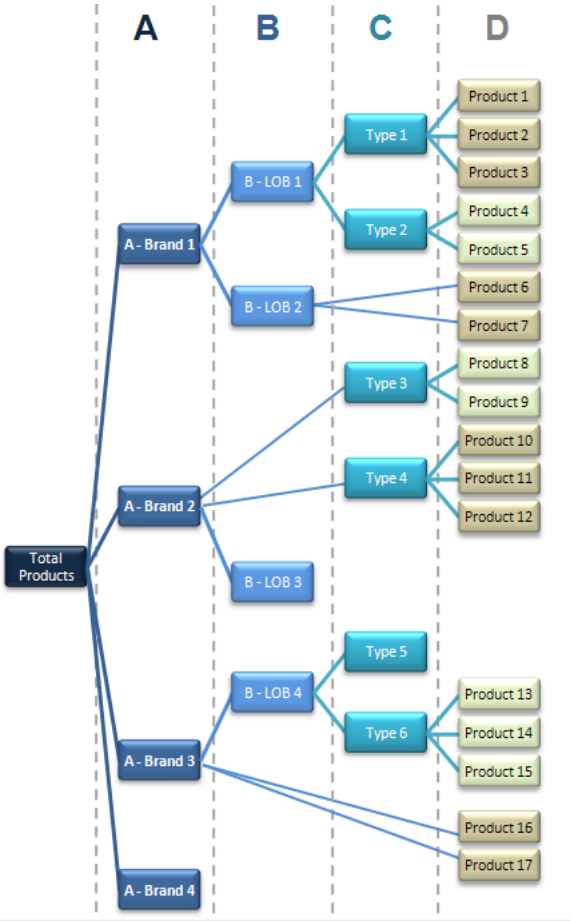
- **Skip-level hierarchy.** A skip-level hierarchy is a hierarchy where there are members that do not have a value for a particular ancestor level. For example, in a Country-State-City-District hierarchy, the city 'Washington, D.C.' does not belong to a State. In this case, you can drill down from the Country level (USA) to the City level (Washington, D.C.) and below.

  In a query, skipped levels are not displayed, and do not affect computations. When sorted hierarchically, members appear under their nearest ancestors.

  A missing member at a particular level is implemented in the data source with a null value for the member value. All computations treat the null value as a unique child within its parent. Level-based measures and aggregate-by calculations group all skip-level nodes together.

Figure 10–1 shows a hierarchy with both unbalanced and skip-level characteristics. For example, A-Brand 4, B-LOB 3, and Type 5 are unbalanced branches, while skips are present between A-Brand 2 and Type 3, B-LOB 2 and Product 6, and others.

Figure 10–1 Hierarchy with Unbalanced and Skip-Level Characteristics



About Using Dimension Hierarchy Levels in Level-Based Hierarchies

Dimension hierarchical levels can be used to perform the following actions:

- Set up aggregate navigation
- Configure level-based measure calculations (refer to Example 10–1).
- Determine what attributes appear when Oracle BI Presentation Services users drill down in their data requests

Manually Creating Dimensions, Levels, and Keys with Level-Based Hierarchies

To create and manage dimension hierarchy levels in level-based hierarchies, perform the tasks described in the following sections:

- Creating Dimensions in Level-Based Hierarchies
- Creating Logical Levels in a Dimension
- Associating a Logical Column and Its Table with a Dimension Level
- Identifying the Primary Key for a Dimension Level
- Selecting and Sorting Chronological Keys in a Time Dimension
- Adding a Dimension Level to the Preferred Drill Path
- Adding Sequence Numbers to a Time Dimension's Logical Level

Creating Dimensions in Level-Based Hierarchies

After creating a dimension, each dimension can be associated with attributes (columns) from one or more logical dimension tables and level-based measures from logical fact tables.

After you associate logical columns with a dimension level, the tables in which these columns exist appear in the Tables tab of the Dimension dialog.

To create a dimension with a level-based hierarchy:

1. In the Business Model and Mapping layer of the Administration Tool, right-click a business model and select **New Object > Logical Dimension > Dimension with Level-Based Hierarchy**.

   Note that this option is only available when there is at least one dimension table that has no dimension associated with it.

2. In the Logical Dimension dialog, in the General tab, type a name for the dimension.

   The **Default root level** field is automatically populated after you associate logical columns with a dimension level.

3. If the dimension is a time dimension, select **Time**.

4. If the dimension is an unbalanced dimension, select **Ragged**.

5. If the dimension is a skip-level dimension, select **Skipped Levels**.

   Note: It is a best practice to ensure that the physical hierarchy type set in the Physical layer matches the dimension properties you select in the Business Model and Mapping layer. See "Working with Physical Hierarchy Objects" for more information.

   In addition, you must ensure that the Ragged and Skipped Levels dimension properties are set correctly for queries to work.

   ---

6. (Optional) Type a description of the dimension.

7. Click **OK**.

Creating Logical Levels in a Dimension

When creating logical levels in a dimension, you also create the hierarchy by identifying the type of level and defining child levels.

See "Automatically Creating Business Model Objects for Multidimensional Data Sources" for more information about creating hierarchies for a multidimensional data source.

To define general properties for a logical level in a dimension:

1. In the Business Model and Mapping layer of the Administration Tool, right-click a dimension and select **New Object**, then select **Logical Level**.

2. In the Logical Level dialog, in the General tab, specify a name for the logical level.

3. For **Number of elements at this level**, specify the number of elements that exist at this logical level. If this level will be the Grand Total level, leave this field blank. The system will set to a value of 1 by default.

   The number does not have to be exact, but ratios of numbers from one logical level to another should be accurate. For relational sources, you can retrieve the row

count for the level key and use that number as the number of elements. For multidimensional sources, you can use the number of members at that level.

The Oracle BI Server uses this number when selecting which aggregate source to use. For example, when aggregate navigation is used, multiple fact sources exist at different grains. The Oracle BI Server multiplies the number of elements at each level for each qualified source as a way to estimate the total number of rows for that source. Then, the Oracle BI Server compares the result for each source and selects the source with the lowest number of total elements to answer the query. The source with the lowest number of total elements is assumed to be the fastest.

4. Choose one of the following options, if appropriate:
   - If the logical level is the Grand Total level, select **Grand total level**. There should be only one Grand Total level for a dimension.
   - If measure values at a particular level fully constitute aggregated measures at its parent level, select **Supports rollup to higher level of aggregation**.
5. To define child logical levels, click **Add**.
6. In the Browse dialog, select the child logical levels and click **OK**.

   The child levels appear in the Child Levels pane.
7. To remove a previously defined child level, select the level in the Child Levels pane and click **Remove**.

   The child level and all of its child levels are deleted from the Child Levels pane.
8. (Optional) Type a description of the logical level.
9. Click **OK**.

Associating a Logical Column and Its Table with a Dimension Level

After you create all logical levels within a dimension, you need to drag and drop one or more columns from the dimension table to each logical level except the Grand Total level.

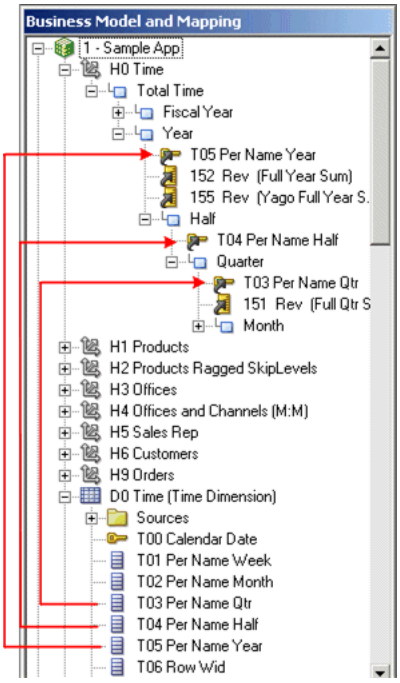
The first time you drag a column to a dimension it associates the logical table to the dimension. It also associates the logical column with that level of the dimension. To change the logical level to be associated with that logical column, you can drag a column from one logical level to another.

The logical column or columns that comprise the logical key of a dimension table must be associated with the lowest level of the dimension.

After you associate a logical column with a dimension level, the tables in which these columns exist appear in the Tables tab of the Dimensions dialog.

For time dimensions, ensure that all time-related logical columns in the source table are defined in the time dimension. For example, if a time-related logical table contains the columns Month Name and Month Code, you must ensure that both columns are dragged to the appropriate level within the dimension. Figure 10–2 shows how to associate logical columns with a logical level.

Figure 10–2 Associating Logical Columns with a Logical Level



To verify tables that are associated with a dimension:

1. In the Business Model and Mapping layer of the Administration Tool, double-click a dimension.
2. In the Dimensions dialog, click the Tables tab.

   The tables list contains tables that you associated with that dimension. This list of tables includes only one logical dimension table and one or more logical fact tables (if you created level-based measures).
3. Click **OK** or **Cancel** to close the Dimensions dialog.

Example 10–1 and Example 10–2 show how to associate measures to different levels of level-based dimension hierarchies.

Example 10–1 Level-Based Measure Calculations

A level-based measure is a column whose values are always calculated to a specific level of aggregation. For example, a company might want to measure its revenue based on the country, based on the region, and based on the city. You can set up columns to measure CountryRevenue, RegionRevenue, and CityRevenue.

When a query containing a presentation hierarchy includes a level-based measure column, and the query grain is higher than the level of aggregation specific to the column, the query results return null. Note that if the request only contains ordinary columns and no hierarchical columns, the level-based measure is not replaced with null.

The measure AllProductRevenue is an example of a level-based measure at the Grand Total level. Level-based measures allow a single query to return data at multiple levels of aggregation. They are also useful in creating share measures, that are calculated by taking some measure and dividing it by a level-based measure to calculate a percentage. For example, you can divide salesperson revenue by regional revenue to calculate the share of the regional revenue each salesperson generates.

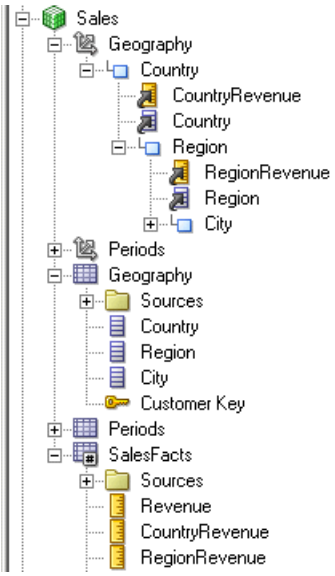
To set up these calculations, you need to build a dimensional hierarchy in your repository that contains the levels Grandtotal, Country, Region, and City. This hierarchy contains the metadata that defines a one-to-many relationship between Country and Region and a one-to-many relationship between Region and City. For each country, there are many regions, but each region is in only one country. Similarly, for each region, there are many cities, but each city is in only one region.

Next, you need to create three logical columns (CountryRevenue, RegionRevenue, and CityRevenue). Each of these columns uses the logical column Revenue as its source. The Revenue column has a default aggregation rule of `SUM` and has sources in the underlying databases.

You then drag the CountryRevenue, RegionRevenue, and CityRevenue columns into the Country, Region, and City levels, respectively. Each query that requests one of these columns returns the revenue aggregated to its associated level.

Figure 10–3 shows what the business model in the Business Model and Mapping layer looks like for this example.

Figure 10–3 Example of Business Model in the Business Model and Mapping Layer



In the Geography Dimension, the CountryRevenue and RegionRevenue columns are attributes of the Country and Region levels. In the Sales Facts table, the Revenue column has a default aggregation rule of `SUM` and is mapped to physical detail data or physical aggregate data. CountryRevenue and RegionRevenue columns use the Revenue column as their source.

Example 10–2 Grand Total Dimension Hierarchy

You might have a product dimensional hierarchy with levels TotalProducts (Grand Total level), Brands, and Products. Additionally, there might be a column called Revenue that is defined with a default aggregation rule of Sum. You can then create a logical column, AllProductRevenue, that uses Revenue as its source (as specified in the General tab of the Logical Column dialog). Now, drag AllProductRevenue to the Grand Total level. Each query that includes this column returns the total revenue for all products. The value is returned regardless of any constraints on Brands or Products. If you have constraints on columns in other tables, the grand total is limited to the scope of the query. For example, if the scope of the query asks for data from 1999 and 2000, the grand total product revenue is for all products sold in 1999 and 2000.

If you have three products, A, B, and C with total revenues of 100, 200, and 300 respectively, then the grand total product revenue is 600 (the sum of each product's revenue). If you have set up a repository as described in this example, the following query produces the results listed:

```
SELECT product, productrevenue, allproductrevenue
FROM sales_subject_area
WHERE product IN 'A' or 'B'
```

The results are as follows:

```
PRODUCT  PRODUCTREVENUE  ALLPRODUCTREVENUE
A        100             600
B        200             600
```

In this example, the AllProductRevenue column always returns a value of 600, regardless of the products on which the query constrains.

Identifying the Primary Key for a Dimension Level

Use the Keys tab in the Logical Level dialog to identify the primary key for a level.

To specify a primary key for a dimension level:

1. In the Business Model and Mapping layer of the Administration Tool, expand a dimension and then expand the highest level (Grand Total level) of the dimension.
2. Double-click a logical level below the Grand Total level.
3. In the Logical Level dialog, click the Keys tab.
4. In the Keys tab, from the **Primary key** list, select a level key.

   If only one level key exists, it is the primary key by default.
5. To add a column to the list, perform the following steps:
   a. In the Logical Level dialog, click **New**.
   b. In the Logical Level Key dialog, type a name for the key.
   c. In the Logical Level Key dialog, select a column or click **Add**.
   d. If you click **Add**, in the Browse dialog, select the column, and then click **OK**.

      The column you selected appears in the **Columns** list of the Logical Level Key dialog and is automatically selected.

      Note: You cannot use a derived logical column that is the result of a `LOOKUP` function as part of a primary logical level key. This limitation exists because the `LOOKUP` operation is applied after aggregates are computed, but level key columns must be available before the aggregates are computed because they define the granularity at which the aggregates are calculated.

      You can use a derived logical column that is the result of a `LOOKUP` function as a secondary logical level key.

      ---
6. If the level is in a time dimension, you can select chronological keys and sort the keys by name.
7. (Optional) Type a description for the key and then click **OK**.
8. Repeat Step 2 through Step 7 to add primary keys to other logical levels.

9. In the Logical Level dialog, click **OK**.

Selecting and Sorting Chronological Keys in a Time Dimension

At least one level of a time dimension must have a chronological key. Although you can select one or more chronological keys for any level and then sort keys in the level, only the first chronological key is used.

To select and sort chronological keys for a time dimension:

1. In the Business Model and Mapping layer of the Administration Tool, expand a time dimension and then expand the highest level (Grand Total level) of the dimension.

   Note: For a dimension to be recognized as a time dimension, you must select **Time** on the General tab of the Dimension dialog.

2. Double-click a logical level below the Grand Total level.
3. In the Logical Level dialog, click the Keys tab.
4. To select a chronological key, in the Keys tab, select the **Chronological Key** option. You may need to scroll to the right to see this option.
5. To sort chronological keys, in the Keys tab, select a chronological key and then click **Edit**.
6. In the Chronological Key dialog, select a chronological key column, click **Up** or **Down** to reorder the column, and then click **OK**.

Adding a Dimension Level to the Preferred Drill Path

You can use the Preferred Drill Path tab to identify the drill path to use when Oracle BI Presentation Services users drill down in their data requests.

You should use this only to specify a drill path that is outside the normal drill path defined by the dimensional level hierarchy. It is most commonly used to drill from one dimension to another. You can delete a logical level from a drill path or reorder a logical level in the drill path.

To add a dimension level to the preferred drill path:

1. Click **Add** to open the Browse dialog, then select the logical levels to include in the drill path. You can select logical levels from the current dimension, or from other dimensions.
2. Click **OK** to return to the Level dialog.

   The names of the levels are added to the Names pane.

Adding Sequence Numbers to a Time Dimension's Logical Level

Adding absolute or relative sequence numbers to time dimensions optimizes time series functions and in some cases improves query time.

By default Oracle BI Server uses a complex RANK Physical SQL expression to generate sequence numbers for time dimensions. Adding an absolute or relative sequence number to the time dimension's logical level provides direct column references in the Time dimension table that contain the precomputed results of the rank expressions. This mapping, while optional, generates a simpler query that is easier for Oracle BI Server to execute against the data source.

Sequence numbers are enumerations of time dimensional members at a certain level. The enumeration must be dense (no gaps) and must correspond to a real time order. For example, months in a year can be enumerated from 1 to 12.

To add sequence numbers:

1. In the Business Model and Mapping layer of the Administration Tool, locate a time dimension and then double click a corresponding logical level.
2. In the Logical Level dialog, in the Sequence Numbers tab, specify the type of sequence numbers to add to the logical level. Consider the following options:
   - **Absolute** – Choose this option to configure an absolute sequence number when the column enumerates the members of the time dimension without any reference. For example, calendar year.
   - **Relative**– Chose this option to configure relative sequence numbers when you have a column that enumerates members of the time dimension relative to some parent level. For example, months in year which can be 1 to 12.
3. Click **OK**.

Automatically Creating Dimensions with Level-Based Hierarchies

You can set up a dimension automatically from a logical dimension table if a dimension for that table does not exist.

To create a dimension automatically, the Administration Tool examines the logical table sources and the column mappings in those sources and uses the joins between physical tables in the logical table sources to determine logical levels and level keys. Therefore, it is best to create a dimension in this way after all the logical table sources have been defined for a dimension table.

The following rules apply:

- Create Dimensions is only available if the selected logical table is a dimension table (defined by 1:N logical joins) and no dimension has been associated with this table.
- An automatically created dimension uses the same name as the logical table, adding Dim as a suffix. For example, if a table is named Periods, the dimension is named Periods Dim.
- A Grand Total level is automatically named *logical_table_name* Total. For example, the Grand Total level of the Periods Dim table is Periods Total.
- When there are multiple tables in a source, the join relationships between tables in the source determine the physical table containing the lowest-level attributes. The lowest level in the hierarchy is named *logical_table_name* Detail. For example, the lowest level of the periods table is Periods Detail.
- The logical key of the dimension table is mapped to the lowest level of the hierarchy and specified as the level key. This logical column should map to the key column of the lowest level table in the dimension source.
  - If there are two or more physical tables in a source, the columns that map to the keys of those tables become additional logical levels. These additional level names use the logical column names of these key columns.
  - The order of joins determines the hierarchical arrangement of the logical levels. The level keys of these new logical levels are set to the logical columns that map to the keys of the tables in the source.

- If there are multiple logical table sources, the tool uses attribute mappings and physical joins to determine the hierarchical order of the tables in the physical sources. For example, you might have three sources (A, B, C) each containing a single physical table and attribute mappings for 10, 15, and 3 attributes, respectively (not counting columns that are constructed from other logical columns). The following is a list of the results of creating a dimension for this table automatically:
  - The Administration Tool creates a dimension containing four logical levels, counting the Grand Total and detail levels.
  - The key of the table in source B (that has the greatest number of columns mapped and contains the column mapping for the logical table key) should be the level key for the detail level.
  - The parent of the detail level should be the logical level named for the logical column that maps to the key of the physical table in source A.
  - Any attributes that are mapped to both A and B should be associated with level A.
  - The parent of level A should be the logical level named for the logical column that maps to the key of the physical table in source C.
  - Any columns that are mapped to both A and C should be associated with level C.
- Table joins in a physical source might represent a pattern that results in a split hierarchy. For example, the Product table can join to the Flavor table and a Subtype table. This would result in two parents of the product detail level, one flavor level and one subtype level.
- You cannot create a dimension automatically in the following situations:
  - If a dimension with joins and levels has already been created, Create Dimension does not appear on the right-click menu.
  - If the table is not yet joined to any other table, the option is not available because it is considered a fact table.
- In a snowflake schema, if you use a table with only one source and create the dimension automatically, the child tables are automatically incorporated into a hierarchy. The child tables form intermediate levels between the Grand Total level and detail level. If more then one child table exists for a dimension table, the hierarchy is a split hierarchy.

To create a dimension automatically:

1. In the Administration Tool, open a repository.
2. In the Business Model and Mapping layer, right-click a logical dimension table that is not associated with any dimension .
3. From the right-click menu, select **Create Logical Dimension**, then select either **Dimension with Level-Based Hierarchy** or **Dimension with Parent-Child Hierarchy**.

   The new dimension is displayed in the Business Model and Mapping layer.

Populating Logical Level Counts Automatically

Estimate Levels enables administrators to automatically populate level counts for one or more dimension hierarchies. Level counts are utilized by the query engine to determine the most optimal query plan and optimizes overall system performance.

The repository must be opened in online mode and the business model must be available for queries. Then, in the Business Model and Mapping layer, you can select any of the following logical layer elements, and then execute the Estimate Levels command:

- **Business model.** Must be available for queries. If you select this object, the Administration Tool attempts to check out all objects in the business model.
- **Dimension.** You should run a consistency check on dimensions to ensure that the dimension is logically sound.
- **A combination of business models and dimensions.** You can select multiple dimensions and multiple business models individually.

When run, the Estimate Levels command also launches a consistency check on the level counts as described in the following list:

- Checks that a level key is valid. Columns in levels have referential integrity.
- Checks the parent-child relationship. If the parent level count is greater that the child level count, an error is returned.
- Generates a run report that lists all the counts that were estimated and any errors or consistency warnings.
- The queries and errors are logged to nqquery.log on the Oracle BI Server.

  Set the log level at 4 or higher to write this information to the log file. For more information about logging, see "Diagnosing and Resolving Issues in Oracle Business Intelligence" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

To populate logical level counts automatically:

1. In the Administration Tool, open a repository in online mode.
2. Right-click one or more business models and dimension objects, and select **Estimate Levels**.
3. In the Check Out Objects dialog, click **Yes** to check out the objects that appear in the list.

   If you click **No**, the action terminates because you must check out items to run Estimate Levels.

   In the Administration Tool dialog, a list of the dimension level counts and any errors or warning messages appear.

When you check in the objects, you can check the global consistency of your repository.

Creating and Managing Dimensions with Parent-Child Hierarchies

A parent-child hierarchy is a hierarchy of members that all have the same type. This contrasts with level-based hierarchies, where members of the same type occur only at a single level of the hierarchy.

This section contains the following topics:

About Parent-Child Hierarchies

The most common real-life occurrence of a parent-child hierarchy is an organizational reporting hierarchy chart, where the following all apply:

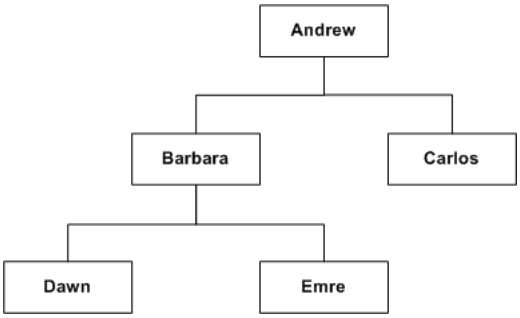
- Each individual in the organization is an employee.
- Each employee, apart from the top-level managers, reports to a single manager.
- The reporting hierarchy has many levels.

These conditions illustrate the basic features that define a parent-child hierarchy, namely:

- A parent-child hierarchy is based on a single logical table (for example, the "Employees" table)
- Each row in the table contains two identifying keys, one to identify the member itself, the other to identify the "parent" of the member (for example, Emp_ID and Mgr_ID)

Figure 10–4 shows an example of a multi-level parent-child hierarchy.

Figure 10–4 Multi-Level Parent-Child Hierarchy



The following table shows how this parent-child hierarchy could be represented by the rows and key values in an Employees table.

Emp_ID	Mgr_ID
Andrew	*null*
Barbara	Andrew
Carlos	Andrew
Dawn	Barbara
Emre	Barbara

You can expose logical dimensions with parent-child hierarchies to Oracle BI Answers users by creating presentation hierarchies that are based on particular logical dimensions. Creating hierarchies in the Presentation layer enables users to create hierarchy-based queries. See "Working with Presentation Hierarchies and Levels" for more information.

This section contains the following topics:

- About Levels and Distances in Parent-Child Hierarchies
- About Parent-Child Relationship Tables

About Levels and Distances in Parent-Child Hierarchies

Unlike the situation with level-based hierarchies, all the dimension members of a parent-child hierarchy occur in a single logical column.

In a parent-child hierarchy, the parent of a member is in another row in the same logical column, pointed to by the parent key. This is unlike a level-based hierarchy, where the parent of a member is in a different logical column in the same row. In other words, navigation in a parent-child hierarchy follows data values, while navigation in a level-based hierarchy follows the metadata structure.

In level-based hierarchies, each level is named, and occupies a position in the hierarchy that corresponds to a real-word attribute or category that is deemed useful for analysis. Unlike level-based hierarchies, where the number of levels is fixed at design time, there is no limit to the depth of a parent-child hierarchy, and the depth can change at run time due to new data."

Every Oracle BI Server parent-child hierarchy has two system-generated entities, "Total" and "Detail," that are automatically defined for each parent-child hierarchy when the logical dimension is created. The "Detail" entity contains all the hierarchy members.

These two system-generated entities are different from the implicit, inter-member levels between ancestors and descendants in a parent-child hierarchy. These implicit levels are referred to as parent-child hierarchical levels in this section.

Closely associated with levels is the concept of distance in parent-child hierarchies. The distance of one member from another is the number of parent-child hierarchical levels from the member to an ancestor or to a descendant. For example, the distance from a member to its parent is always 1, and the distance from Andrew to Emre in Figure 10–4 is 2.

About Parent-Child Relationship Tables

In relational tables, the relationships between different members in a parent-child hierarchy are implicitly defined by the identifier key values in the associated base table.

However, for each Oracle BI Server parent-child hierarchy defined on a relational table, you must also explicitly define the inter-member relationships in a separate parent-child relationship table.

The parent-child relationship table must include four columns, as follows:

- A column that identifies the member
- A column that identifies an ancestor of the member

Note: The ancestor may be the parent of the member, or a higher-level ancestor.

- A "distance" column that specifies the number of parent-child hierarchical levels from the member to the ancestor
- A "leaf" column that indicates if the member is a leaf member (1=Yes, 0=No)

The column names can be user defined. The data types of the columns must satisfy the following conditions:

- The member and ancestor identifier columns have the same data type as the associated columns in the logical table that contains the hierarchy members.

  Note that the example shown in Table 10–1 uses text strings for readability, but you normally use integer surrogate keys for member_key and ancestor_key, if they exist in the source dimension table.
- The "distance" and "leaf" columns are `INTEGER` columns.

Note the following about the rows in a parent-child relationship table:

- Each member must have a row pointing at itself, with distance zero.
- Each member must have a row pointing at each of its ancestors. For a root member, this is a termination row with null for the parent and distance values.

Table 10–1 shows an example of a parent-child relationship table with rows that represent the inter-member relationships for the employees shown in Figure 10–4.

Table 10–1 Example Parent-Child Relationship Table

Member_Key	Ancestor_Key	Distance	Isleaf
Andrew	Andrew	0	0
Barbara	Barbara	0	0
Carlos	Carlos	0	0
Dawn	Dawn	0	0
Emre	Emre	0	0
Andrew	null	null	0
Barbara	Andrew	1	0
Carlos	Andrew	1	1
Dawn	Barbara	1	1
Dawn	Andrew	2	1
Emre	Barbara	1	1
Emre	Andrew	2	1

Typically, you generate scripts to create and populate the parent-child relationship table through a wizard that you can invoke when you define the parent-child hierarchy. Note the following about the create and load scripts:

- You run the create script only once, to create the parent-child relationship table in the data source.

- You must run the load script after each time the data changes in the dimension table. Because of this, you typically call the load script in your ETL processing. The load script reloads the entire parent-child relationship table; it is not incremental.

If you do not choose to use the wizard, then you must create the parent-child relationship table manually and then import it into the Physical layer before associating it with the parent-child hierarchy. In this latter case, it is also your responsibility to populate the table with the data required to describe the inter-member relationships in the parent-child hierarchy.

Creating Dimensions with Parent-Child Hierarchies

The key elements that you must define for a parent-child hierarchy are the identifier columns for the member and the parent of the member. This basic principle applies to all parent-child hierarchies, regardless of the data source from which the hierarchy is derived.

Parent-child hierarchies that are based on relational tables must have an accompanying parent-child relationship table. See "About Parent-Child Relationship Tables" for more information.

To create dimensions with a parent-child hierarchy:

1. In the Business Model and Mapping layer of the Administration Tool, perform one of the following steps:
   - Right-click a business model and select **New Object > Logical Dimension > Dimension with Parent-Child Hierarchy**. Note that this option is only available if there is at least one logical dimension table in the business model that has no dimension associated with it.
   - Right-click a dimension table that is not associated with any dimension and select **Create Logical Dimension**, then select **Dimension with Parent-Child Hierarchy**.
2. In the Logical Dimension dialog, in the General tab, type a name for the dimension.
3. Click **Browse** beside the Member Key field.

   The Browse window shows the logical dimension tables in the business model, each with their primary keys.
4. Select a Member Key for the parent-child hierarchy and click **OK**.
5. Click **Browse** beside the Parent Key field.

   The Browse window shows the columns, other than the primary key, in the logical table that you selected in step 4.
6. Select a column that will be the Parent Key for the parent-child hierarchy and click **OK**.
7. If the logical table that you selected in step 4 is not from a relational table source, click **OK** to finish the process of creating the dimension.

   If the logical table you selected in step 4 is from a relational table source, you must continue the dimension definition process by setting up the parent-child relationship table for the hierarchy. See "Defining Parent-Child Relationship Tables" for details.

Defining Parent-Child Relationship Tables

For parent-child hierarchies based on relational tables, you must define a parent-child relationship table.

See "About Parent-Child Relationship Tables" for more information.

When you create the parent-child relationship table, you must choose one of the following options:

- Select a previously-created parent-child relationship table
- Use a wizard that will generate scripts to create and populate the parent-child relationship table

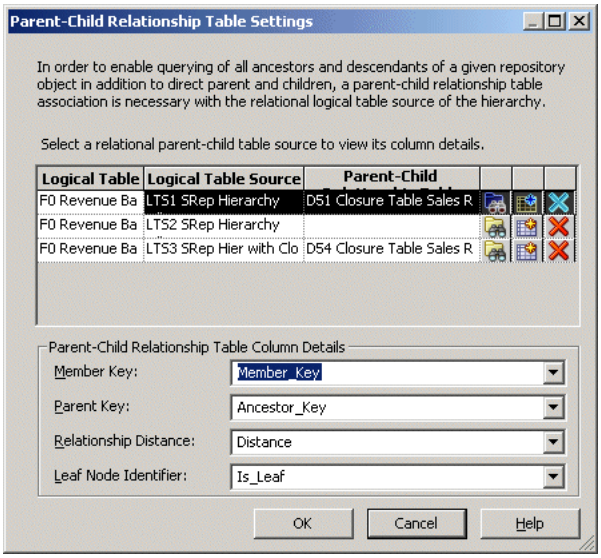
To define parent-child relationship tables:

1. In the Logical Dimension dialog, click **Parent-Child Settings**.

   The Parent-Child Relationship Table Settings windows appears, with the Logical Table and Logical Table Source values filled in.

   Figure 10–5 shows the Parent-Child Relationship Table Settings dialog.

Figure 10–5 Parent-Child Relationship Table Settings Dialog



2. You can either manually define the parent-child relationship table for the hierarchy, or you can start a wizard that will perform the definition for you (recommended).
   - To start the manual process, continue at step 3.
   - To start the wizard, continue at step 7.
3. Click the **Select Parent-Child Relationship Table** button to start the manual process of defining the parent-child relationship table for the parent-child hierarchy.
4. Select the physical table that acts as the parent-child relationship table for your parent-child hierarchy. The table must already exist in the Physical layer.

   The parent-child relationship table must have at least four columns that describe how the inter-member relationships are derived in the logical table selected for the hierarchy. See "About Parent-Child Relationship Tables" for more information.

5. Map the four columns from the physical parent-child relationship table to the fields in the Parent-Child Table Column Details area, as follows:
   - Select the Member Key column
   - Select the Parent Key column
   - Select the Relationship Distance column
   - Select the Leaf Node Identifier column
6. Click **OK**, then click **OK** again to finish the manual process of defining the parent-child relationship table.
7. Click the **Create Parent-Child Relationship Table** button to start the wizard.

   The Generate Parent-Child Relationship Table Wizard generates SQL scripts for creating and populating the parent-child relationship table. At the end of the wizard, the Oracle BI Server stores the scripts into directories chosen during the wizard session. The scripts, when executed, will make the parent-child relationship table available to the metadata repository.

   The wizard contains the following three main windows:
   - Script Location
   - Parent-Child Relationship Table Details
   - Preview Script
8. In the Generate Parent-Child Relationship Table - Script Location screen, enter the **Name** for the **DDL Script to Generate the Parent-Child Table**, and select the **Location** where the Generate script will be placed.
9. Enter the **Name** for the **DDL Script to Populate the Parent-Child Table**, and select the **Location** where the Populate script will be placed.
10. Click **Next**.
11. In the Parent-Child Relationship Table Details screen, enter the **Name** for the parent-child relationship table.
12. Click **Browse** beside the Catalog/Schema field to select the catalog or schema for the parent-child relationship table.
13. Click **Next**.
14. In the Preview Script window, you can view either or both of the scripts.
15. Click **Finish**.
16. In the Parent-Child Relationship Table Settings window, click **OK**.
17. In the Logical Dimension window, click **OK**.
18. If you used the Generate Parent-Child Relationship Table Wizard to generate create and load scripts, run the scripts to create and load the parent-child relationship table in your data source.

Modeling Aggregates for Parent-Child Hierarchies

Fact tables in level-based hierarchies typically only contain facts for a single level of the hierarchy. Facts for higher-level dimension members can be calculated by aggregating the facts from the lower-level fact table or from a higher-level summary table.

In contrast, parent-child hierarchies require data modelers to make some additional decisions related to the following:

- How to store the base facts in the fact table
- How to aggregate the base facts to obtain the facts for higher-level members of the parent-child hierarchy

This section describes how to store and aggregate facts for parent-child hierarchies and contains the following topics:

Storing Facts for Parent-Child Hierarchies

There are two options for storing the base facts in the fact table for parent-child hierarchies:

- Store facts for only the leaf members of the parent-child hierarchy.
- Store facts for members at any level of the parent-child hierarchy, including non-leaf members.

The first option is more appropriate if the facts for the non-leaf members of the parent-child hierarchy can be derived entirely from the facts of the leaf members. For example, if you have a parent-child product hierarchy in which the actual product members appear only as leaf members of the hierarchy, then it makes sense for a revenue fact table to only record revenue facts for the leaf members of this product hierarchy. The revenue figures for the non-leaf members of the product hierarchy (such as the product categories) can be derived entirely by aggregating the facts for the leaf product members at the bottom of the hierarchy.

Example 10–3 shows example data for a situation where facts are stored only for leaf members in a parent-child hierarchy.

Example 10–3 Facts Stored for Leaf Member Only

The following table shows example data for the dimension table PRODUCT_DIM:

MemberKey	Name	ParentKey
P1	Product1	C1
P2	Product2	C1
C1	Category1	C2
C2	Category2	C3
C3	Category3	-

The following table shows example data for the fact table REVENUE_FACTS:

ProductKey	YearKey	Revenue
P1	2011	100,000
P1	2012	105,000
P2	2011	75,000
P2	2012	80,000

The second option in which facts are stored for members at any level of the parent-child hierarchy is necessary when the facts for the non-leaf members are not completely derived from facts of the leaf members. A good example is a sales person hierarchy in which a sales person might report to a manager who is also a sales person. Each individual sales person, including the manager, could have a different revenue figure stored in the fact table.

Example 10–4 shows example data for this situation.

Example 10–4 Facts Stored for Both Leaf and Non-Leaf Members

The following table shows example data for the dimension table SALES_REP_DIM:

MemberKey	Name	ParentKey
101	Phillip	201
102	Vivian	201
201	Jacob	301
202	Audrey	301
301	Ryan	-

The following table shows example data for the fact table REVENUE_FACTS:

SalesRepKey	YearKey	Revenue
101	2012	1,200,000
102	2012	1,100,000
201	2012	250,000
202	2012	1,400,000

Another case in which storing facts for both leaf and non-leaf members is appropriate is when the rules for aggregating the parent-child hierarchy are complex, or when aggregating the hierarchy at query time is expensive and would lead to unacceptably long query response times. In this case, the fact table would store preaggregated facts for the non-leaf members in addition to the facts stored for the leaf members.

Aggregating Parent-Child Hierarchies

Whether or not the fact table contains data for both leaf members and non-leaf members or just for leaf members, the data modeler must determine how to aggregate the stored facts to calculate the aggregated facts for higher level members of the parent-child hierarchy. In addition to choosing the correct aggregation function for the measure, the data modeler must decide whether the fact values recorded for lower-level members should be rolled up to calculate the values for higher-level members. In some cases, rolling up the facts of lower-level members of the parent-child hierarchy makes sense. In other cases, such as with a preaggregated fact table or a measure that is intended to show each member's individual contribution, rolling up the facts from lower-level members of the parent-child hierarchy is incorrect.

Rolling up Facts from Lower-Level Members of a Parent-Child Hierarchy If a fact table only stores facts for the leaf members of a parent-child hierarchy or if the fact table only records each member's individual contribution, then most likely the values

stored in the fact table must be rolled up to obtain the correct aggregated value for higher-level members of the parent-child hierarchy. Rolling up the facts along a parent-child hierarchy is achieved by joining the fact table to the dimension table through the parent-child relationship table. See "Adding the Parent-Child Relationship Table to the Model" for more information.

For a fact table that stores facts only for the leaf members such as the product revenue fact table in Example 10–3, this modeling technique calculates aggregate values that correctly summarize all the facts for the leaf-level members.

For a fact table that stores the individual contribution of both leaf members and non-leaf members such as the sales rep revenue fact table in Example 10–4, this technique computes a hierarchical aggregate that summarizes the individual contributions of the member and all the members beneath it.

Modeling Individual Contribution Measures To be able to report the individual contribution of each member in addition to reporting the summarized hierarchical aggregate that rolls up the individual contributions of multiple members, the data modeler must create two separate fact logical table sources. One fact logical table source maps the base fact table and the parent child relationship table. This is the logical table source for the hierarchical aggregate measure. The second fact logical table source maps only an alias of the fact table. This fact table alias should join directly with the dimension table rather that joining indirectly through the parent-child relationship table. This is the logical table source for the individual contribution measure.

Modeling Preaggregated Measures Some fact tables contain preaggregated data that is populated for all members of the parent-child hierarchy. For example, the fact value for a root member might be populated with the aggregation of the data for all of its descendent members. It is important to ensure that queries do not aggregate the members from this dimension to avoid erroneous results.

To correctly model this type of parent-child hierarchy, you must still create a parent-child relationship table to support hierarchical filter functions like IsAncestor and IsDescendant. However, you can join the parent-child dimension table directly with the fact table rather than joining through the parent-child relationship table. Doing so ensures that the preaggregated member value is returned, rather than rolling up all the descendants.

Note: Do not modify the parent-child relationship table script to only include the "self" rows, because doing so would break the IsAncestor and IsDescendant functions.

To achieve the correct aggregation for dimensions of this type, you must first determine what you want to see as a grand total when the parent-child hierarchy is aggregated. For example, assume that your hierarchy contains a single root member, and you want to display the preaggregated value for this root member. In this case, you first create an additional fact logical table source mapped at the Total level of the parent-child hierarchy. Then, in the logical table source, create a WHERE clause filter that selects only the root member.

With this model in place, for queries that are at the Total level of the parent-child hierarchy, the Oracle BI Server selects the aggregate logical table source and applies the root member WHERE clause filter. For queries that are at the Detail level, the Oracle BI Server selects the detailed logical table source and returns the preaggregated

member values. In either case, it does not matter how the aggregation rule is set, because there is a preaggregated source at each level.

Note that this approach only works if the queries are either at the Total or Detail level of the parent-child dimension. However, for queries that group by some non-unique attribute of the parent-child dimension, the aggregation might not be correct. For example, if an Employee dimension has a Location attribute, and a query groups by Employee.Location, then there will likely be some double counting because an employee often reports to other employees at the same location. Because of this, when fact tables contain preaggregated member values, you should avoid grouping by non-unique attributes of the parent-child dimension. If grouping by those attributes is unavoidable, then you should model them as separate dimensions.

Adding the Parent-Child Relationship Table to the Model

For measures in fact tables that should be aggregated by rolling up the facts from lower-level members, you must edit Physical layer joins to include the parent-child relationship table. You also need to add the parent-child relationship table to the appropriate logical table source.

Note: "For fact tables containing preaggregated data for a parent-child hierarchy or for individual contribution measures, you should join the parent-child dimension table directly with the fact table rather than joining through the parent-child relationship table.

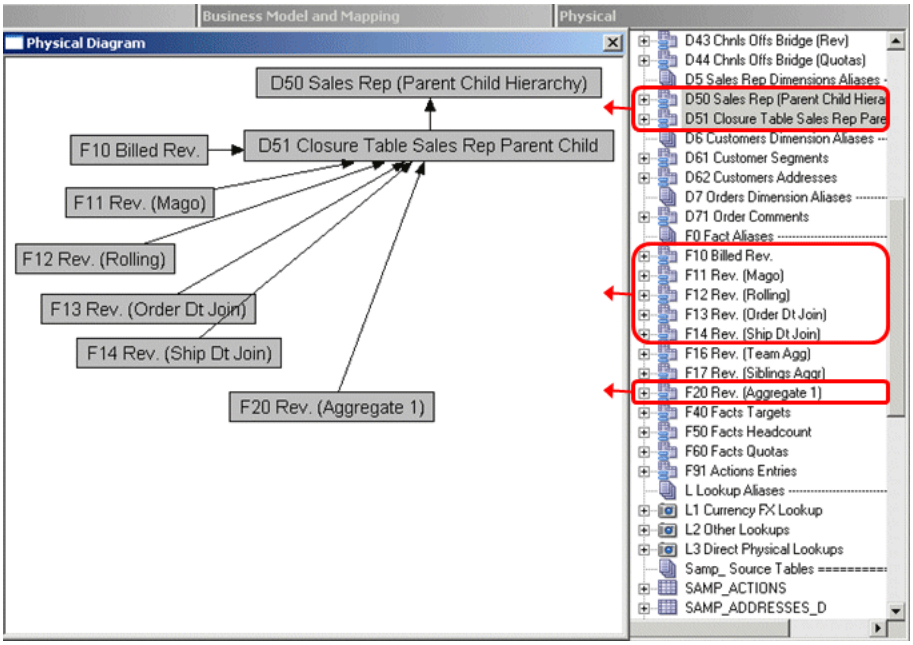
Doing so ensures that the preaggregated value or individual contribution value is returned, rather than rolling up all the descendants. Also, for situations where preaggregated data is populated for all members, do not add the parent-child relationship table to the logical table source to avoid overcounting. See "Modeling Preaggregated Measures" for more information.

To add the parent-child relationship table to the model:

1. In the Administration Tool, in the Physical layer of the repository, open the Physical Diagram so that it shows the parent-child relationship table and associated dimension table and fact tables. To do this, right-click the appropriate physical tables and select **Physical Diagram > Selected Object(s) Only**.
2. Delete the direct joins from the dimension table to each of the fact tables.
3. Create joins from each the fact tables to the dimension table through the parent-child closure table, as follows:
   a. Create a join from the parent-child relationship table to the dimension table using the ancestor key.
   b. Create joins from the fact tables to the parent-child relationship table using the member key.

   Figure 10–6 shows joins from a dimension table to fact tables that go through a parent-child relationship table.

Figure 10–6 Physical Layer Joins Through a Parent-Child Relationship Table



4. In the Business Model and Mapping layer, double-click the logical table source for the logical fact table that is used in your parent-child hierarchy.
5. In the General tab of the Logical Table Source dialog, click the **Add** button.
6. Browse to locate the parent-child relationship table in the Physical layer and click **Select**.
7. Click **OK** in the Logical Table Source dialog.

Maintaining Parent-Child Hierarchies Based on Relational Tables

For parent-child hierarchies based on relational tables, you must ensure that the data in the parent-child relationship table accurately reflects the inter-member relationships in the dimension.

You may have created scripts manually to create and populate the parent-child relationship table, or you may have used the Generate Parent-Child Relationship Table Wizard to create the scripts. You must run these scripts, adapting them if necessary, as often as required to guarantee the integrity of the parent-child relationships in the hierarchy. You typically want to add the Populate script to your ETL process so that it runs after the dimension table is updated.

Modeling Time Series Data

Time series functions provide the ability to compare business performance with previous time periods, allowing you to analyze data that spans multiple time periods. For example, time series functions enable comparisons between current sales and sales a year ago, a month ago, and so on.

Because SQL does not provide a direct way to make time comparisons, you must model time series data in the Oracle BI repository. First, set up time dimensions based on the period table in your data warehouse. Then, you can define measures that take advantage of this time dimension to use the `AGO`, `TODATE`, and `PERIODROLLING` functions. At query time, the Oracle BI Server then generates highly optimized SQL

that pushes the time offset processing down to the database whenever possible, resulting in the best performance and functionality.

This section contains the following topics:

- About Time Series Functions
- Creating Logical Time Dimensions
- Creating AGO, TODATE, and PERIODROLLING Measures

About Time Series Functions

Time series functions operate on time-oriented dimensions. To use these functions on a particular dimension, you must designate the dimension as a Time dimension and set one or more keys at one or more levels as chronological keys. These keys identify the chronological order of the members within a dimension level.

Time series functions include `AGO`, `TODATE`, and `PERIODROLLING`. These functions let you use Expression Builder to call a logical function to perform time series calculations instead of aliasing physical tables and modeling logically. The time series functions calculate `AGO`, `TODATE`, and `PERIODROLLING` functions based on the calendar tables in your data warehouse, not on standard SQL date manipulation functions.

Figure 10–7 shows a sample report that includes several measures derived using time series functions.

Figure 10–7 Example Measures Derived Using Time Series Functions

	2008 Q1			2008 Q2			2008 Q3			2008 Q4		
	2008 / 01	2008 / 02	2008 / 03	2008 / 04	2008 / 05	2008 / 06	2008 / 07	2008 / 08	2008 / 09	2008 / 10	2008 / 11	2008 / 12
Dollars	100	200	300	101	202	303	110	220	330	444	555	666
Dollars Qago				100	200	300	101	202	303	110	220	330
Dollars QTD	100	300	600	101	303	606	110	330	660	444	999	1,665
Dollars 3-Period Rolling Sum	100	300	600	601	603	606	615	633	660	994	1,329	1,665
Dollars 3-Period Rolling Avg	33.3	100.0	200.0	200.3	201.0	202.0	205.0	211.0	220.0	331.3	443.0	555.0

Several different grains may be used in the time query, such as:

- **Query grain.** The lowest time grain of the request. In the report example shown in Figure 10–7, the query grain is Month.
- **Time Series grain.** The grain at which the aggregation or offset is requested, for both `AGO` and `TODATE` functions. In the report example shown in Figure 10–7, the time series grain is Quarter. Time series queries are valid only if the time series grain is at the query grain or higher. Note that the `PERIODROLLING` function does not have a time series grain; instead, you specify a start and end period in the function.
- **Storage grain.** The report example shown in Figure 10–7 can be computed from daily sales or monthly sales. The grain of the source is called the storage grain. A chronological key must be defined at this level for the query to work, but performance is generally much better if a chronological key is also defined at the query grain.

Note that queries against time series data must be an exact match to hit the query cache. See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about the Oracle BI Server query cache.

The following sections describe the time series conversion functions:

- About the AGO Function
- About the TODATE Function
- About the PERIODROLLING Function

About the AGO Function

The AGO function offsets the time dimension to display data from a past period. This function is useful for comparisons, such as *Dollars* compared to *Dollars a Quarter Ago*. Note that the value of "Dollars Qago" for month 2008/08 equals the value of "Dollars" for month 2008/05.

Figure 10–8 shows example values for the Dollars and Dollars Qago measures.

Figure 10–8 Example Dollars and Dollars Qago Measures

	2008 Q1			2008 Q2			2008 Q3			2008 Q4		
	2008 / 01	2008 / 02	2008 / 03	2008 / 04	2008 / 05	2008 / 06	2008 / 07	2008 / 08	2008 / 09	2008 / 10	2008 / 11	2008 / 12
Dollars	100	200	300	101	202	303	110	220	330	444	555	666
Dollars Qago				100	200	300	101	202	303	110	220	330

In the example shown in Figure 10–8, the Dollars Qago measure is derived from the Dollars measure.

In Expression Builder, the AGO function has the following template:

```
Ago(<<Measure>>, <<Level>>, <<Number of Periods>>)
```

<<Measure>> represents the logical measure column from which you want to derive. In this example, you would select the measure "Dollars" from your existing logical fact tables.

<<Level>> is the optional time series grain you want to use. In this example, you would select "Quarter" from your time dimension.

<<Number of Periods>> is the size of the offset, measured in the grain you provided in the <<Level>> argument. For example, if the <<Level>> is Quarter and the <<Number of Periods>> is 2, the function displays dollars from two quarters ago.

Using this function template, you can create an expression for a One Quarter Ago measure, as follows:

```
Ago("Sales"."Base Measures"."Dollars" , "Sales"."Time MonthDim"."Quarter" , 1)
```

The <<Level>> parameter is optional. If you do not want to specify a time series grain in the AGO function, the function uses the query grain as the time series grain.

For example, you could define Dollars_Ago as `Ago(Dollars, 1)`. Then, you could perform the following logical query:

```
SELECT Month, Dollars, Dollars_Ago
```

The result is the same as if you defined Dollars_Ago as `Ago(Dollars, Month, 1)`. Alternatively, you could perform the following logical query:

```
SELECT Quarter, Dollars, Dollars_Ago
```

The result is the same as if you defined Dollars_Ago as `Ago(Dollars, Quarter, 1)`.

See "AGO" for additional information about the AGO function syntax.

About the TODATE Function

The TODATE function accumulates the measure from the beginning of the time series grain period to the current displayed query grain period.

For example, Figure 10–9 shows a report with the measure "Dollars QTD," which is the Quarter To Date version of the "Dollars" measure.

Figure 10–9 Example Dollars and Dollars QTD Measures

	2008 Q1			2008 Q2			2008 Q3			2008 Q4		
	2008 / 01	2008 / 02	2008 / 03	2008 / 04	2008 / 05	2008 / 06	2008 / 07	2008 / 08	2008 / 09	2008 / 10	2008 / 11	2008 / 12
Dollars	100	200	300	101	202	303	110	220	330	444	555	666
Dollars QTD	100	300	600	101	303	606	110	330	660	444	999	1,665

In the example shown in Figure 10–9, Dollars QTD for Month 2008/05 is the sum of Dollars for 2008/04 and 2008/05. In other words, Dollars QTD is the sum of the values for all the query grain periods (month) for the current time series grain period (quarter). The accumulation starts over for the next quarter.

In the example shown in Figure 10–9, the Dollars QTD measure is derived from the Dollars measure.

In Expression Builder, the TODATE function has the following template:

```
ToDate(<<Measure>>, <<Level>>)
```

<<Measure>> represents the logical measure column from which you want to derive. In this example, you would select the measure "Dollars" from your existing logical fact tables.

<<Level>> is the time series grain you want to use. In this example, you would select "Quarter" from your time dimension.

Using this function template, you can create the following expression for the measure:

```
ToDate("Sales"."Base Measures"."Dollars" , "Sales"."Time MonthDim"."Quarter" )
```

Note that the query grain is specified in the query itself at run time. For example, this measure can display Quarter To Date at the Day grain, but still accumulates up to the end of the Quarter.

See "TODATE" for additional information about TODATE function syntax.

About the PERIODROLLING Function

The PERIODROLLING function lets you perform an aggregation across a specified set of query grain periods, rather than within a fixed time series grain. The most common use is to create rolling averages, such as "13-week Rolling Average."

Note that because this function has no time series grain, the length of the rolling sequence is determined by the query grain. For example, "Dollars 3-Period Rolling Average" averages the last 3 months if the query grain is Month, but averages the last 3 years if the query grain is Year.

This section describes how to build two measures using the PERIODROLLING function: "Dollars 3-Period Rolling Sum," and "Dollars 3-Period Rolling Average." Figure 10–10 shows a report with these two measures.

Figure 10–10 Example Dollars, Dollars 3-Period Rolling Sum, and Dollars 3-Period Rolling Avg Measures

	2008 Q1			2008 Q2			2008 Q3			2008 Q4		
	2008 / 01	2008 / 02	2008 / 03	2008 / 04	2008 / 05	2008 / 06	2008 / 07	2008 / 08	2008 / 09	2008 / 10	2008 / 11	2008 / 12
Dollars	100	200	300	101	202	303	110	220	330	444	555	666
Dollars 3-Period Rolling Sum	100	300	600	601	603	606	615	633	660	994	1,329	1,665
Dollars 3-Period Rolling Avg	33.3	100.0	200.0	200.3	201.0	202.0	205.0	211.0	220.0	331.3	443.0	555.0

In the example shown in Figure 10–10, the Dollars 3-Period Rolling Sum and Dollars 3-Period Rolling Avg measures are derived from the Dollars measure.

In Expression Builder, the `PERIODROLLING` function has the following template:

```
PeriodRolling(<<Measure>>, <<Starting Period Offset>>, <<Ending Period Offset>>)
```

<<Measure>> represents the logical measure column from which you want to derive. To create the measure Dollars 3-Period Rolling Sum, you would select the measure "Dollars" from your existing logical fact tables.

`<<Starting Period Offset>>` and `<<Ending Period Offset>>` identify the first period and last period used in the rolling aggregation, respectively. The integer is the relative number of periods from the displayed period. In the example shown in Figure 10–10, the query grain is month, and the 3-month rolling sum starts 2 periods in the past and includes the current period. That is, for month 2008/07, the rolling sum includes 2008/05, 2008/06 and 2008/07. Therefore, to create the measure Dollars 3-Period Rolling Sum, the integers to indicate these offsets are -2 and 0.

Using this function template, you can create the following expression for the measure:

```
PeriodRolling("Sales"."Base Measures"."Dollars" , -2, 0)
```

The example shown in Figure 10–10 also shows a 3-month rolling average. To compute this measure, you can divide the rolling sum that you previously created by 3 to get a 3-period rolling average. We know to divide the rolling sum by 3 because the `<<Starting Period Offset>>` and `<<Ending Period Offset>>` fields for the rolling sum are -2 and 0.

The expression for the 3-month rolling average is:

```
PeriodRolling("Sales"."Base Measures"."Dollars" , -2, 0) /3
```

It is usually a mistake to use the `AVG` function to create a rolling average. `AVG` computes the average of the database rows accessed at the storage grain, but you need an average where the denominator is the number of rolling periods at the query grain.

Note that the `PERIODROLLING` function includes a fourth optional hierarchy argument that lets you specify the name of a hierarchy in a time dimension, such as `yr, mon, day`, that you want to use to compute the time window. This option is useful when there are multiple hierarchies in a time dimension, or when you want to distinguish between multiple time dimensions. See "PERIODROLLING" for more information about the hierarchy argument and for details on the function syntax.

Creating Logical Time Dimensions

Compared to modeling an ordinary dimension, the time dimension requires just two additional steps: selecting the **Time** option in the Logical Dimension dialog, and designating a chronological key for every level of every dimension hierarchy.

Follow these additional guidelines when modeling time series data:

- It only makes sense to use a time series function when the data source contains history. Usually, a relational database that contains history uses a star or snowflake schema with an explicit time dimension table. A normalized, historical database is much rarer, but would still include a time hierarchy with levels in a schema similar to a snowflake. A simple date field is not adequate.
- Oracle Business Intelligence requires the time dimension physical table (or set of normalized tables) to be separate from the physical fact table to which it is related.

  However, a somewhat common source schema pattern is a fully denormalized relational table or flat file, where the time dimension columns are in the same table as the facts and other dimensions. This cannot qualify as a time dimension, because the time dimension table is combined with the fact table.

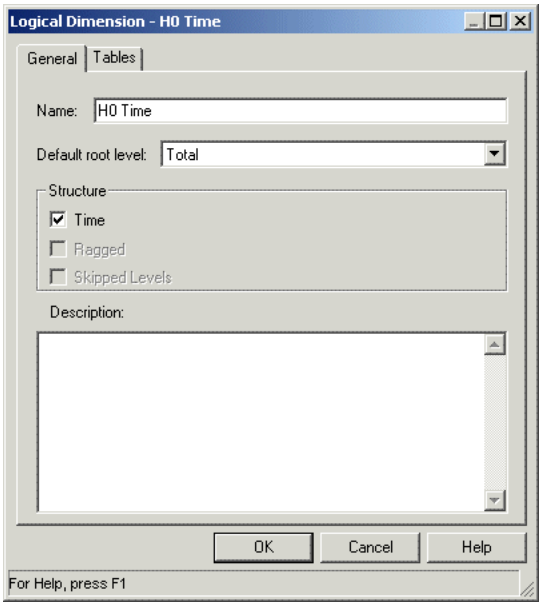
  In this case, if you cannot change the model in the source, the best practice is to create an Opaque View of the physical table containing the time columns, which acts as the distinct physical time dimension table. This Opaque View time dimension must then be joined to the physical table that contains the facts.
- In the Physical layer, the time dimension table (or lowest-level table in the normalized/snowflake case) must join directly to the fact table without intervening tables. This join must be a foreign key join.
- The tables in the physical model containing the time dimension cannot join to other data sources, except at the most detailed level.
- A member value (for example, a row in relational sources) must be physically present for every period at every hierarchy level. There cannot be any skips in the sequence. Note that it does not matter whether there is fact data for every period; only the dimension data must be complete.
- Each unit of distance between members, such as "month," "half," or "year," must be modeled in a separate hierarchy level.

Selecting the Time Option in the Logical Dimension Dialog

Select the **Time** option in the General tab of the Logical Dimension dialog to enable time series functions on this dimension. Only logical dimensions with the **Time** option selected can be used as the time dimension for the time series functions `AGO`, `TODATE`, and `PERIODROLLING`.

Figure 10–11 shows the Time option in the Logical Dimension dialog.

Figure 10–11 Time Option in Logical Dimension Dialog



Setting Chronological Keys for Each Level

Designate a chronological key for every level of each dimension hierarchy. The chronological keys must be comparable with the standard SQL `ORDER BY` clause. The `ORDER BY` clause on the chronological key must reflect the real world chronological order of the time dimension members represented by the key. For example, if the time dimension members are: Jan-3-2013, Jan-4-2013, Jan-5-2013 then the following chronological keys can be assigned to them in the same order: 1, 5, 9. However, assigning chronological keys such as 2,1,3 would result in Jan-4-2013, Jan-3-2013, Jan-5-2013, which is an incorrect chronological order.

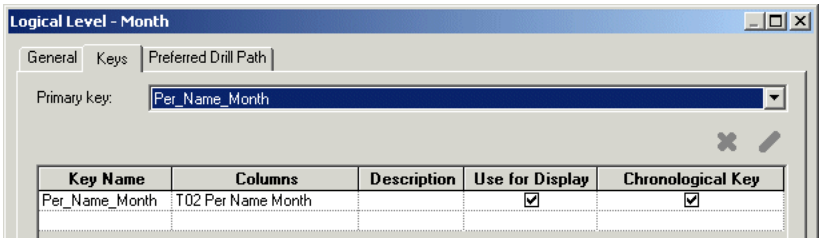
The Oracle BI Server uses the chronological key to create mathematically correct time series predictions, such as Jan + 2 months = Mar. You should set a chronological key for every level (except for the Grand Total level) so that you can perform time series operations on all levels with good performance. This enables you to use an `AGO`, `TODATE`, or `PERIODROLLING` function for any level of any time dimension hierarchy, such as fiscal month ago, calendar year ago, and day ago.

Theoretically, time series functions operate correctly if only the bottom level key in the Logical Dimension is chronological. In practice, however, this causes performance problems because it forces the physical query to use the lowest grain, causing joins of orders of magnitude more rows (for example, 365 times more rows for a "year ago" joining at the "day" grain).

As with any level key, be sure the key is unique at its level. For example, a column containing simple month names such as "January" is not unique unless it is concatenated to a column containing year names.

Figure 10–12 shows how to designate a chronological key in the Logical Level dialog.

Figure 10–12 Designating a Chronological Key in the Logical Level Dialog



Creating AGO, TODATE, and PERIODROLLING Measures

You can build time series measures by creating derived expressions from base measures. To do this, create a new logical column and select **Derived from existing columns using an expression**, then open Expression Builder to build the appropriate time series function.

Follow these guidelines when modeling time series functions:

- Time series functions cannot be derived from measures that use the fragmentation form of federation. This rule prevents some complex boundary conditions and cross-source assumptions in the query generation and result merging, such as the need to join some time dimension rows from one source to some of the fact rows in a different source.
- To reduce maintenance and increase accuracy, it is best to create a single base measure, and then derive a family of time series measures from it. For example, start with a base measure, then define variations for month-ago, year-ago, month-to-date, and so on. To do this, select **Derived from existing columns using an expression** and refer to the base measure in the expression.

Example 10–5 shows how to build the `AGO` measure. See Appendix C, "Logical SQL Reference" for detailed syntax for the other time series functions, `TODATE` and `PERIODROLLING`.

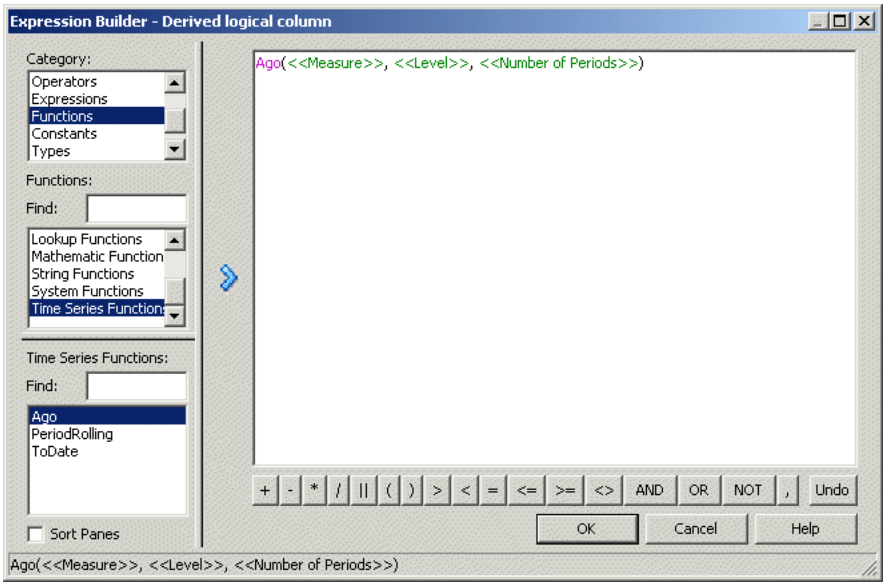
Example 10–5 Creating the AGO Measure

This example explains how to create one of the derived `AGO` measures in the Sampleapp demonstration repository.

1. In the Business Model and Mapping layer, create a new logical column. Name the column **2-04 Billed Qty (Mago)**.
2. In the Column Source tab, select **Derived from existing columns using an expression** and click the Expression Builder button.
3. In Expression Builder, select the Ago function to create a template for the arguments. To do this, select **Functions** in the Category pane, **Time Series Functions** in the Functions pane, and **Ago** in the Time Series Functions pane.

   Figure 10–13 shows the `AGO` function in Expression Builder.

Figure 10–13 AGO Function in Expression Builder

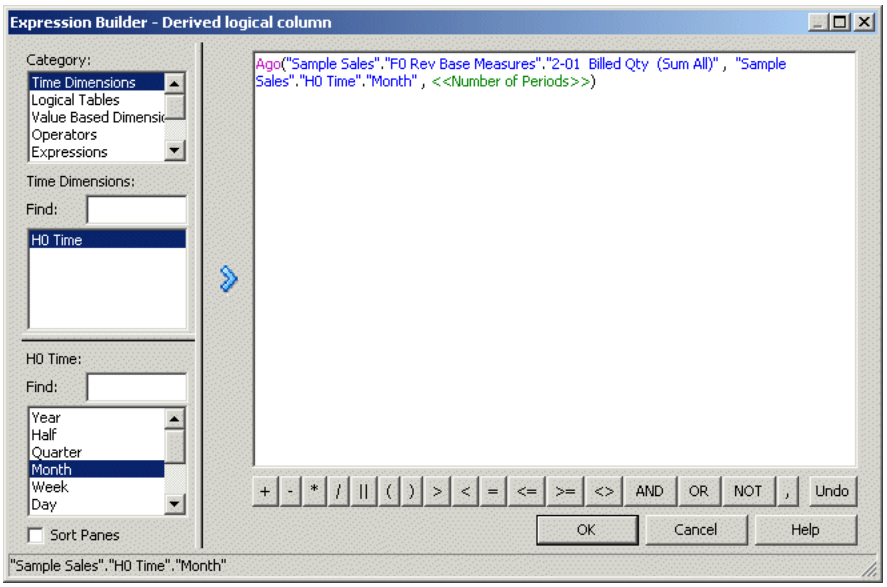


4. Select the first argument, **Measure**, then use the selection panes to select the base measure from which to derive this column. In this example, select "Sample Sales"."F0 Rev Base Measures"."2-01 Billed Qty (Sum All)."

5. Select the second argument, **Level**, then use the selection panes to select the unit of the ago offset. It must be defined as a level of the time dimension, so that it can take advantage of the chronological keys built in the time dimension. In this example, select **Time Dimensions** in the Category pane, **HO Time** in the Time Dimensions pane, and **Month** in the HO Time pane.

   Figure 10–14 shows the Month level in Expression Builder.

Figure 10–14 Selecting the Level Argument in Expression Builder



6. Select the third argument, Number of Periods, and enter the size of the offset you want to use for this measure. In this example, type **1**.

7. Click **OK** in the Expression Builder dialog, then click **OK** in the Logical Column dialog.

11

Managing Logical Table Sources (Mappings)

This chapter explains how to work with logical table source objects and their mappings in the Business Model and Mapping layer of the Oracle BI repository.

This chapter contains the following topics:

About Logical Table Sources

Logical table sources define the mappings from a single logical table to one or more physical tables. The physical to logical mapping can also be used to specify transformations that occur between the Physical layer and the Business Model and Mapping layer, as well as to enable aggregate navigation and fragmentation.

You can view logical table sources in the Business Model and Mapping layer pane, and from the Sources tab of the Logical Table dialog.

Logical tables can have many physical table sources. A single logical column might map to many physical columns from multiple physical tables, including aggregate tables that map to the column if a query asks for the appropriate level of aggregation on that column.

This section contains the following topics:

How Fact Logical Table Sources Are Selected to Answer a Query

The following criteria are used by the system to select the fact logical table source to answer a query, listed from the highest precedence to the lowest precedence:

1. **Logical table source priority group.** A higher priority logical table source is used before a lower priority logical table source, even if the higher priority source is at a more detailed grain. Note that a lower number indicates higher priority. See "Setting Priority Group Numbers for Logical Table Sources" for more information.
2. **The grain of the logical table source.** A higher-grain logical table source is used before a lower-grain logical table source, given that the priority group numbers are the same.
3. **Number of elements at this level.** If the grains are not comparable, the number specified for the **Number of elements at this level** field is considered.

   For example, assume you have the following two logical table sources with grains that are not comparable: LTS1(year, city) and LTS2(month, state). If you have 10 years, 100 cities, 120 months, and 9 states, the worst case size of LTS1 is 10 x 100 = 1000, and the worst case size of LTS2 is 120 x 9 = 1080. In this scenario, LTS1 is selected because the source with the lowest estimated number of total elements is assumed to be the fastest.

   See "Creating Logical Levels in a Dimension" for more information about this field.
4. **First logical table source listed.** If all other criteria are equal, the first logical table source listed is selected, as shown in the Business Model and Mapping layer.

Note that every column in a query is sourced from a single logical table source based on these expected performance factors. Queries are not load-balanced across multiple logical table sources.

How Dimension Logical Table Sources Are Selected to Answer a Query

After the appropriate fact logical table sources have been selected, the system selects the best dimensional logical table sources to answer the query.

The following criteria are used by the system to select the dimension logical table source, listed from the highest precedence to the lowest precedence:

1. **Logical table source priority group.** Like fact logical table sources, a higher priority dimension logical table source is used before a lower priority dimension logical table source. Note that a lower number indicates higher priority.
2. **Lower join cost.** The dimension logical table source with the lowest join cost is selected before dimension logical tables sources with higher join costs, given that the priority group numbers are the same.
3. **Higher level.** If the priority group and join cost are the same, the higher level logical table source is chosen, because that logical table source will likely require fewer rows to be joined.

Changing the Default Selection Criteria for Dimension Logical Table Sources

You can optionally change the default logical table source selection criteria to favor dimension logical table sources that are at the same level as the fact logical table source before considering the higher level logical table source. To do this, set the session variable DIMENSION_LTS_JOIN_RESTRICTIONS to PREFER_SAME_LEVEL.

If there is no suitable dimension logical table source at the same level as the fact logical table source, then the Oracle BI Server selects the highest level dimension logical table source that is joinable to the fact. Note that these factors are only considered after priority group and join cost.

The PREFER_SAME_LEVEL value for the session variable DIMENSION_LTS_JOIN_RESTRICTIONS sets the following criteria for selecting the dimension logical table source to answer the query:

1. Logical table source priority group
2. Lower join cost
3. Same level as the fact logical table source
4. Higher level than other dimension logical table sources if no other logical table source is at the same level as the fact logical table source

When DIMENSION_LTS_JOIN_RESTRICTIONS is set to NONE (the default value), fact logical table sources can be joined to a higher level dimension logical table source even if there is another joinable dimension logical table source at the same level as the fact.

Consistency Among Data in Multiple Sources

It is important to ensure that the data in your sources is consistent. For example, your year-level logical table source and your month-level logical table source for your time dimension should cover the same time period.

Be aware that consistency issues with data in your sources might become apparent when you issue queries that override null suppression (in other words, when you create an analysis in Answers and select **Include Null Values**). For example, some aggregate tables might not include the dimension records that correspond to the null fact values, such as a yearly sales aggregate table that does not include years for which there were no sales. In other words, all years in the year dimension must exist for the null values to be included in the result.

See "Understanding Null Suppression" in *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition* for more information about overriding null suppression in analyses.

Creating Logical Table Sources

When you create logical tables and columns by dragging and dropping from the Physical layer, the logical table sources are generated automatically. If you create the logical tables manually, you need to also create the sources manually.

You also add new logical table sources when multiple physical tables can be the source of information. For example, many tables could hold information for revenue. You might have three different business units (each with its own order system) where you get revenue information. In another example, you might periodically summarize revenue from an orders system or a financial system and use this table for high-level reporting.

Use the General tab of the Logical Table Source dialog to define general properties for the logical table source.

To create a logical table source:

1. In the Business Model and Mapping layer of the Administration Tool, right-click a logical table and select **New Object**, then select **Logical Table Source**.
2. In the Logical Table Source dialog, in the General tab, type a name for the logical table source.

3. Select **Disabled** to make this logical table source inactive, or leave it deselected to make the logical table source active.
4. Select **Allow Unmapped Tables** to enable this logical table source to have physical tables that are not mapped to logical columns. This option is useful for snowflake physical tables in an A > B > C configuration, where a logical table only maps to columns in A and C, but B needs to be included in the logical table source because it is in the join path between A and C.
5. Click the **Add** button. In the Browse dialog, you can view joins and select tables for the logical table source. When there are two or more tables in a logical table source, all of the participating tables must have joins defined between them.
6. To view the joins, in the Browse dialog, select a table and click **View**. After reviewing the joins in the Physical Table dialog, click **Cancel**.
7. To add tables to the table source, select the desired tables in the **Name** list and click **Select**.
8. Optionally, in the **Priority Group** field, enter a priority group number for this logical table source. See "Setting Priority Group Numbers for Logical Table Sources" for more information.
9. In the Logical Table Source dialog, click the Column Mapping tab and complete the fields. See "Defining Physical to Logical Table Source Mappings and Creating Calculated Items" for instructions.
10. In the Logical Table dialog, click the Content tab and complete the fields. See "Defining Content of Logical Table Sources" for instructions.
11. Click **OK**.

Setting Priority Group Numbers for Logical Table Sources

You can set priority group numbers to determine which logical table source should be used for queries for which there is more than one logical table source that can satisfy the requested set of columns.

For example, you might have user queries that can be fulfilled by both a data warehouse and an OLTP source. Often, access to an operational system is "expensive," while access to a data warehouse is "cheap." In this situation, you can assign a higher priority to the data warehouse to ensure that all queries are fulfilled by the data warehouse if possible.

Note that the priority group of a given logical table source does not always ensure that a particular query will be fulfilled by that source. Priority group assignments are only one of many factors used by the Oracle BI Server to determine which logical table source to select for a given query. However, the logical table source priority is the most significant metric and is considered before any other cost metric.

To assign priority group numbers, rank your logical table sources in numeric order, with 0 being the highest-priority source. You can assign the same number to multiple sources. For example, you can have two logical table sources in priority group 0, two logical table sources in priority group 1, and so on. Often, only two priority groups are necessary (0 and 1).

Assigning priority groups is optional. All logical table sources are set to priority 0 by default. It is important that you do not use priority groups as a method fine tuning the choice of logical table sources used to answer queries. Oracle BI server tries to automatically use the most optimal logical table sources, but only within the same

priority group. When you set a different priority group to each logical table source, it might cause Oracle BI Server to use suboptimal logical table sources.

In some situations, you might want to allow users to reverse the normal logical table source priority ranking at query time. To accomplish this, you can use a combination of session variables and request variables with logical table source priority groups. This feature provides a way to dynamically select a source at run time, depending on user preference.

To enable this dynamic selection, you must first create the `REVERSIBLE_LTS_PRIORITY_SA_VEC` session variable in the repository. Create this variable as a string vector session variable that uses a row-wise session initialization block. `REVERSIBLE_LTS_PRIORITY_SA_VEC` should list the subject areas for which you want to allow users to reverse the logical table source priority ranking. You must define this variable to enable priority ranking reversal.

After you have defined the set of subject areas for which you want to allow priority ranking reversal, users can include the request variable `REVERSE_LTS_PRIORITY` with their queries to reverse the logical table source priority ranking. This request variable can be set to 1 to reverse the logical table source priority, or 0 to keep the normal logical table source priority.

As an alternative to using a request variable at query time, you can also define a predetermined set of subject areas for which the logical table source priority should be permanently reversed. To do this, create the session variable `REVERSED_LTS_PRIORITY_SA_VEC` in the repository. Create this variable as a string vector session variable that uses a row-wise session initialization block. `REVERSED_LTS_PRIORITY_SA_VEC` should list the subject areas for which you want the logical table source priority to be permanently reversed.

See "Creating Session Variables" for more information about how to define session variables in the Administration Tool.

Example of REVERSIBLE_LTS_PRIORITY_SA_VEC

You could create a table called SA_TABLE that contains two columns: SUBJECT_AREA_NAME and REVERSIBLE. This table could contain rows mapping subject area names to their reversible values (1 or 0), as follows:

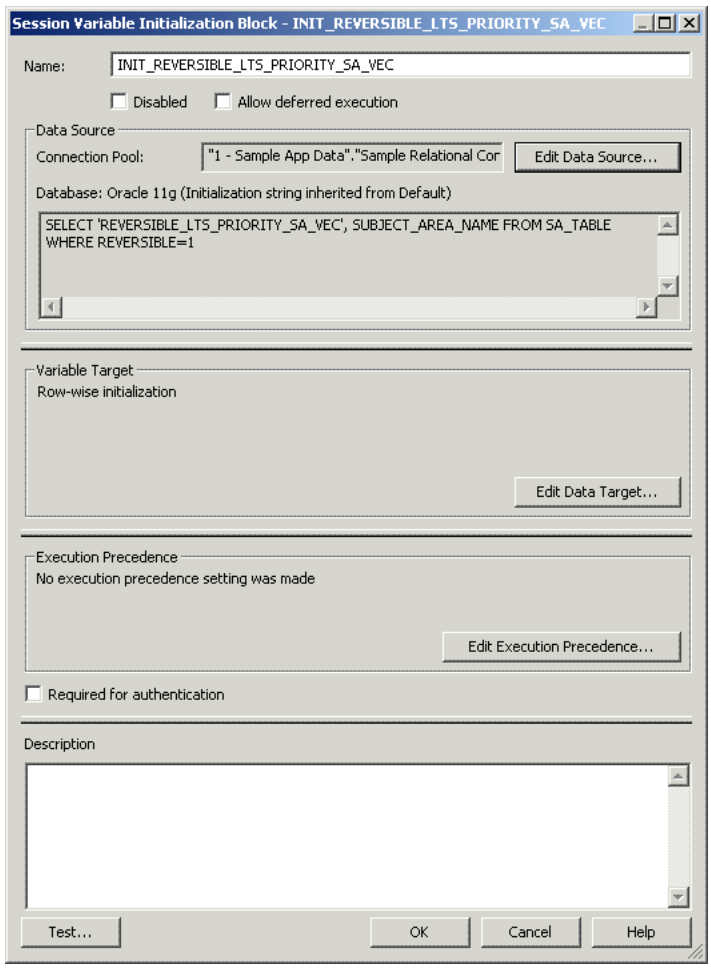
SUBJECT_AREA_NAME	REVERSIBLE
my_sa_1	1
my_sa_2	0

Then, you would create a string vector session variable called REVERSIBLE_LTS_PRIORITY_SA_VEC with a row-wise session initialization block. The initialization string for this initialization block could be similar to the following:

```
SELECT 'REVERSIBLE_LTS_PRIORITY_SA_VEC', SUBJECT_AREA_NAME FROM SA_TABLE
WHERE REVERSIBLE=1
```

Figure 11–1 shows the Session Variable Initialization Block dialog for this example.

Figure 11–1 Session Variable Initialization Block Dialog for REVERSIBLE_LTS_PRIORITY_SA_VEC Example



Defining Physical to Logical Table Source Mappings and Creating Calculated Items

Use the Column Mapping tab of the Logical Table Source dialog to map logical columns to physical columns. The physical to logical mapping can also be used to specify transformations that occur between the Physical layer and the Business Model and Mapping layer. The transformations can be simple, such as changing an integer data type to a character, or more complex, such as applying a formula to find a percentage of sales per unit of population. Applying these transformations is typically referred to as creating calculated items.

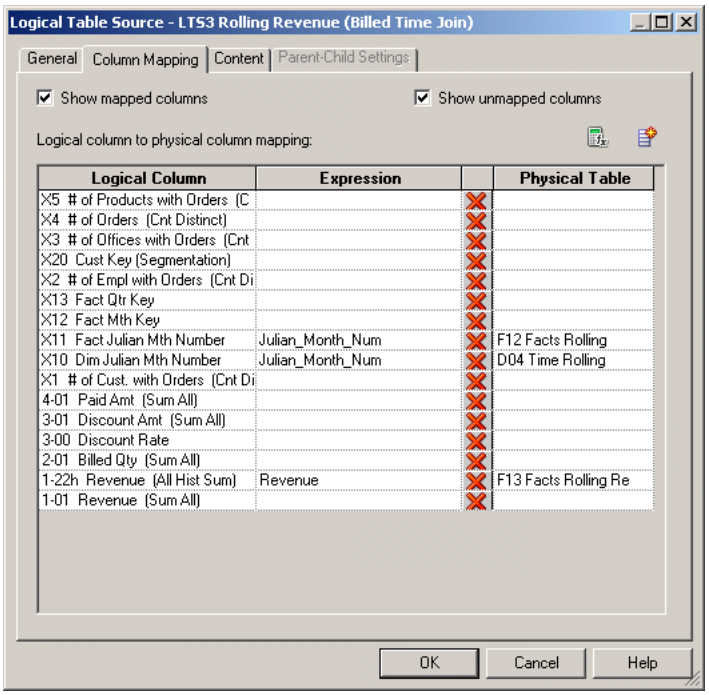
The data type of a logical column is determined by its logical table source mappings. For example, if a logical column has one physical source with a data type of VARCHAR(50) not-nullable, and another physical source with a data type of VARCHAR(20), nullable, then the data type of the logical column is VARCHAR(50) nullable. This final type is called a promoted type. Because of the rules governing logical table source mappings, you cannot map physical sources with data types that cannot be promoted (such as an INT with a VARCHAR).

To map logical columns to physical columns:

1. In the Business Model and Mapping layer of the Administration Tool, double-click a logical table source.
2. In the Logical Table Source dialog, click the Column Mapping tab.
3. In the Column Mapping tab, maximize or enlarge the dialog to show all the contents, as shown in Figure 11–2.

   In the Column Mapping tab, in the **Logical column to physical column mapping** area, you can sort the rows (toggle among ascending order, descending order, and then restore original order) by clicking a column heading.

Figure 11–2 Column Mapping Tab of Logical Table Source Dialog



4. In the **Physical Table** column, select the table that contains the column you want to map.

   When you select a cell in the Physical Table column, a list appears. It contains a list of tables currently included in this logical table source.

5. In the **Expression** column, select the physical column corresponding to each logical column.

   When you select a cell in the **Expression** column, a list appears. It contains a list of physical columns currently included in this logical table source.

6. To open Expression Builder, click the **Expression Builder** button.

   All columns used in creating physical expressions must be in tables included in the logical table source. You cannot create expressions involving columns in tables outside the source.

   You can use Expression Builder to create calculated items, in which formulas are applied pre-aggregation. For example, to create the measure "tons sold" using the columns units_sold and unit_weight, you apply a pre-aggregation formula (fact.units_sold*product.unit_weight), and then apply the aggregation rule `SUM` in the measure object. Another example is using `CAST` to transform a column of type

`TIMESTAMP` to type `DATE` for faster display in Answers and other clients (for example, `CAST("DB"."TABLE"."COL" AS DATE)`).

You can also conform sources by creating expressions that perform transformations on physical data. For example, you can use the `CAST` function to transform a column with a character data type to an integer data type, to match data coming from a second logical table source. Other examples include using `CONCATENATE` or math functions to make similar transformations on physical data.

See "Creating Derived Columns" for calculations that need to occur post-aggregation.

7. To remove a column mapping, click the **Delete** button. You might need to scroll to the right to locate the **Delete** button.
8. After you map the appropriate columns, click **OK**.

Unmapping a Logical Column from Its Source

In the Logical Column dialog, the Column Source tab contains information about the logical column. You can edit the logical table sources from which the column derives its data, or unmap it from its sources.

To unmap a logical column from its source:

1. In the Business Model and Mapping layer of the Administration Tool, double-click a logical column.
2. In the Logical Column dialog, click the Column Source tab.
3. In the **Logical Table Source** list, select a source and click **Unmap**.
4. Click **OK**.

Defining Content of Logical Table Sources

To use a source correctly, the Oracle BI Server has to know what each source contains in terms of the business model. Therefore, you need to define aggregation content for each logical table source of a fact table. The aggregation content rule defines at what level of granularity the data is stored in this fact table. For each dimension that relates to this fact logical table, define the level of granularity, making sure that every related dimension is defined. See "Setting Up Aggregate Navigation by Creating Sources for Aggregated Fact Data" for more information.

If a logical table is sourced from a set of fragments, it is not required that every individual fragment maps the same set of columns. However, the server returns different answers depending on how columns are mapped.

- If all the fragments of a logical table map the same set of columns, than the set of fragmented sources is considered to be the whole universe of logical table sources for the logical table. This means that measure aggregations can be calculated based on the set of fragments.
- If the set of mapped columns differ across the fragments, than the Oracle BI Server assumes that it does not have the whole universe of fragments, and therefore it would be incorrect to calculate aggregate rollups (since some fragments are missing). In this case, the server returns NULL as measure aggregates.

Note: Oracle highly recommends that all the fragments map the same set of columns.

Use the Content tab of the Logical Table Source dialog to define any aggregate table content definitions, fragmented table definitions for the source, and WHERE clauses (if you want to limit the number of rows returned). See "Setting Up Fragmentation Content for Aggregate Navigation" for additional information.

Verifying that Joins Exist from Dimension Tables to Fact Table

This source content information tells the Oracle BI Server what it needs to know to send queries to the appropriate physical aggregate fact tables, joined to and constrained by values in the appropriate physical aggregate dimension tables. Be sure that joins exist between the aggregate fact tables and the aggregate dimension tables in the Physical layer.

One recommended way to verify joins is to select a fact logical table and open a Business Model Diagram (Selected Tables and Direct Joins). Only the dimension logical tables that are directly joined to this fact logical table appear in the diagram. It does not show dimension tables if the same physical table is used in logical fact and dimension sources.

Figure 11–3 shows an example of how the Fact - Assess fact logical table appears in a Business Model Diagram (Selected Tables and Direct Joins) view.

Figure 11–3 Business Model Diagram of Direct Joins for a Fact Table

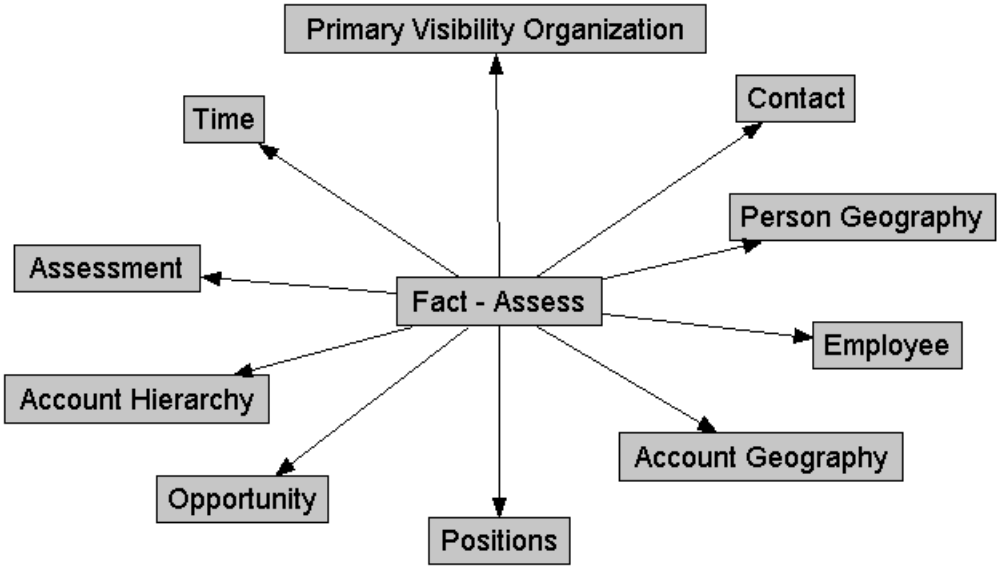


Table 11–1 contains a list of the logical level for each dimension table that is directly joined the Fact - Assess fact table shown in Figure 11–3.

Table 11–1 Dimension and Logical Level as Shown in Content Tab

Dimension	Logical Level
Account Geography	Postal Code Detail
Person Geography	Postal Code Detail
Time	Day Detail
Account Organization	Account Detail
Opportunity	Opty Detail

Table 11–1 (Cont.) Dimension and Logical Level as Shown in Content Tab

Dimension	Logical Level
Primary Visibility Organization	Detail
Employee	Detail
Assessment	Detail
Contact (W_PERSON_D)	Detail
FINS Time	Day
Positions	Details

To create logical table source content definitions:

1. In the Business Model and Mapping layer of the Administration Tool, double-click a logical table source.
2. In the Logical Table Source dialog, click the Content tab and perform the following steps using Table 11–2 as a guide.
3. If a logical source is an aggregate table and you have defined logical dimensions, select **Logical Level** from the **Aggregation content, group-by** list. Then, in the **Logical Level** list, select the appropriate level for each logical dimension table to which the logical fact table is joined.

   You should specify a logical level for each dimension, unless you are specifying the Grand Total level. Dimensions with no level specified are interpreted as being at the most detailed level.

   Caution: Although you have the option to specify aggregate content by logical level or column, it is recommended that you use logical levels exclusively. If you must define content by columns, do the following:

   1. Select **Column** from the **Aggregation content, group-by** list.
   2. In the **Table** pane, select each logical dimension table that defines the aggregation level of the source.
   3. In the **Column** pane, select the logical column for each dimension that defines how the aggregations were grouped.

      When there are multiple logical columns that could be used, select the one that maps to the key of the source physical table. For example, if data has been aggregated to the Region logical level, pick the logical column that maps to the key of the Region table.

   Do not mix aggregation by logical level and column in the same business model.

   ---

4. To specify fragmented table definitions for the source, use the **Fragmentation content** box to describe the range of values included in the source when a source represents a portion of the data at a given level of aggregation.

   You can type the formula directly into the box, or click the Expression Builder button to the right of the box. In the Expression Builder for Fragmentation Content, you can specify content in terms of existing logical columns. See "Setting Up Fragmentation Content for Aggregate Navigation" for additional information.

5. (Optional) Select **This source should be combined with other sources at this level**.

   Choose this option only if all fragments on this level are disjointed. Consider the following examples:

   - **Example 1** - Suppose you have two fragments: all sales including current year and current year sales with the fragmentation predicate set to year = 2015. In this case you *should not select* the **This source should be combined with other sources at this level** option because the two fragments overlap. Oracle BI Server can use any single fragment based on query predicate or fragmentation predicate compatibility.
   - **Example 2** - Suppose you have two fragments: sales for year 2000 and before (according to the fragmentation predicate) and sales for year 2001 and after (according to the fragmentation predicate). In this case *you should select* the **This source should be combined with other sources at this level** option because the fragments do not overlap. In this case Oracle BI server will union all the logical table sources on this level that cannot be disqualified based on query predicate or fragmentation predicate compatibility.

6. (Optional) To limit the number of rows the source uses in the resultant table, specify `WHERE` clause filters in the box labeled **Use this "WHERE clause" filter to limit rows returned (exclude the "WHERE")**. You can enter the `WHERE` clause directly, or you can click the Expression Builder button to open the Expression Builder, create the `WHERE` clause, and click **OK**.

   See "About WHERE Clause Filters" for more information.

7. If the values for the source are unique, select the option **Select distinct values**.

Table 11–2 Content Tab Options for Logical Table Source

Options	Description
Aggregation content, group by	How the content is aggregated.

Table 11–2 (Cont.) Content Tab Options for Logical Table Source

Options	Description
More button	When you click More, the following options appear: ■ **Copy.** (Available only for fact tables) Copies aggregation content to the Windows clipboard. You can paste the Dimension.Level info into a text editor and use it for searching or for adding to documentation. Note that **Copy** is not available if the expression is empty. ■ **Copy from.** (Available for fact tables and dimension tables) Copies aggregation content from another logical table source in the same business model. You need to specify the source from which to copy the aggregation content. (Multiple business models appear but only the logical table sources from the current business model are selectable.) ■ **Get Levels.** (Available only for fact tables) Changes aggregation content. If joins do not exist between fact table sources and dimension table sources (for example, if the same physical table is in both sources), the aggregation content determined by the Administration Tool does not include the aggregation content of this dimension. ■ **Check Levels.** (Available only for fact tables) Checks the aggregation content of logical fact table sources (not dimension table sources). The information returned depends on the existence of dimensions and hierarchies with logical levels and level keys, and physical joins between tables in dimension table sources and the tables in the fact table source. (If the same tables exist in the fact and dimension sources and there are no physical joins between tables in the sources, Check Levels does not include the aggregation content of this dimension.)
Fragmentation content	A description of the contents of a data source in business model terms. Data is fragmented when information at the same level of aggregation is split into multiple tables depending on the values of the data. A common situation would be to have data fragmented by time period. See "Setting Up Fragmentation Content for Aggregate Navigation" for additional information.
This source should be combined with other sources at this level	Select this option when data sources at the same level of aggregation do not contain overlapping information. In this situation, all sources must be combined to get a complete picture of information at this level of aggregation.
Select distinct values	Used if the values for the source are unique.

About WHERE Clause Filters

The `WHERE` clause filter is used to constrain the physical tables referenced in the logical table source. If there are no constraints on the aggregate source, leave the `WHERE` clause filter blank.

Each logical table source should contain data at a single intersection of aggregation levels. You would not want to create a source, for example, that had sales data at both the Brand and Manufacturer levels. If the physical tables include data at multiple levels, add an appropriate `WHERE` clause constraint to filter values to a single level.

Any constraints in the `WHERE` clause filter are made on the physical tables in the source.

About Working with Parent-Child Settings in the Logical Table Source

Sometimes, a logical table is part of a dimension with a parent-child hierarchy that is based on relational tables. When this is the case, the logical table includes both a physical source and a source for the parent-child relationship table required for the parent-child hierarchy. Parent-child relationship tables explicitly define the inter-member relationships for parent-child hierarchies.

Typically, logical table sources for parent-child relationship tables are created automatically when you run the scripts created by the Generate Parent-Child Table Wizard. You access this wizard from the Parent-Child Table Settings dialog, available in the dimension object.

Note: The Generate Parent-Child Table Wizard feature is not available from the Logical Table Source dialog. You must go to the dimension object to create scripts to generate the parent-child relationship table.

You can view details for the parent-child relationship table source in the Parent-Child Settings tab of the Logical Table Source dialog. The following information appears in the tab:

- **Parent-Child Table:** Shows the name of the parent-child relationship table on which this source is based.
- **Member Key:** The name of the column in the parent-child relationship table that identifies the member.
- **Parent Key:** The name of the column in the parent-child relationship table that identifies an ancestor of the member.
- **Relationship Distance:** The name of the column in the parent-child relationship table that specifies the number of parent-child hierarchical levels from the member to the ancestor.
- **Leaf Node Identifier:** The name of the column in the parent-child relationship table that indicates if the member is a leaf member (1=Yes, 0=No).

See "Creating Dimensions with Parent-Child Hierarchies" for more information about parent-child relationship tables.

Setting Up Aggregate Navigation by Creating Sources for Aggregated Fact Data

Aggregate tables store precomputed results from measures that have been aggregated over a set of dimensional attributes. Each aggregate table column contains data at a given set of levels. For example, a monthly sales table might contain a precomputed sum of the revenue for each product in each store during each month. You configure this metadata in the Content tab of the Logical Table Source dialog.

When you create a logical table source for an aggregate fact table, you should create corresponding logical dimension table sources at the same levels of aggregation.

You need to have at least one logical dimension table source for each level of aggregation. If the sources at each level already exist, you do not need to create new ones.

For example, you might have a monthly sales fact table containing a precomputed sum of the revenue for each product in each store during each month. You need to have the following three other dimension sources, one for each of the logical dimension tables referenced in the example:

- A source for the Product logical table with one of the following content specifications:
  - By logical level: ProductDimension.ProductLevel
  - By column: Product.Product_Name
- A source for the Store logical table with one of the following content specifications:
  - By logical level: StoreDimension.StoreLevel
  - By column: Store.Store_Name
- A source for the Time logical table with one of the following content specifications:
  - By logical level: TimeDimension.MonthLevel
  - By column: Time.Month

At query time, the Oracle BI Server first determines which sources have enough detail to answer the query. Out of these sources, the Oracle BI Server chooses the most aggregated source to answer the query, because it is assumed to be the fastest. The most aggregated source is the one with the lowest multiplied number of elements. See "Creating Logical Levels in a Dimension" for more information about specifying the number of elements at each level.

Setting Up Fragmentation Content for Aggregate Navigation

When a logical table source does not contain the entire set of data at a given level, you need to specify the portion, or fragment, of the set that it does contain. You describe the content in terms of logical columns in the **Fragmentation content** box in the Content tab of the Logical Table Source dialog.

The examples in this section illustrate techniques and rules for specifying the fragmentation content of sources.

This section contains the following topics:

Specifying Fragmentation Content for Single Column, Value-Based Predicates

The `IN` predicates can be replaced with either an equality predicate or multiple equality predicates separated by the `OR` connective.

Fragment 1:

```
logicalColumn IN <valueList1>
```

Fragment n:

```
logicalColumn IN <valueListN>
```

Specifying Fragmentation Content for Single Column, Range-Based Predicates

Fragment 1:

```
logicalColumn >= valueof(START_VALUE) AND logicalColumn < valueof(MID_VALUE1)
```

Fragment 2:

```
logicalColumn >= valueof(MID_VALUE1) AND logicalColumn < valueof(MID_VALUE2)
```

Fragment n:

```
logicalColumn >= valueof(MID_VALUEN-1) AND logicalColumn < valueof(END_VALUE)
```

Pick your start point, midpoints, and endpoint carefully.

> **Note:** Use >= and < predicates to make sure the fragment content descriptions do not overlap. For each fragment, the upper value must be expressed as <. You will get an error if you use <=. Likewise, you cannot use the `BETWEEN` predicate to describe fragment range content.

The `valueof` referenced here is the value of a repository variable. If you use repository values in your expression, note that the following construct does not work for Fragment 2:

```
logicalColumn >= valueof(MID_VALUE1)+1 AND logicalColumn < valueof(MID_VALUE2)
```

Use another repository variable instead of `valueof(MID_VALUE1)+1`.

The same variables, for example, `valueof(MID_VALUE1)`, do not have to appear in the content of both fragments. You could set another variable, and create statements of the following form:

Fragment 1:

```
logicalColumn >= valueof(START_VALUE) AND logicalColumn < valueof(MID_VALUE1)
```

Fragment 2:

```
logicalColumn >= valueof(MID_VALUE2) AND logicalColumn < valueof(MID_VALUE3)
```

For more information about variables, see Chapter 19.

Specifying Multicolumn Content Descriptions

An arbitrary number of predicates on different columns can be included in each content filter. Each column predicate can be value-based or range-based.

Fragment 1:

```
<logicalColumn1 predicate> AND <logicalColumn2 predicate > ... AND <logicalColumnM
predicate>
```

Fragment n:

```
<logicalColumn1 predicate> AND <logicalColumn2 predicate > ... AND <logicalColumnM
predicate>
```

Ideally, all fragments will have predicates on the same M columns. If there is no predicate constraint on a logical column, the Oracle BI Server assumes that the fragment contains data for all values in that logical column. See "Specifying Parallel Content Descriptions" for exceptions using the `OR` predicate.

Specifying Parallel Content Descriptions

Unfortunately, the preceding techniques are still not sufficient to handle dates because of the multiple hierarchical relationships across logical columns, such as year > year month > date; month > year month > date. For example, consider fragments delineated by different points in time, such as year and month. Constraining sufficiently far back on year should be enough to drive the selection of just the historical fragment. The parallel `OR` technique supports this, as shown in the next example. This example assumes that the snapshot month was April 1, 12:00 a.m. in the year 1999.

Fragment 1 (Historical):

```
EnterpriseModel.Period."Day" < VALUEOF("Snapshot Date") OR
EnterpriseModel.Period.MonthCode < VALUEOF("Snapshot Year Month") OR
EnterpriseModel.Period."Year" < VALUEOF("Snapshot Year") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Month in Year" < VALUEOF("Snapshot Month") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Monthname" IN ('Mar', 'Feb', 'Jan')
```

Fragment 2 (Current):

```
EnterpriseModel.Period."Day" >= VALUEOF("Snapshot Date") OR
EnterpriseModel.Period.MonthCode >= VALUEOF("Snapshot Year Month") OR
EnterpriseModel.Period."Year" > VALUEOF("Snapshot Year") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Month in Year" >= VALUEOF("Snapshot Month") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Monthname" IN ('Dec', 'Nov', 'Oct', 'Sep', 'Aug', 'Jul',
  'Jun', '', 'Apr')
```

If the logical model does not go down to the date level of detail, then omit the predicate on `EnterpriseModel.Period."Day"` in the preceding example.

Notice the use of the `OR` connective to support parallel content description tracks.

Examples of Parallel Content Descriptions In this section, the Track *n* labels in the examples are shown to help relate the examples to the discussion that follows. You would not include these labels in the actual fragmentation content statement.

Example 11–1 Fragment 1 (Historical)

```
Track 1  EnterpriseModel.Period."Day" < VALUEOF("Snapshot Date") OR
Track 2  EnterpriseModel.Period.MonthCode < VALUEOF("Snapshot Year Month") OR
Track 3  EnterpriseModel.Period."Year" < VALUEOF("Snapshot Year") OR
Track 4  EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Month in Year" < VALUEOF("Snapshot Month") OR
Track 5  EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Monthname" IN ('Mar', 'Feb', 'Jan')
```

For example, consider the first track on `EnterpriseModel.Period."Day."` In the historical fragment, the < predicate tells the Oracle BI Server that any queries that constrain on Day before the Snapshot Date fall within the historical fragment. Conversely, the >= predicate in the current fragment on Day indicates that the current fragment does not contain data before the Snapshot Date.

The second track on `MonthCode` (for example, 199912) is similar to Day. It uses the < and >= predicates, as there is a nonoverlapping delineation on month (because the snapshot date is April 1). The key rule to remember is that each additional parallel track must reference a different column set. Common columns can be used, but the

overall column set must be unique. The Oracle BI Server uses the column set to select the most appropriate track.

The third track on `Year` (< in the historical fragment and > in the current fragment) tells the Oracle BI Server that optimal (single) fragment selections can be made on queries that just constrain on year. For example, a logical query on Year IN (1997, 1998) should only hit the historical fragment. Likewise, a query on Year = 2000 should only hit the current fragment. However, a query that hits the year 1999 cannot be answered by the content described in this track, and therefore hits both fragments, unless additional information can be found in subsequent tracks.

The fourth track describes the fragment set for Year and Month in Year (month integer). Notice the use of the multicolumn content description technique, described previously. Notice the use of < and >= predicates, as there is no ambiguity or overlap for these two columns.

The fifth track describes fragment content in terms of Year and Monthname. It uses the value-based `IN` predicate technique.

As an embellishment, suppose the snapshot date fell on a specific day within a month: therefore, multicolumn content descriptions on just year and month would overlap on the specific snapshot month. To specify this ambiguity, <= and >= predicates are used.

Fragment 1 (Historical):

```
EnterpriseModel.Period."Day" < VALUEOF("Snapshot Date") OR
EnterpriseModel.Period.MonthCode <= VALUEOF("Snapshot Year Month") OR
EnterpriseModel.Period."Year" < VALUEOF("Snapshot Year") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Month in Year" <= VALUEOF("Snapshot Month") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Monthname" IN ('Apr', 'Mar', 'Feb', 'Jan')
```

Fragment 2 (Current):

```
EnterpriseModel.Period."Day" >= VALUEOF("Snapshot Date") OR
EnterpriseModel.Period.MonthCode >= VALUEOF("Snapshot Year Month") OR
EnterpriseModel.Period."Year" > VALUEOF("Snapshot Year") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Month in Year" >= VALUEOF("Snapshot Month") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Monthname" IN ('Dec', 'Nov', 'Oct', 'Sep', 'Aug', 'Jul',
  'Jun', '', 'Apr')
```

Specifying Unbalanced Parallel Content Descriptions

In an order entry application, time-based fragmentation between historical and current fragments is typically insufficient. For example, records might still be volatile, even though they are historical records entered into the database before the snapshot date.

Assume, in the following example, that open orders can be directly updated by the application until the order is shipped or canceled. After the order has shipped, however, the only change that can be made to the order is to type a separate compensating return order transaction.

There are two parallel tracks in the following content descriptions. The first track uses the multicolumn, parallel track techniques described in the preceding section. Notice the parentheses nesting the parallel calendar descriptions within the Shipped-or-Canceled order status multicolumn content description.

The second parallel track is present only in the Current fragment and specifies that all Open records are in the Current fragment only.

Fragment 1 (Historical):

```
Marketing."Order Status"."Order Status" IN ('Shipped', 'Canceled') AND
  Marketing.Calendar."Calendar Date" <= VALUEOF("Snapshot Date") OR
Marketing.Calendar."Year" <= VALUEOF("Snapshot Year") OR
Marketing.Calendar."Year Month" <= VALUEOF("Snapshot Year Month")
```

Fragment 2 (Current):

```
Marketing."Order Status"."Order Status" IN ('Shipped', 'Canceled') AND
  Marketing.Calendar."Calendar Date" > VALUEOF("Snapshot Date") OR
Marketing.Calendar."Year" >= VALUEOF("Snapshot Year") OR
Marketing.Calendar."Year Month" >= VALUEOF("Snapshot Year Month") OR
Marketing."Order Status"."Order Status" = 'Open'
```

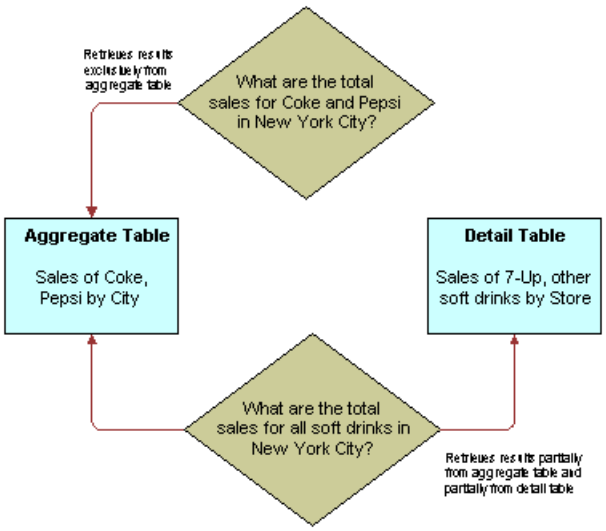
The overlapping Year and Month descriptions in the two fragments do not cause a problem, as overlap is permissible when there are parallel tracks. The rule is that at least one of the tracks has to be nonoverlapping. The other tracks can have overlap.

Specifying Fragmentation Content for Aggregate Table Fragments

Information at a given level of aggregation is sometimes stored in multiple physical tables. When individual sources at a given level contain information for a portion or fragment of the domain, the Oracle BI Server needs to know the content of the sources in order to pick the appropriate source for the query.

For example, suppose you have a database that tracks the sales of soft drinks in all stores. The detail level of data is at the store level. Aggregate information, as described in Figure 11–4, is stored at the city level for the sales of Coke and Pepsi, but there is no aggregate information for the sales of 7-Up or any other of the sodas.

Figure 11–4 Aggregating Information



The goal of this type of configuration is to maximize the use of the aggregate table. If a query asks for sales figures for Coke and Pepsi, the data should be returned from the aggregate table. If a query asks for sales figures for all soft drinks, the aggregate table should be used for Coke and Pepsi and the detail data for the other brands.

The Oracle BI Server handles this type of partial aggregate navigation. To configure a repository to use aggregate fragments for queries whose domain spans multiple fragments, you need to define the entire domain for each level of aggregate data, even if you must configure an aggregate fragment as being based on a less summarized physical source.

This section contains the following topics:

Specifying the Aggregate Table Content

You configure the aggregate table navigation in the logical table source mappings. In the soft drink example, the aggregate table contains data for Coke and Pepsi sales at the city level. Its Aggregate content specification (in the Content tab of the Logical Table Source window) is similar to the following:

Group by logical level:

```
GeographyDim. CityLevel, ProductDim.ProductLevel
```

Its Fragmentation content specification (also in the Content tab of the Logical Table Source dialog) is similar to the following:

```
SoftDrinks.Products.Product IN ('Coke', 'Pepsi')
```

This content specification tells the Oracle BI Server that the source table has data at the city and product level for two of the products. Additionally, because this source is a fragment of the data at this level, you must select **This source should be combined with other sources at this level**, in the Content tab of the Logical Table Source dialog, to indicate that the source combines with other sources at the same level.

Defining a Physical Layer Table with a Select Statement to Complete the Domain

The data for the rest of the domain (the other types of sodas) is all stored at the store level. To define the entire domain at the aggregate level (city and product, in this example), you need to have a source that contains the rest of the domain at this level. Because the data at the store level is at a lower (that is, more detailed) level than at the city level, it is possible to calculate the city and product level detail from the store and product detail by adding up the product sales data of all of the stores in a city. This can be done in a query involving the store and product level table.

One way to do this is to define a table in the Physical layer with a Select statement that returns the store level calculations. To define the table, create a table in the Physical layer by right-clicking the physical schema object that the `SELECT` statement will be querying and selecting **New Physical Table**. Choose **Select** from the **Table Type** list, and type the SQL statement in the **Default Initialization String** box.

The SQL statement must define a virtual table that completes the domain at the level of the other aggregate tables. In this case, there is one existing aggregate table, and it contains data for Coke and Pepsi by city. Therefore, the SQL statement has to return all of the data at the city level, except for the Coke and Pepsi data.

Specifying the SQL Virtual Table Content

Next, create a new logical table source for the Sales column that covers the remainder of the domain at the city and product level. This source contains the virtual table created in the previous section. Map the Dollars logical column to the USDollars physical column in this virtual table.

The Aggregate content specification (in the Content tab of the Logical Table Source dialog) for this source is:

Group by logical level:

```
GeographyDim.CityLevel, ProductDim.ProductLevel
```

This tells the Oracle BI Server this source has data at the city and product level.

The Fragmentation content specification might be:

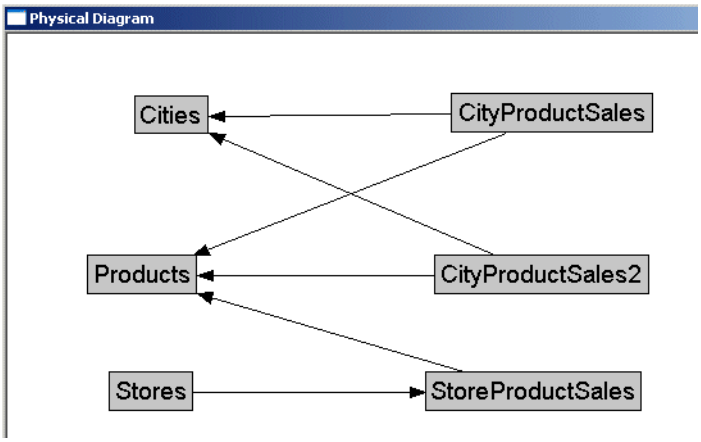
```
SoftDrinks.Products.Product = '7-Up'
```

Additionally, because it combines with the aggregate table containing the Coke and Pepsi data at the city and product level to complete the domain, you need to select the option in the Content tab of the Logical Table Source dialog indicating that the source is combined with other sources at the same level.

Creating Physical Joins for the Virtual Table

Construct the correct physical joins for the virtual table. Notice that CityProductSales2 joins to the Cities and Products tables in Figure 11–5.

Figure 11–5 Example Physical Joins



In this example, the two sources comprise the whole domain for soda sales. A domain can have many sources. The sources have to all follow the rule that each level must contain sources that, when combined, comprise the whole domain of values at that level. Setting up the entire domain for each level helps ensure that queries asking for Coke, Pepsi, and 7-Up do not leave out 7-Up. It also helps ensure that queries requesting information that has been precomputed and stored in aggregate tables can retrieve that information from the aggregate tables, even if the query requests other information that is not stored in the aggregate tables.

12

Creating and Maintaining the Presentation Layer

This chapter explains how to use the Oracle BI Administration Tool to create, edit, and maintain objects in the Presentation layer of the Oracle BI repository.

This chapter contains the following topics:

About the Presentation Layer

The Presentation layer provides a way to present customized, secure, role-based views of a business model to users. Role-based views provide object security and also provide a way to hide some of the complexity of the business model.

The Presentation layer also provides some of the functionality of the metadata model, such as the ability to set an implicit fact column.

Presentation layer views are called subject areas (formerly called presentation catalogs). You can have a subject area that is identical to your business model, or you can provide smaller, role-based subject areas that show a single subject, or that support a particular business role. Create subject areas that help you organize your content in a way that makes sense for your users.

Subject areas in the Presentation layer appear as catalogs to client tools that use the Oracle BI Server as an ODBC data source. Subject areas contain presentation tables, columns, hierarchies, and levels.

Even though the Logical SQL requests from Answers and other clients query the presentation tables and columns, the real logic for entities, relationships, joins, and so on is in the Business Model and Mapping layer. The primary function of the Presentation layer is to provide custom names, dictionary entries, organization, and security for different groups of users.

Creating and Customizing the Presentation Layer

After you have created the Business Model and Mapping layer, you can drag and drop entire business models to the Presentation layer in the Administration Tool to create subject areas.

Alternatively, you can create subject areas and other Presentation layer objects manually.

This section contains the following topics:

About Creating Subject Areas

There are several ways to create subject areas in the Presentation layer. The recommended method is to drag and drop a business model from the Business Model and Mapping layer to the Presentation layer, and then modify the Presentation layer based on what you want users to see. You can move columns between presentation tables, remove columns that do not need to be seen by the users, or even present all of the data in a single presentation table. You can create presentation tables to organize and categorize measures in a way that makes sense to your users.

You can also duplicate an existing subject area and its corresponding business model. See "Duplicating a Business Model and Subject Area" for more information.

Although each subject area must be populated with contents from a single business model, you can create multiple subject areas for one business model. For very large business models, you may want to do this to help users work with the content. Users in Oracle BI Answers can create queries that span multiple subject areas, as long as the subject areas correspond to the same business model.

There are many ways to create multiple subject areas from a single business model. One method is to drag a particular business model to the Presentation layer multiple times, then edit the properties or objects of the resulting subject areas as needed.

For example, if you have a business model called `ABC` that contains the `Geography` and `Products` dimensions, you can drag it to the Presentation layer twice. Two subject areas are created, with the default names `ABC` and `ABC#1`. You can then edit the subject areas as follows:

- Rename the `ABC` subject area to `DEF`, then delete the `Geography` presentation hierarchy
- Rename the `ABC#1` subject area to `XYZ`, then delete the `Products` presentation hierarchy

Users in Oracle BI Answers can then run queries that span both the `DEF` subject area (containing the `Products` hierarchy), and the `XYZ` subject area (containing the `Geography` hierarchy).

Note: Typically, when you query a single subject area, all the columns exposed in that subject area are compatible with all the dimensions exposed in the same subject area. However, when you combine columns and dimensions from multiple subject areas, you must ensure that you do not include combinations of columns and dimensions that are incompatible with one another.

For example, a column in one subject area might not be dimensioned by Project. If columns from the Project dimension from another subject area are added to the request along with columns that are not dimensioned by Project, then the query might fail to return results, or cause the Oracle BI Server error "No fact table exists at the requested level of detail: XXXX."

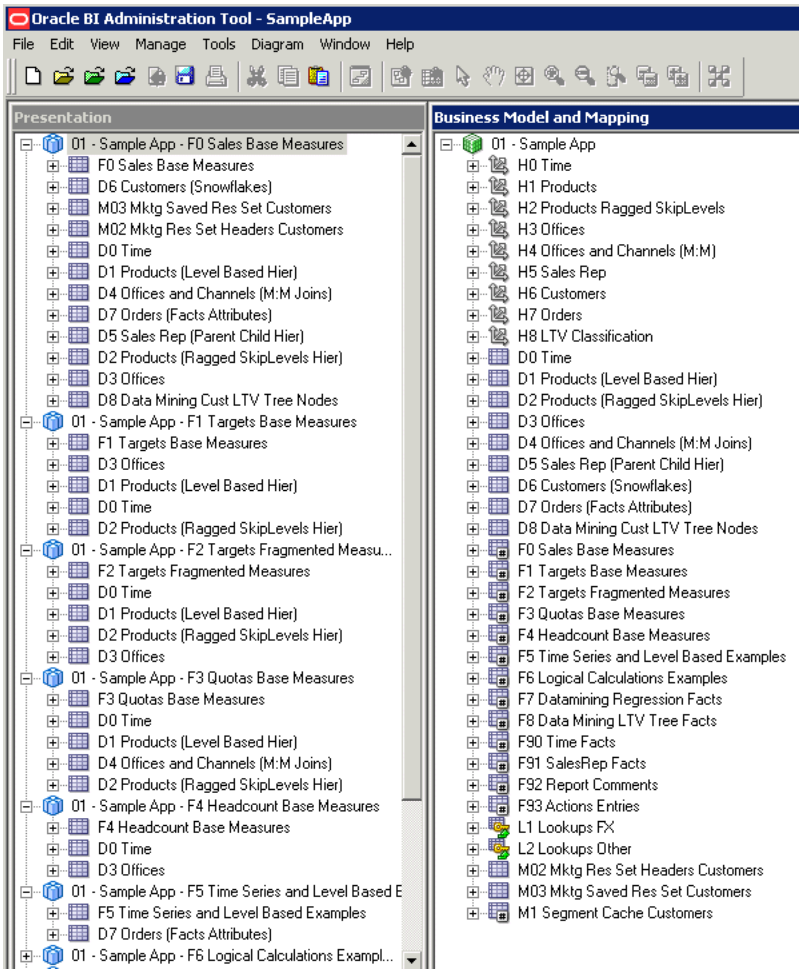
Automatically Creating Subject Areas Based on Logical Stars and Snowflakes

You can automatically create one subject area for each logical star or logical snowflake in your business model. Logical stars and logical snowflakes are both composed of a centralized fact table connected to multiple dimension tables. This feature provides another way to create multiple subject areas from a single business model.

To create a subject area for each fact table that is part of a logical star or snowflake, right-click the business model and select **Create Subject Areas for Logical Stars and Snowflakes**. The new subject areas are automatically created, each containing a fact table and only the dimension tables with which it is associated. This option is available for any business model that contains logical stars or logical snowflakes.

For example, if you choose this option for the SampleApp business model with nine fact tables, nine corresponding subject areas are created, each with one fact table and its associated dimension tables. Subject areas are also created for lookup tables. Figure 12–1 shows exactly how the logical fact tables and dimension tables are modeled in the Presentation layer.

Figure 12–1 Creating Subject Areas for Logical Stars and Snowflakes in SampleApp



About Removing Unneeded or Unwanted Columns

One important reason to use a custom Presentation layer is to make the schema as easy to use and understand as possible. Therefore, users should not be able to view columns that have no meaning to them.

The following columns are examples of columns that you might want to remove from the Presentation layer:

- Key columns that have no business meaning
- Columns that users do not need to view (for example, codes, when text descriptions exist)
- Columns that users are not authorized to see

Renaming Presentation Columns to User-Friendly Names

By default, presentation columns have the same name as the corresponding logical column in the Business Model and Mapping layer. It is recommended to keep presentation column names and their source logical column names synchronized to reduce maintenance.

To do this, ensure that **Use Logical Column Name** is selected in the Presentation Column dialog.

In some cases, however, you may want a different presentation column name to be shown to users. To do this, change the name of the presentation column in the Presentation Column dialog.

When you change the name of a presentation column, an alias is automatically created for the old name, so compatibility to the old name remains. See "Creating Aliases (Synonyms) for Presentation Layer Objects" for more information about aliases.

Note that you cannot rename a Presentation layer object to a name that is already in use as an alias for an object of the same type.

Exporting Logical Keys in the Subject Area

For each subject area in the Presentation layer, you can decide whether to export any logical keys as key columns to tools that access it. Exporting logical keys is irrelevant to users of Oracle BI Presentation Services, but it may be advantageous for some query and reporting tools.

If you decide to export logical keys, make sure that the logical key columns exist in the table folders. In this situation, your business model should use logical key/foreign key joins.

When you select the option **Export logical keys** in the Subject Area dialog, any columns in the Presentation layer that are key columns in the Business Model and Mapping layer are listed as key columns to any ODBC client. This is the default selection. In most situations, this option should be selected.

Note: If you are using a tool that issues parameterized SQL queries, such as Microsoft Access, do not select the option **Export logical keys**. This stops the tool from issuing parameterized queries.

Setting an Implicit Fact Column in the Subject Area

For each subject area in the Presentation layer, you can set an implicit fact column. The implicit fact column is added to a query when it contains columns from two or more dimension tables and no measures.

The column is not visible in the results. It is used to specify a default join path between dimension tables when there are several possible alternatives or contexts.

Maintaining the Presentation Layer

There is no automatic way to synchronize all changes between the Business Model and Mapping layer and the Presentation layer. For example, if you add logical columns to an existing logical table, or edit existing columns, you must manually update the corresponding Presentation layer objects.

However, the Administration Tool can automatically synchronize the name of presentation columns with their corresponding logical column names. To take advantage of this feature, ensure that **Use Logical Column Name** is selected in the Presentation Column dialog.

In some cases, if there are many changes to a logical table or even to an entire business model, it is easiest to delete the corresponding presentation table or subject area, and then and drag and drop the updated logical objects to the Presentation layer. For this reason, it is best to wait until the Business Model and Mapping layer is relatively stable before adding customizations in the Presentation layer.

Working with Subject Areas

In the Presentation layer, subject areas enable you to show different views of a business model to different sets of users. Subject areas have to be populated with contents from a single business model. They cannot span business models.

Typically, subject areas are created automatically by dragging and dropping business models from the logical layer.

To edit the properties of a subject area:

1. In the Presentation layer, double-click a subject area. The Subject Area dialog is displayed.
2. In the General tab, you can change the name for the subject area. Note that aliases are created automatically whenever presentation objects are renamed, so that any queries using the original name do not break.

   Also, a subject area cannot have the same name as any of its child presentation tables. For example, you cannot have a subject area called Customer that has a Customer table within it.

   Note: You must enable the **Edit presentation names** Administration Tool option before you can edit the subject area's name.

3. To set permissions for this subject area, click **Permissions**. See "Setting Permissions for Presentation Layer Objects" for more information.
4. Select **Custom display name** to dynamically display a name based on a session variable and to edit the **Translation Key** field. Select **Custom description** to dynamically display a custom description based on a session variable.

   These options are used typically for localization purposes. When you externalize strings in the Presentation layer and run the Externalize Strings utility, the results contain the session variable information and the translation key.

   See "Localizing Oracle Business Intelligence" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about localization.
5. The **Business model** list displays the business model for this subject area.
6. To expose the logical keys to other applications, select the option **Export logical keys**.

   In most situations, this option should be selected. Many client tools differentiate between key and nonkey columns, and the option **Export logical keys** provides client tools access to the key column metadata. Any join conditions the client tool adds to the query, however, are ignored, because the Oracle BI Server uses the joins defined in the repository.

   Note: If you are using a tool that issues parameterized SQL queries, such as Microsoft Access, do not select the **Export logical keys** option. Not exporting logical keys stops the tool from issuing parameterized queries.

7. Optionally, you can set an **Implicit Fact Column**. This column is added to a query when it contains columns from two or more dimension tables and no measures.

The column is not visible in the results. It is used to specify a default join path between dimension tables when there are several possible alternatives or contexts.

8. Optionally, you can specify an expression in the **Hide object if** field that controls whether this subject area is visible in the Subject Area Tree in Answers and BI Composer. Leave this field blank (the default) to show the object. See "Controlling Presentation Object Visibility" for more information.
9. Optionally, type a description. This description appears in a mouse-over tooltip for the subject area in Oracle BI Answers.
10. In the Presentation Tables tab, you can add, remove, edit, or reorder the presentation tables for this subject area.
11. Use the Aliases tab to specify or delete aliases for this subject area. See "Creating Aliases (Synonyms) for Presentation Layer Objects" for more information about aliases.
12. Click **OK**.

Working with Presentation Tables and Columns

Presentation tables and presentation columns appear as folders and columns in Oracle BI Answers. You can customize presentation tables and presentation columns to help users craft queries based on their business needs.

This section contains the following topics:

- Creating and Managing Presentation Tables
- Creating and Managing Presentation Columns
- Nesting Folders in Answers and BI Composer

Creating and Managing Presentation Tables

You can use presentation tables to organize columns into categories that make sense to the user community. A presentation table can contain columns from one or more logical tables. The names and object properties of the presentation tables are independent of the logical table properties.

Typically, presentation tables are created automatically by dragging and dropping logical tables from the logical layer.

To edit the properties of a presentation table:

1. In the Presentation layer, double-click a presentation table. The Presentation Table dialog appears.
2. In the General tab, you can change the name for the presentation table. Note that aliases are created automatically whenever presentation objects are renamed, so that any queries using the original name do not break.

   Also, a presentation table cannot have the same name as its parent subject area. For example, you cannot have a subject area called Customer that has a Customer table within it.

Note: You must enable the **Edit presentation names** Administration Tool option before you can edit the presentation table's name.

---

3. To set permissions for this presentation table, click **Permissions**. See "Setting Permissions for Presentation Layer Objects" for more information.
4. Select **Custom display name** to dynamically display a name based on a session variable and to edit the **Translation Key** field. Select **Custom description** to dynamically display a custom description based on a session variable.

   These options are used for localization purposes. When you externalize strings in the Presentation layer and run the Externalize Strings utility, the results contain the session variable information and the translation key.

   See "Localizing Oracle Business Intelligence" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about localization.
5. Optionally, you can specify an expression in the **Hide object if** field that controls whether this presentation table is visible in the Subject Area Tree in Answers and BI Composer. Leave this field blank (the default) to show the object. See "Controlling Presentation Object Visibility" for more information.
6. In the Columns tab, you can add, remove, edit, or reorder the presentation columns for this presentation table.
7. In the Hierarchies tab, you can add, remove, edit, or reorder the presentation hierarchies for this presentation table.
8. Use the Aliases tab to specify or delete aliases for this presentation table. See "Creating Aliases (Synonyms) for Presentation Layer Objects" for more information about aliases.
9. Use the Child Presentation Tables tab to specify presentation tables that you want to show as nested folders in Answers and BI Composer. See "Nesting Folders in Answers and BI Composer" for more information.
10. Click **OK**.

To reorder a table or sort all tables in a subject area:

1. In the Presentation layer, double-click a subject area.
2. In the Subject Area dialog, click the Presentation Tables tab.
3. To move a table, in the **Name** list, select the table you want to reorder. Then, use drag-and-drop to reposition the table, or click the **Up** and **Down** buttons.
4. To sort all tables in alphanumeric order, click the **Name** column heading. This toggles the sort between ascending and descending alphanumeric order.

Creating and Managing Presentation Columns

The presentation column names are, by default, identical to the logical column names in the Business Model and Mapping layer.

However, you can present a different name by clearing both the **Use Logical Column Name** and the **Custom display name** options in the Presentation Column dialog.

To provide a convenient organization for your users, you can drag and drop a column from a single logical table in the Business Model and Mapping layer onto multiple presentation tables. This lets you create categories that make sense to the users. For example, you can create several presentation tables that contain different classes of measures: one containing volume measures, one containing share measures, one containing measures from a year ago, and so on.

Caution: When you drag columns to presentation tables, make sure that columns with the same name or an alias of the same name do not already exist.

Typically, presentation columns are created automatically by dragging and dropping logical columns from the logical layer.

To edit the properties of a presentation column:

1. In the Presentation layer, double-click a presentation column to display the Presentation Column dialog.
2. In the General tab, to specify a name that is different from the Logical Column name, clear **Use Logical Column Name**, and then type a name for the column. Note that aliases are created automatically whenever presentation objects are renamed, so that any queries using the original name do not break.

   Note: You must enable the **Edit presentation names** Administration Tool option before you can edit the presentation column's name.

3. To set permissions for this presentation column, click **Permissions**. See "Setting Permissions for Presentation Layer Objects" for more information.
4. Select **Custom display name** or **Custom description** to dynamically display a custom name based on a session variable or custom description based on a session variable.

   Clearing **Use Logical Column Name** and selecting **Custom display name** allows you to edit the **Translation Key** field.

   The **Custom display name**, **Custom description**, and **Translation Key** fields are used typically for localization purposes. When you externalize strings in the Presentation layer and run the Externalize Strings utility, the results contain the session variable information and the translation key.

   See "Localizing Oracle Business Intelligence" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about localization.

   Note: You can also use the **Custom display name** and **Custom description** fields to propagate UI hints (labels and tooltips) from an ADF data source to display in Oracle BI Answers. See "Propagating Labels and Tooltips from ADF Data Sources" for more information about how to set up ADF data source UI hints.

5. The **Logical Column** field displays the name of the logical column for this presentation column. Click **Edit** to make any changes to the logical column object.
6. Optionally, you can specify an expression in the **Hide object if** field that controls whether this presentation column is visible in the Subject Area Tree in Answers and BI Composer. Leave this field blank (the default) to show the object. See "Controlling Presentation Object Visibility" for more information.
7. Use the Aliases tab to specify or delete aliases for this presentation column. See "Creating Aliases (Synonyms) for Presentation Layer Objects" for more information about aliases.

To reorder a presentation column:

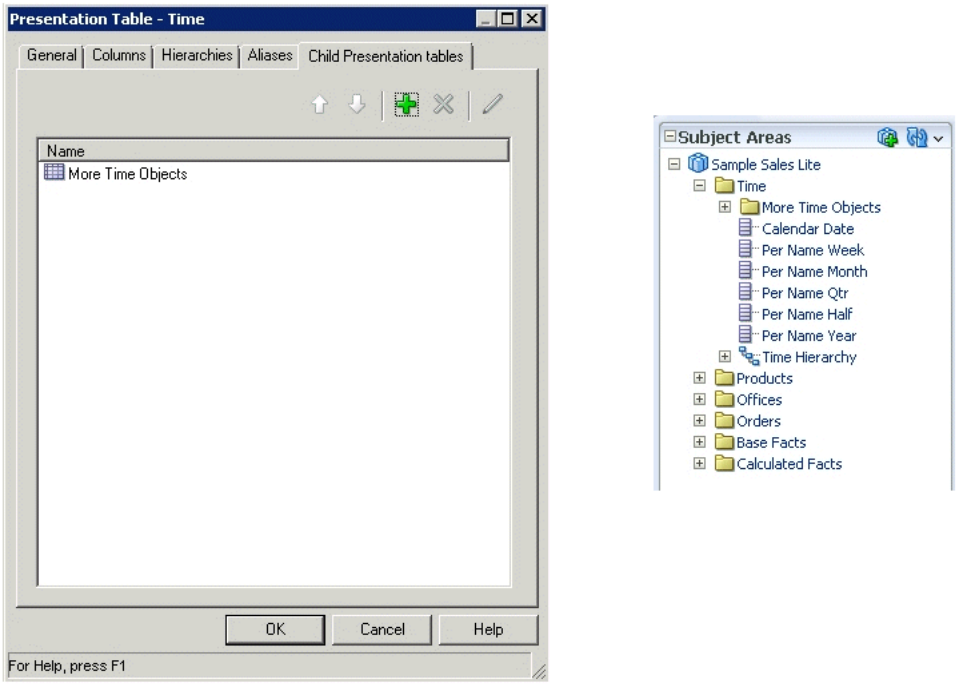
1. In the Presentation layer, right-click a presentation table and select **Properties**.
2. Click the **Columns** tab.
3. Select the column you want to reorder.
4. Use drag-and-drop to reposition the column, or click the **Up** and **Down** buttons.
5. Click **OK**.

Nesting Folders in Answers and BI Composer

You can use the Child Presentation Tables tab in the Presentation Table dialog to designate child presentation tables. You designate child presentation tables to give the appearance of nested folders in Answers and BI Composer. You can add multiple layers of nesting using this method.

Figure 12–2 shows how a designated child presentation table appears nested in Answers.

Figure 12–2 Achieving Nested Folders in Answers



Note that the folders only appear to be nested - they are not actually nested in terms of drill-down, and the qualified names of the objects remain the same. In addition, the Presentation layer in the Administration Tool does not display the nesting; the nesting only appears in Answers and BI Composer. This feature only works for presentation tables, and not for other Presentation layer objects.

When you run a consistency check, the Consistency Check Manager detects any circularity in parent-child presentation table assignment. It also detects and reports project definitions that include child presentation tables without parent presentation tables.

In previous releases, repository developers could achieve one level of nesting by adding a hyphen at the beginning of a presentation table name, or by adding an arrow (->) at the beginning of the presentation table description. This method is now deprecated for this release and will be removed in a future release.

If you previously used hyphens at the beginning of presentation table names or arrows at the beginning of presentation table descriptions to achieve nesting, it is recommended that you run the Convert Presentation Folders utility to convert your metadata to the new structure. See "Using the Convert Presentation Folders Utility" for more information.

Working with Presentation Hierarchies and Levels

Presentation hierarchies and presentation levels provide an explicit way to expose the multidimensional model in Oracle BI Answers. When presentation hierarchies and levels are defined in the Presentation layer, roll-up information is displayed in the Oracle BI Answers navigation pane, providing users with important contextual information.

Be aware that members in a presentation hierarchy are not visible in the Presentation layer. Instead, you can see hierarchy members in Answers.

Most importantly, users can create hierarchy-based queries using these objects. Presentation hierarchies expose analytic functionality such as member selection, custom member groups, and asymmetric queries.

As with other Presentation layer objects, you can also provide localization information and apply fine-grained access control to presentation hierarchies and levels.

If you have a repository from a previous release, note that presentation hierarchies do not appear in the Presentation layer automatically as part of the Oracle BI repository upgrade process. You must manually create these objects by dragging logical dimensions from the Business Model and Mapping layer to the appropriate presentation tables.

This section contains the following topics:

Creating and Managing Presentation Hierarchies

To create a presentation hierarchy, you can drag a logical dimension hierarchy from the Business Model and Mapping layer to a table in the Presentation layer. The presentation hierarchy object must be located within a presentation table, unlike in the Business Model and Mapping layer, where logical dimensions are peer objects of tables. Presentation hierarchies are also displayed within their associated tables in Oracle BI Answers, providing a conceptually simpler model.

If a logical dimension spans multiple logical tables in the Business Model and Mapping layer, it is a best practice to model the separate logical tables as a single presentation table in the Presentation layer.

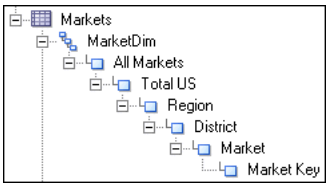
There are several ways to create presentation hierarchies:

- When you drag an entire business model to the Presentation layer, the presentation hierarchies and constituent levels appear automatically, along with other presentation objects.

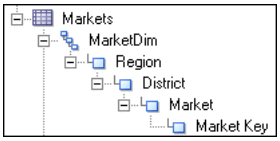
- When you drag a logical dimension table to the Presentation layer, presentation hierarchies and levels based on those dimensions are created automatically.
- You can also drag individual logical dimensions to the appropriate presentation tables to create corresponding presentation hierarchies within those tables.
- As with most other objects in the Administration Tool, you can right-click a presentation table and select **New Object > Presentation Hierarchy** to manually define the object.

You can also drag an individual logical level from the Business Model and Mapping layer to a presentation table to create a presentation hierarchy that is a subset of the logical dimension hierarchy.

For example, suppose a logical dimension has the levels All Markets, Total US, Region, District, Market, and Market Key. Dragging and dropping the entire logical dimension to the corresponding presentation table appears as follows:

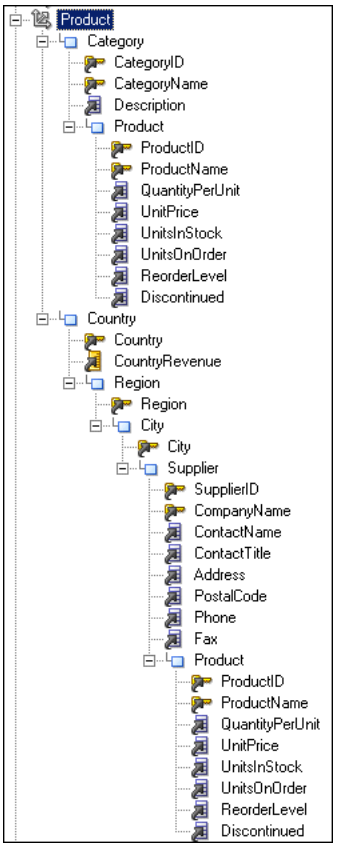


However, dragging and dropping the Region level to the same presentation table appears as follows:

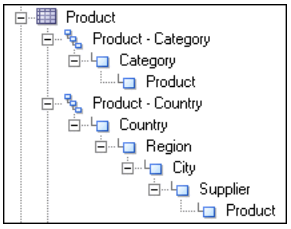


Modeling Dimensions with Multiple Hierarchies in the Presentation Layer

For logical dimensions that contain multiple logical hierarchies, multiple separate presentation hierarchies are created. For example, the following logical dimension called Product contains the two hierarchies Category and Country:



In the Business Model and Mapping layer, this logical dimension is modeled as a single dimension object that contains multiple hierarchies. In contrast, the Presentation layer models this dimension as two separate objects: one that displays the drill path through the Category level, and another that shows the drill path through the Country level, as follows:



Editing Presentation Hierarchy Objects

You can edit presentation hierarchy properties, including setting permissions to apply role-based access control, setting a custom display name for localization purposes, and changing the levels in a hierarchy.

To edit the properties of a presentation hierarchy:

1. In the Presentation layer, double-click a presentation hierarchy to display the Presentation Hierarchy dialog.
2. In the General tab, you can change the following:
   - **Name.** Note that aliases are created automatically whenever presentation objects are renamed, so that any queries using the original name do not break.

Note: You must enable the **Edit presentation names** Administration Tool option before you can edit the presentation hierarchy's name.

- **Permissions.** See "Setting Permissions for Presentation Layer Objects" for more information.
- **Custom display name** and **Custom description.** Select **Custom display name** to dynamically display a name based on a session variable and to edit the **Translation Key** field. Select **Custom description** to dynamically display a custom description based on a session variable.

  These options are used typically for localization purposes. When you externalize strings in the Presentation layer and run the Externalize Strings utility, the results contain the session variable information and the translation key.

  See "Localizing Oracle Business Intelligence" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about localization.
- **Logical Dimension.** This field displays the name of the logical dimension for this presentation hierarchy. Click **Browse** to select a different logical dimension.
- **Hide object if.** This field lets you specify an expression that controls whether this presentation hierarchy is visible in the Subject Area Tree in Answers and BI Composer. Leave this field blank (the default) to show the object. See "Controlling Presentation Object Visibility" for more information.

3. The Levels tab lists the levels within the hierarchy and their order. This tab is not available for parent-child hierarchies. You can add, delete, or reorder levels. You can also click the **Edit** button to edit properties for a particular level. See "Creating and Managing Presentation Levels" for information about level properties.
4. The Display Columns tab is only available for parent-child hierarchies. Because parent-child hierarchies do not contain levels, display columns are defined for the presentation hierarchy object as a whole. Use the Display Columns tab to define which columns should be used for display for this parent-child hierarchy.

   You can add, delete, or reorder display columns. You can also click the **Edit** button to edit properties for a particular column.
5. Use the Aliases tab to specify or delete aliases for this presentation hierarchy. See "Creating Aliases (Synonyms) for Presentation Layer Objects" for more information about aliases.

Creating and Managing Presentation Levels

Presentation levels are displayed within hierarchical columns in Oracle BI Answers. Presentation levels are typically created automatically when presentation hierarchies are created.

To edit the properties of a presentation level:

1. In the Presentation layer, double-click a presentation level to display the Presentation level dialog.
2. In the General tab, you can change the following:

- **Name.** Note that aliases are created automatically whenever presentation objects are renamed, so that any queries using the original name do not break.

  Note: You must enable the **Edit presentation names** Administration Tool option before you can edit a presentation level's name.

- **Permissions.** See "Setting Permissions for Presentation Layer Objects" for more information.
- **Custom display name** and **Custom description.** Select **Custom display name** to dynamically display a name based on a session variable and to edit the **Translation Key** field. Select **Custom description** to dynamically display a custom description based on a session variable.

  These options are used typically for localization purposes. When you externalize strings in the Presentation layer and run the Externalize Strings utility, the results contain the session variable information and translation key.

  See "Localizing Oracle Business Intelligence" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about localization.
- **Logical Level.** This field displays the name of the logical level for this presentation level. Click **Browse** to select a different logical level.

Note that specifying an expression in the **Hide object if** field has no effect on the visibility of presentation levels in the Subject Area Tree in Answers and BI Composer. This field is reserved for a future release.

3. The Drill To Levels and Drill From Levels tabs are reserved for a future release and are not currently used. An additional option called **Generate Drill Graph**, available as a right-click option for any Presentation layer object, is also reserved for a future release.
4. Use the Display Columns tab to define which columns should be used for display for that level (on drill-down). For example, if two columns called "Name" and "ID" exist at the same level, you can choose to display "Name" because it is the more user-friendly option.

   You can add, delete, or reorder display columns. You can also click the **Edit** button to edit properties for a particular column.

   As an alternative to defining display columns in this tab, you can drag a presentation column directly onto the presentation level in the Presentation layer of the Administration Tool. Doing this automatically adds the column as a display column for the presentation level.

   Note that the display columns that appear by default when a presentation level is created are based on which key columns for the corresponding logical level have the **Use for display** option selected.
5. Use the Aliases tab to specify or delete aliases for a presentation level. See "Creating Aliases (Synonyms) for Presentation Layer Objects" for more information about aliases.

Setting Permissions for Presentation Layer Objects

You can apply access control to restrict which individual users or application roles (groups) can access particular presentation layer objects.

For example, you can provide read-only access to a set of presentation tables for a particular application role, read-write access for a second application role, and no access for a third application role.

You can also use the Identity Manager to set up privileges and permissions. The Identity Manager is useful for setting permissions for individual application roles to many objects at once, unlike permissions in the Presentation layer, which you can only set for one object at a time. See "Setting Up Object Permissions" for information about setting up object permissions in the Identity Manager. For a full description of data access security in Oracle Business Intelligence, see Chapter 14.

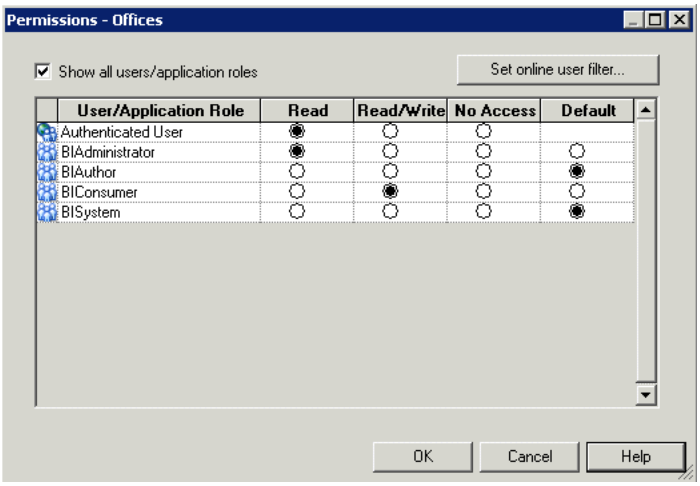
You can control what level of privilege is granted by default to the AuthenticatedUser application role, which is the default application role associated with new repository objects. To do this, set the DEFAULT_PRIVILEGES parameter in the NQSConfig.INI file. See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

To set permissions for presentation layer objects:

1. In the Presentation layer, double-click a presentation object, such as a subject area, table, column, or hierarchy.
2. In the General tab, click **Permissions**.

   Figure 12–3 shows the Permissions dialog.

Figure 12–3 Permissions Dialog



3. In the Permissions dialog, any users or application roles with the **Default** permission do not appear in the User/Application Roles list. Select **Show all users/application roles** to see users and application roles with the Default permission.

   In online mode only, by default, no users are retrieved, even when **Show all users/application** roles is selected. Click **Set online user filter** to specify the set of users you want to retrieve.

   The filter is empty by default, which means that no users are retrieved. Enter * to retrieve all users, or enter a combination of characters for a specific set of users, such as A* to retrieve all users whose names begin with the letter A. The filter is not case-sensitive.

4. For each user and application role, you can allow or disallow access privileges for this presentation object by selecting one of the following options:

- **Read.** Only allows read access to this object.
- **Read/Write.** Provides both read and write access to this object.
- **No Access.** Explicitly denies all access to this object.
- **Default.** The permission is inherited from the parent object. For subject areas, because they are a top-level object, Default is equivalent to the permission granted to the AuthenticatedUser application role.

5. Click **OK**.
6. Click **OK** in the Properties dialog for this presentation object.

Generating a Permission Report for Presentation Layer Objects

You can generate a permission report for individual presentation layer objects to see a summary of how permissions have been applied for that object.

To do this, right-click any presentation object and select Permission Report. The Permission Report dialog displays the name and a description of the presentation object, along with a list of users/application roles and their permissions.

Sorting Columns in the Permissions Dialog

There are six ways that you can sort the types and User/Application Role names in the Permissions dialog.

To change the sort, click the heading of the first or second column. The first column has no heading and contains an icon that represents the type of user or application role. The second column contains the name of the User/Application Role object. Note that you cannot sort on the columns for individual object permissions (like Read, Read/Write, and so on).

There are three ways to sort by type, and two ways to sort the list of user and application role names. This results in a total of six possible sort results (3 x 2 = 6). The following list shows the sort results available by clicking the type column:

- AuthenticatedUser, Application Roles, Users (ascending by name of type)
- Users, Application Roles, AuthenticatedUser (descending by name of type)
- Type column is in no particular order (Type value is ignored, as all names in User/Application Role column are sorted in ascending order by value in User/Application Role column)

The following list shows the sort results available by clicking the User/Application Role column:

- Ascending within the type
- Descending within the type

Creating Aliases (Synonyms) for Presentation Layer Objects

Each presentation object can have a list of aliases (synonyms) for its name that can be used in Logical SQL queries.

To create the list of aliases, use the Alias tab in the Properties dialog for the appropriate presentation object (subject area, presentation table, presentation hierarchy, presentation level, or presentation column).

Because Presentation layer objects are often deleted and then re-created during the repository development process, it is best to wait until your logical business model is relatively stable before creating aliases for presentation objects.

You can use this feature to rename presentation objects without breaking references that any existing requests have to the old names, including requests from Answers, Oracle BI Publisher, or other Logical SQL clients. If you are still developing a new repository, you might want to wait until the repository is stable before renaming objects.

For example, consider a subject area called "Sample Sales Reduced" that contains a presentation table called "Facts Other." If you rename the presentation column called "# of Customers" to "Number of Customers," any requests that use "# of Customers" fail. However, if you add "# of Customers" to the list of synonyms in the Alias tab for the "Number of Customers" column, then queries containing both "# of Customers" and "Number of Customers" succeed and return the same results.

Note the following:

- Aliases for presentation objects do not appear in Answers or other query clients when creating new queries. Only the primary names of subject areas, hierarchies, levels, tables, and columns appear.
- This feature works in a different way from SQL aliases or the alias feature in the Physical layer. It simply provides synonyms for object names, much like "synonyms" in SQL.
- Aliases are created automatically when you rename presentation objects. For example, if you change Catalog to Catalog1, the original name Catalog is added to the Aliases list.
- You cannot rename a Presentation layer object to a name that is already in use as an alias for an object of the same type.

To add or delete an alias for a presentation object:

1. In the Presentation layer, double-click a presentation object, such as a subject area, table, column, or hierarchy.
2. Click the Aliases tab.
3. To add an alias, click the **New** button, and then type the text string to use for the alias.
4. To delete an alias, select the alias you want to delete from the **Aliases** list, then click the **Delete** button.
5. Click **OK**.

Controlling Presentation Object Visibility

You can use the **Hide object if** field to hide selected Presentation layer objects in the Subject Area Tree in Answers and BI Composer. You can hide subject areas, tables, columns, and hierarchies. Note that although the **Hide object if** field is shown for presentation levels, it is a placeholder for a future release and currently has no effect on presentation level objects.

The **Hide object if** field only controls object visibility and does not affect object access. For example, objects that are hidden can be queried using tools like `nqcmd`.

There are three different types of expressions that you can use in the **Hide object if** field to determine Presentation layer object visibility:

- **Constant.** Enter any non-zero constant in the field to hide the object. Enter 0 or leave the field blank to display the object.
- **Session variable.** You can use a session variable in the expression to control whether the object is hidden. If the expression evaluates to a non-zero value, the object is hidden. If the expression evaluates to zero, is empty, or has no value definition, the object is displayed. The session variable must be populated using a session initialization block or a row-wise initialization block. You must properly populate the session variable to control the visibility.

  The SQL for the init block can use `CASE` statements to control whether to return zero or a non-zero number in the session variable. An example of an expression with a session variable might be:

  ```
  VALUEOF(NQ_SESSION."VISIBLE")
  ```

  Note that session variable names that include periods must be enclosed in double quotes.
- **Session variable comparison.** You can use an equality or inequality comparison to control whether the object is hidden, using the following form:

  ```
  'session_variable_expression' '=|<>' 'constant'
  ```

  If the expression evaluates to zero, null, or empty, the object is displayed. If the expression evaluates to a non-zero value, the object is hidden.

  For example:

  ```
  NQ_SESSION."VISIBLE" = 'ABC'
  NQ_SESSION."VISIBLE" <> 'ABC'
  ```

  Note that session variable names that include periods must be enclosed in double quotes.

  You can use any scalar function supported by Oracle BI EE in the **Hide object if** expression. Scalar functions include any function that accepts a simple value for each of its arguments and returns a single value. Most functions listed in the following sections can be used, except for functions that return non-deterministic results like `RAND`, `NOW`, `CURRENT_DATE`, `CURRENT_TIMESTAMP`, `CURRENT_TIME`, and so on:

  - All String Functions
  - Most Math Functions (except `RAND`)
  - Most Calendar Date/Time Functions (except `NOW`, `CURRENT_DATE`, and so on)
  - Most Conversion Functions (like CAST, IFNULL, TO_DATETIME, and VALUEOF)

  For example, the following expression checks to see if the NQ_SESSION.VISIBLE session variable begins with the letters ABC:

  ```
  LEFT(VALUEOF(NQ_SESSION."VISIBLE"), 3) = 'ABC'
  ```

  Similarly, the following expression checks to see if the given variable begins with ExtnAttribute:

  ```
  Left(VALUEOF(NQ_SESSION."ADF_LABEL_ORACLE.APPS.CRM.MODEL.ANALYTICS.
  APPLICATIONMODULE.CRMANALYTICSAM_CRMANALYTICSAMLOCAL_CRMANALYTICSAM.
  OPPORTUNITYAM.OPPORTUNITY_EXTNATTRIBUTECHAR001"), 13) = 'ExtnAttribute'
  ```

When you run a consistency check, the Consistency Check Manager detects any inconsistencies in the visibility filter expression.

13

Creating and Persisting Aggregates for Oracle BI Server Queries

This chapter explains how to set up and use aggregate persistence in Oracle Business Intelligence.

Most data warehouse practitioners create aggregated data tables to dramatically improve the performance of highly summarized queries. These aggregate tables store precomputed results that are aggregated measures (typically summed) over a set of dimensional attributes. Using aggregate tables is a typical technique used to improve query response times in decision support systems.

If you write SQL queries or use a tool that only understands what physical tables exist and not their meaning, then using aggregate tables becomes more complex as the number of aggregate tables increases. The aggregate navigation capability of the Oracle BI Server allows queries to use the information stored in aggregate tables automatically. The Oracle BI Server lets you concentrate on asking the right business question, and then the server decides which tables provide the fastest answers.

Oracle Business Intelligence has an aggregate navigation feature to take advantage of those aggregates in source databases (for more information, see Chapter 11). However, it can be time consuming to create and maintain the data aggregation, as well as load database scripts and the corresponding metadata mappings. For that reason, Oracle Business Intelligence provides an aggregate persistence feature that automates the creation and loading of the aggregate tables and their corresponding Oracle Business Intelligence metadata mappings.

This chapter contains the following topics:

About Aggregate Persistence in Oracle Business Intelligence

Use the Aggregate Persistence feature to create aggregates for Oracle BI Server queries. The Aggregate Persistence Wizard lets you automate the creation of the aggregate specification script. When you run this script against a live Oracle BI Server, aggregate tables are created by the aggregate persistence engine and are mapped into the metadata for navigation. When aggregates are persisted, indexes and statistics are created on relational tables for greater performance.

The Aggregate Persistence Wizard creates a SQL script that you can run on a scheduled basis against the Oracle BI Server. In the Aggregate Persistence Wizard, you specify the measures, dimensionality, and other parameters of each star or cube based on your performance design. The script should run after each load of the base-level tables, so that the aggregates are always synchronized with the detail-level data when the load window completes and users begin to run queries.

Aggregate creation runs against the master server in a cluster. It takes some time for the metadata changes to propagate to the slaves. The cluster refresh time is a user-controlled option and results might be incorrect if a query hits a slave server before it is refreshed. It is the administrator's responsibility to set an appropriate cluster refresh interval.

Aggregate persistence requires a dedicated connection pool to create tables or cubes in the target database that will hold the aggregates. Because the Oracle BI repository enables federation, the aggregated target can be on the same database as the detailed source, or in a completely different database. This dedicated connection pool must be created before you run the Aggregate Persistence Wizard, so it can be selected during the appropriate step of the wizard.

The default prefix `SA_` is automatically added to dimension (level) aggregates. You can change this default prefix by updating the `AGGREGATE_PREFIX` parameter in the `AGGREGATE_PERSISTENCE` section of the NQSConfig.INI file:

```
AGGREGATE_PREFIX = "prefix_name" ;
```

The target schema used to store aggregates must be appropriately secured and should not allow public access. The schema should have privileges to connect, create, and drop tables and indexes. By default, only users who have administrator privileges can manage aggregates.

Do not use aggregate persistence against tables with active Virtual Private Database (VPD) security filters. There is a possibility that the aggregate information might be persisted without the VPD filter, posing a security risk.

Aggregate Persistence Improvements

In versions prior to Oracle BI EE 11*g* Release 1 (11.1.1.9), aggregate persistence limitations prevented the creation of some aggregates or created ineffectual aggregates. Oracle Business Intelligence now automatically creates more usable aggregates and creates aggregates without having to fix data set errors or modeling problems.

Surrogate Keys

Aggregate persistence can create surrogate keys for joining dimensions to fact aggregate tables. In most cases the source and the target databases are not the same

instance, and in releases prior to Oracle BI EE 11*g* Release 1 (11.1.1.9) the Oracle BI Server needed to perform cross database (sort-merge) joins during surrogate key creation. This join method was costly because it required the fact aggregate to be internally-sorted multiple times when joining dimensions to the fact table.

The Oracle BI Server now uses the hash join method to improve surrogate key creation. Where possible, a new request variable is automatically added to the fact aggregate population query and when this request variable is set, the query engine builds hash joins for the dimension tables in parallel before joining to the fact table.

The Oracle BI Summary wizard displays the **Use surrogate keys** option to suggests when you should use surrogate keys. When this option is selected, the `using_surrogate_key` clause is added to all levels in the aggregate specification.

Auto Correction (Hardening) of Level Keys

In releases prior to Oracle BI EE 11*g* Release 1 (11.1.1.9), aggregate persistence required unique level keys for the specified aggregate level or levels. If the level keys were not unique, then the Model Check Manager would warn about non-unique level keys, incorrect values would result when a query was sourced from such an aggregate, or aggregate creation on TimesTen would fail with the "unique index" error.

Aggregate persistence now auto-corrects, or hardens, level keys that are not unique. During aggregate creation, the aggregate persistence engine identifies any level key that is not unique and, when possible, generates a unique expanded key by adding key columns from the parent level. This unique expanded key is then used for all operations on the aggregate. Note that the key is expanded internally only for the purpose of the aggregate, and the metadata (key definition) is not modified.

If you use the Oracle BI Summary Advisor feature to identify which aggregates will increase query performance and if the level key for the level is not unique and cannot be auto-corrected by aggregate persistence, then all fact aggregates dependent on the level will not be materialized.

The Oracle BI Summary Advisor recommends aggregates with level keys that are unique as defined, or with level keys that can be auto-corrected (hardened) to be made unique. However, be aware that modifications to underlying data might impact such aggregates.

To improve performance, Oracle suggests creating aggregates using surrogate key rather than natural keys. Auto-correction, or hardening, is not as effective when natural keys are used, especially in the prepare-create mode of operation.

Unbalanced (Ragged) and Skip-Level Hierarchies

In releases prior to Oracle BI EE 11*g* Release 1 (11.1.1.9), aggregate persistence could not create aggregates for logical dimensions with unbalanced or skip-level hierarchies.

Aggregate persistence now creates aggregates for logical dimensions with unbalanced or skip-level hierarchies. These aggregates can be created with or without using surrogate keys. The Oracle BI Summary Advisor has also been improved to recommend aggregates that contain logical dimensions with unbalanced and skip-level hierarchies.

Chronological Keys

The Oracle BI Server requires chronological keys to support time series functions such as AGO, TODATE, and PERIODROLLING. Time series functions operate correctly when only the lowest key in the logical dimension is chronological. However, in

releases prior to Oracle BI EE 11*g* Release 1 (11.1.1.9), this lack of chronological keys at higher levels caused aggregates at those levels to be unusable by the Oracle BI Server when it processed queries that use time series functions.

Aggregate persistence now generates chronological keys with the `CK_` prefix for time levels without chronological keys but at which aggregates are to be created. A new column is added to the physical dimension aggregate table to store the chronological key value, and a new logical column is added to the logical table of the time dimension. This column is then mapped to the new column added to the physical dimension aggregate table.

In the current release, the `delete aggregates` statement will automatically remove all metadata created to support generated chronological keys.

Count Distinct Measures

In releases prior to Oracle BI EE 11*g* Release 1 (11.1.1.9), aggregate persistence computed and stored the count values for the specified level combination. However, these aggregate could not be used to satisfy queries involving such measures at a higher grain, and the Oracle BI Server had to fetch relevant data from the base fact table to satisfy such queries. Due to these factors, the usefulness of such an aggregate was reduced, and sub-optimal queries were generated.

The Oracle BI Server now utilizes aggregates with count distinct measures to serve queries for these measures at higher grains. The Aggregate Persistence wizard includes the **Persist Count Distinct Measures as raw values** option which when selected appends `as_raw_values` to all the valid count distinct measures specified. When this option is selected, aggregate persistence also sets an aggregation expression override on the corresponding logical column for the system-generated aggregate logical table source. The Oracle BI Summary Advisor now recommends both methods of persistence for count distinct measures.

Identifying Query Candidates for Aggregation

When creating aggregates, you must identify which queries would benefit substantially from aggregated data. You will achieve the best results by aggregating to the highest level possible.

To identify slow-running queries, perform the following tasks:

- **Enable usage tracking in the Oracle BI Server.** Usage tracking statistics can be used in a variety of ways, such as database optimization, aggregation strategies, and billing users or departments based on the resources they consume. The Oracle BI Server tracks usage at the detailed query level. When you enable usage tracking, statistics for every query are written to a usage tracking log file or inserted into a database table.

  Note: It is strongly recommended that you use the direct insertion into a database method for usage tracking. See "Managing Usage Tracking" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for full information about usage tracking.

- **Analyze the query run times and identify the slowest running queries as candidates for aggregation.** The run time for creating aggregates is dependent on the type of aggregates selected by the user. Creating aggregates from large fact

tables is slower than from smaller tables. You should carefully select the aggregates to be created.

Using Oracle BI Summary Advisor to Identify Query Candidates for Aggregation

If you are running Oracle Business Intelligence on the Oracle Exalytics Machine, you can use the Oracle BI Summary Advisor feature to identify which aggregates will increase query performance and to generate a script for creating the recommended aggregates.

Note: If you are not running Oracle Business Intelligence on the Oracle Exalytics Machine, the Oracle BI Summary Advisor feature is not available.

This section contains the following topics:

About Oracle BI Summary Advisor

To reduce query time, you can create aggregate tables that store precomputed results for queries that include rolled-up data. Before creating aggregates, however, you need to analyze usage tracking statistics to identify which aggregates will increase query performance. As an alternative to manually identifying aggregates, which can be a slow and laborious process, you can use the Summary Advisor feature, which intelligently recommends an optimal list of aggregate tables based on query patterns that will achieve maximum query performance gain while meeting specific resource constraints. Summary Advisor then generates an aggregate creation script that can be run to create the recommended aggregate tables.

There are several parts to the Summary Advisor feature:

- Statistics collection into a designated statistics database. You must enable the Usage Tracking feature to collect Summary Advisor statistics.
- Summary Advisor execution through the Summary Advisor Wizard in the Administration Tool to evaluate the collected statistics and recommend aggregates.

This section contains the following topics:

Gathering Summary Advisor Statistics

Before Summary Advisor can generate recommendations, you must obtain a representative sample of usage statistics for Summary Advisor to use.

Use one of the following approaches to accomplish this task:

- Enable Usage Tracking and Summary Advisor Logging on a production system, and let users run queries against BI Server for several days. The Summary Advisor Statistics Table will be populated with usage statistics. See "Turning On Usage Tracking" and "Turning On Summary Advisor Logging" for more information.

  Note that enabling Usage Tracking and Summary Advisor Logging will have a minor system performance impact on production systems.

- In a test environment, run a representative workload against the Oracle BI Server to gather Summary Advisor statistics. A representative workload is a list of commonly requested Logical SQL statements. You typically obtain a representative workload from your production environment.

  After you have the representative workload, enable Usage Tracking and Summary Advisor Logging on the Oracle BI Server in your test environment, and use the `nqcmd` utility to run the workload against the Oracle BI Server. See "Using nqcmd to Test and Refine the Repository" for more information about using `nqcmd`. The Summary Advisor Statistics Table will be populated with usage statistics.

Generating and Using Summary Advisor Recommendations

After the Summary Advisor Statistics Table is populated with representative data, Summary Advisor can analyze the data and generate aggregate recommendations to speed up queries.

To accomplish this, run the Oracle BI Summary Advisor Wizard in the Administration Tool to generate an aggregate specification, and then use the aggregate specification to create aggregates using `nqcmd`. See "Using the Oracle BI Summary Advisor Wizard" and "Running the Aggregate Specification Against the Oracle BI Server" for more information.

Currently, Oracle BI Summary Advisor only supports aggregate creation on Oracle TimesTen In-Memory Database. Refer to "System Requirements and Certification" for information about supported versions.

You can also save your Summary Advisor options to a file, so that you can re-run the Oracle BI Summary Advisor Wizard later without having to enter the same options again.

About Measure Subset Recommendations

When the **Only include measures used in queries** option is set in the Miscellaneous screen in the Summary Advisor Wizard, then Summary Advisor recommends only aggregates that contain specific measures that are both present in the analyzed query workload, and that can optimize the query workload if aggregates are created.

Note the following information about measure subset recommendations:

- Summary Advisor does not include measures that are not used in the query workload in its recommended aggregates.
- Size estimation of aggregate fact tables is based on the recommended measure subset instead of using all the measures of a logical fact table during estimation.
- Summary Advisor does not include measures that are invalid for aggregate persistence in its recommended aggregates.

If you are upgrading from a release previous to Oracle BI EE 11*g* Release 1 (11.1.1.7), then note the following:

- This feature requires a summary statistics table that contains the MEASURECOLUMNIDVECTOR and GROUPBYCOLUMNIDVECTOR columns, with summary statistics data with Version value of 11.1.1.7.0 or later. After running the Patch Set Assistant, you must import the new columns into the repository and re-run the workload. Then, re-run Summary Advisor with the new statistics and re-create the aggregates.

  See *Oracle Fusion Middleware Patching Guide* for more information about running the Patch Set Assistant.

- If you do not import the new columns after running the Patch Set Assistant, you can use Summary Advisor, but measure subset recommendation will not be enabled.

- If both pre-11.1.1.7.0 and 11.1.1.7.0 statistics data are analyzed by Summary Advisor for a particular aggregate grain, pre-11.1.1.7.0 behavior is enforced for the aggregate recommended at this grain (in other words, all measures will be included).

Setting Up the Statistics Database

Before you can use the Oracle BI Summary Advisor feature, you must set up a database to store the collected statistics. You must run the Repository Creation Utility (RCU) on the target database to create the required statistics schema.

See *Oracle Fusion Middleware Installation Guide for Oracle Business Intelligence* for more information about using RCU to create database schemas.

Note the following items:

- Typically, you use the database you installed for use with Oracle Business Intelligence as the statistics database because this database already has the RCU-created schemas. The RCU-created table name for Summary Advisor is `S_NQ_SUMMARY_ADVISOR`.

- You also need to import the database into the Physical layer of the Oracle BI repository.

- You must use the same database for Summary Advisor that you use for usage tracking. If you already have a database and schema set up for usage tracking, you can skip the steps in this section.

To set up the statistics database:

1. Run the Repository Creation Utility on an external database of your choice. You can skip this step if you choose to use the database you installed for use with Oracle Business Intelligence for Summary Advisor statistics, because this database has the RCU-created tables already.
2. Open the Administration Tool and import the database into the Physical layer. See "Importing Metadata from Relational Data Sources" for more information.
3. Save and close the repository.
4. Use the "Upload Repository Command" to upload the repository and make it available for queries. See "Making the Repository Available for Queries" for more information.

Columns in the S_NQ_SUMMARY_ADVISOR Table

Table 13–1 shows the columns in the S_NQ_SUMMARY_ADVISOR table.

Table 13–1 Columns in S_NQ_SUMMARY_ADVISOR

Column	Description
GROUPBYCOLUMNIDVECTOR	Upgrade IDs for logical column objects that represent group-by columns in a processing path[1].
LOGICALFACTTABLEID	Upgrade ID of the logical fact table.
LOGICALTABLESOURCEIDVECTOR	Upgrade IDs of the logical table sources.
LOGICAL_QUERY_ID	Foreign key that references the ID column in S_NQ_ACCT. This column helps identify the logical SQL that generated this processing path.
MEASURECOLUMNIDVECTOR	Upgrade IDs for logical column objects that represent measures in a processing path.
PROCESSINGTIMEINMILLISEC	Time spent on this processing path, in milliseconds.
QUERYLEVELIDVECTOR	Upgrade IDs of the logical levels in a processing path.
QUERYSTATUS	For internal use only.
ROW_COUNT	The number of rows retrieved in a processing path. Data in this column is reserved for use by Oracle BI Summary Advisor in a future release.
SOURCECELLLEVELIDVECTOR	Upgrade IDs of the logical levels in the logical table source.
VERSION	Version number of the Oracle BI Server.

[1] A *processing path* is an internal Oracle BI Server term. It represents a subquery that involves a single fact logical table source.

Turning On Usage Tracking

You must enable usage tracking before collecting Summary Advisor statistics.

See "Managing Usage Tracking" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

Turning On Summary Advisor Logging

When you are ready to collect statistics, you can enable Summary Advisor logging. For new (non-upgraded) installations, the Summary Advisor parameters are centrally managed. You use the System MBean Browser in Fusion Middleware Control to set the Summary Advisor parameters.

To enable Summary Advisor logging using the System MBean Browser:

1. In Fusion Middleware Control, in the Navigator window, expand the WebLogic Domain folder and the bifoundation_domain node.
2. Right-click the **AdminServer** node and select **System MBean Browser**.
3. Expand Application Defined MBeans, then expand oracle.biee.admin, then expand Domain: bifoundation_domain.
4. Lock the domain, as follows:
   a. Expand BIDomain and select the BIDomain MBean where group=Service.

b. Display the Operations tab.

c. Click the **lock** link.

5. Expand BIDomain.BIInstance.ServerConfiguration, then select the **BIDomain.BIInstance.ServerConfiguration** MBean.

6. Ensure that the **UsageTrackingCentrallyManaged** attribute is set to **true**. When UsageTrackingCentrallyManaged is set to false, the following parameters are managed using the NQSConfig.INI file on each Oracle BI Server computer rather than the SystemMBean Browser:

   - SummaryAdvisorTableName
   - SummaryStatisticsLogging
   - UsageTrackingConnectionPool
   - UsageTrackingDirectInsert
   - UsageTrackingEnabled
   - UsageTrackingPhysicalTableName

7. Set the **SummaryAdvisorTableName** attribute to the name of the fully-qualified database table for collecting statistics, as it appears in the Physical layer of the Oracle BI repository. For example:

   "My_DB"."DEV_BIPLATFORM"."S_NQ_SUMMARY_ADVISOR"

   The table name you specify must belong to the same database object and connection pool that you are using for usage tracking.

8. Set **SummaryStatisticsLogging** to one of the following options:

   - Enter **YES** to enable Summary Advisor logging.
   - Enter **LOG_OUTER_JOINT_QUERIES_ONLY** to enable Summary Advisor logging only for logical queries that contain outer joins. Consider using this option when the minor performance impact of enabling full Summary Advisor logging is a concern.

9. After applying your changes, release the lock on the domain, as follows:

   a. Return to the BIDomain MBean where group=Service under oracle.biee.admin, Domain:bifoundation_domain, BIDomain.

   b. Display the Operations tab.

   c. Click one of the commit operations.

10. Go to the Oracle Business Intelligence Overview page and click **Restart**.

For upgrading customers, the Summary Advisor parameters are not centrally managed by default. You can set UsageTrackingCentrallyManaged to true as described in the previous procedure, and use the System MBean Browser to update the parameters, or you can manage the Summary Advisor parameters using NQSConfig.INI.

To enable Summary Advisor logging in NQSConfig.INI when central management is disabled for these parameters, follow these steps:

1. On the Oracle BI Server computer, open the NQSConfig.INI file in a text editor. You can find this file at:

   ORACLE_INSTANCE/config/OracleBIServerComponent/coreapplication_obis*n*

Make a backup copy of the file before editing.

2. In the [USAGE_TRACKING] section, update the following parameters:
   - Set `SUMMARY_STATISTICS_LOGGING` to one of the following options:
     - `YES`: Enables Summary Advisor logging.
     - `LOG_OUTER_JOINT_QUERIES_ONLY`: Enables Summary Advisor logging only for logical queries that contain outer joins. Consider using this option when the minor performance impact of enabling full Summary Advisor logging is a concern.
   - Set `SUMMARY_ADVISOR_TABLE_NAME` to the name of the fully-qualified database table for collecting statistics, as it appears in the Physical layer of the Oracle BI repository. For example:

     ```
     SUMMARY_ADVISOR_TABLE_NAME = "My_DB"."DEV_BIPLATFORM"."S_NQ_SUMMARY_
     ADVISOR";
     ```

     The table name you specify must belong to the same database object and connection pool that you are using for usage tracking.
3. Save and close the file.
4. Restart the Oracle BI Server.
5. If you have multiple Oracle BI Server instances, then repeat these steps in each NQSConfig.INI file for all Oracle BI Server instances.

Using the Oracle BI Summary Advisor Wizard

After Summary Advisor statistics have been generated, you can run the Oracle BI Summary Advisor Wizard in the Administration Tool to generate an aggregate specification script that you can then run to create the aggregates. The Summary Advisor Wizard can only be run in online mode.

Alternatively, you can run the Oracle BI Summary Wizard from the command line. See "Using the nqaggradvisor Utility to Run the Oracle BI Summary Advisor" for more information.

Before you run the Summary Advisor Wizard, you must map the target database where you plan to create the aggregates into the Physical layer. To do this, manually create the necessary database, connection pool, and physical schema objects.

To run the Oracle BI Summary Advisor Wizard:

1. Open your repository in the Administration Tool in online mode.
2. It is recommended that you run Model Check Manager to ensure that your repository does not contain modeling problems that will affect Oracle BI Summary Advisor performance and results. See "Using Model Check Manager to Check for Modeling Problems" for more information.

   Note: Because Model Check Manager runs queries against back-end data sources for some checks, it is recommended to run it during off-peak periods. In addition, it can take a long time to run Model Check Manager for large repositories. Use **Filtered by Statistics**, or run it only for selected objects, to improve performance.

3. Select **Tools**, then select **Utilities**.

4. Scroll and select **Oracle BI Summary Advisor**, then click **Execute**.

   The Oracle BI Summary Advisor Wizard is only available if you are running Oracle Business Intelligence on the Oracle Exalytics Machine.

5. On the first page of the Oracle BI Summary Advisor Wizard, Filter Logs - Logical Fact Tables, you can optionally select specific logical fact tables for which to generate Summary Advisor recommendations. By default, all logical fact tables are included.

   If you have used the Oracle BI Summary Advisor Wizard previously and saved your filter criteria, targets, and other options as an XML file, you can click **Load Parameters from File** to load the previously saved options into your current wizard session. See step 11 for information about saving your criteria to a file.

   If your Summary Advisor table (specified in the SummaryAdvisorTableName in the System MBean Browser, or the SUMMARY_ADVISOR_TABLE_NAME parameter in NQSConfig.INI) is empty, Summary Advisor cannot proceed.

   Click **Next** when you are ready to move to the next screen.

6. On the Filter Logs - Time Window screen, you can optionally filter the Summary Advisor logging statistics based on time period. To do this, enter a **Start Date** and **End Date** for statistics that you want to include in the Summary Advisor execution. Click **Update** to refresh the view after entering a time period.

   Click **Next** when you are ready to move to the next screen.

7. On the Filter Logs - Execution Time Threshold screen, you can optionally filter by setting a minimum query time threshold for each logical table source. For example, if you specify 5 seconds for **Minimum Cumulative Time**, only logical table sources with a cumulative total query time of five seconds or greater will be included in Summary Advisor execution.

   Click **Next** when you are ready to move to the next screen.

8. On the Targets screen, select the target container and associated connection pool for aggregates. This is the location where the aggregate tables will be created. You can optionally specify more than one target container.

   Specify the **Database Schema**, **Connection Pool**, and **Capacity** for the target in megabytes, then click **Add Target** to add it to the list. Click **Clear All** to clear the existing list of targets, or click the **Delete** button next to an individual target to remove it.

   Only database schemas for Oracle TimesTen In-Memory Database are currently supported for use with Oracle BI Summary Advisor.

   Click **Next** when you are ready to move to the next screen.

9. On the Select File Location screen, click **Browse** to select the location where the aggregate specification (a SQL script) will be stored.

   Click **Next**.

10. On the Stopping Criteria screen, you can optionally specify run constraints for this set of recommendations, as follows:

    - You can specify the maximum time that Summary Advisor will run before returning results.
    - You can specify a minimum percentage improvement to performance gain of all affected queries in the workload when adding a new aggregate.

Summary Advisor uses an iterative optimization algorithm. For each round of the iteration, Summary Advisor Summary Advisor evaluates a different set of aggregates. When you specify a minimum percentage improvement on this screen, Summary Advisor compares the estimated gain between two consecutive rounds, and stops when the incremental improvement is less than the specified minimum percentage.

The following formula describes the estimated gain between rounds:

```
Estimated Gain = [(total query time at the beginning of the round) -
(total query time at the end of the round)] / (Initial total query time
prior to the first round)
```

For example:

- Initial total query time = 1000s
- End of Round 1:

  Total query time = 500s

  Gain = (1000 - 500)/1000 = 50%
- End of Round 2:

  Total query time = 250s

  Gain = (500 - 250)/1000 = 25%

Click **Next** when you are ready to move to the next screen.

11. On the Miscellaneous screen, you can optionally specify the maximum size of any single aggregate, in megabytes. You can also specify the location of an XML output file that stores the criteria and options from this session to re-use in a future Summary Advisor session.

    Oracle recommends that you select the **Use surrogate keys** option, as surrogate keys improve the performance of queries that use the aggregates created.

    Select **Prefer optimizer estimates** to improve performance during the Summary Advisor process. Selecting this option enables Summary Advisor to use cardinality estimates that originate out of the database query optimizer whenever possible, rather than issuing actual count queries. This option is available for Oracle Database, Microsoft SQL Server, and IBM DB2.

    For Summary Advisor to use database query optimizer estimates, the statistics on the concerned database objects must be available and up-to-date. See the Oracle Database documentation for more information.

    If you do not select this option, Summary Advisor issues count queries to the back-end data sources to obtain row counts (cardinality) for certain queries on the data sources, which can sometimes take a long time to execute.

    Refer the appropriate database documentation for guidelines on how to obtain the best estimates. For example, for Oracle Database, you might want to use the column group feature to improve cardinality estimates for multiple column queries.

Note: A query that attempts to sample a particular grain is not issued by Summary Advisor if an entry for that particular grain already exists in the Summary Advisor cache files, regardless of whether it is an actual count query or a cardinality estimate query.

It is a best practice to remove the Summary Advisor cache files when selecting or deselecting the **Prefer optimizer estimates** option. To do this, delete `NQAggregate.Stats.Cache.txt` and `NQAggregate.LTS.Stats.Cache.txt` in the following directory on the Administration Tool computer:

ORACLE_INSTANCE\bifoundation\OracleBIServerComponent\coreapplication_obisn\aggr

Select **Only include measures used in queries** to include measures used in queries. See "About Measure Subset Recommendations" for more information. If you do not select this option, all measures in a logical fact table are included in the recommendation, including measures that were not used in the workload analyzed by Summary Advisor.

Click **Next**.

12. On the Run screen, click **Run** to generate recommendations using the Summary Advisor process.

    Click **Stop** at any point to stop the process. When Summary Advisor stops or runs to completion, it will display the aggregate recommendations it has found so far.

    When the process completes, click **Next**.

13. On the Filter Aggregates screen, review the current set of aggregate recommendations. You can optionally exclude certain aggregates from the creation process by deselecting the Include option for that row.

    Click **Next**.

14. On the Finish Script screen, review the script that will be generated. If you are satisfied, click **Finish** to save the script.

See "Running the Aggregate Specification Against the Oracle BI Server" for information about using the SQL file to create aggregate tables.

Using the nqaggradvisor Utility to Run the Oracle BI Summary Advisor

As an alternative to using the Oracle BI Summary Advisor Wizard in the Administration Tool, you can use the Oracle BI Server utility `nqaggradvisor` to run the Summary Advisor from the command line.

After Summary Advisor statistics have been generated, use `nqaggradvisor` to generate an aggregate specification script that you can then run to create the aggregates. The `nqaggradvisor` utility is only available if you are running Oracle Business Intelligence on the Oracle Exalytics Machine.

The location of the `nqaggradvisor` utility is:

ORACLE_HOME/bi/bifoundation/server/bin

Syntax

The `nqaggradvisor` utility takes the following parameters.

```
nQAggrAdvisor -d dataSource | -u userName | -o outputFile |
```

```
-c tupleInQuotes [-p password] [-F factFilter] [-z maxSizeAggr]
[-g gainThreshold] [-l minQueryTime] [-t timeoutMinutes]
[-s startDate] [-e endDate] [-C on/off] [-M on/off] [-K on/off]
```

Where:

dataSource is the ODBC data source name for the Oracle BI Server to which you want to connect and run Summary Advisor.

userName is the user name with which to log into the data source. The specified user must have the privilege required to open the Administration Tool in online mode and use the Oracle BI Summary Advisor Wizard.

outputFile is the fully qualified path and file name of the output aggregate specification script.

tupleInQuotes is the aggregate persistence target. You must specify the fully qualified connection pool, fully qualified schema name, and capacity in megabytes.

password is the password corresponding to the *userName*. If not specified, the user is prompted for a password when executing `nQAggrAdvisor`.

factFilter is the fact filter file name. The fact filter file contains the fully qualified names of logical fact tables for which to generate Summary Advisor recommendations. Add each logical fact table's fully qualified name on a separate line. If a fact filter file is not specified, then all logical fact tables in the repository are included in the analysis.

maxSizeAggr is the maximum size of an aggregate in megabytes.

gainThreshold is the minimum percentage improvements to performance gain of all affected queries in the workload required by Summary Advisor when adding a new aggregate in its iterative optimization algorithm. Summary Advisor stops when this value is not satisfied. The default value is `1`.

minQueryTime is the minimum query time threshold in seconds for each logical table source before it is included in the Summary Advisor execution. The default value is `0`.

timeoutMinutes is the maximum time in minutes that Summary Advisor runs before returning results. Specify `0` for unlimited. The default value is `0`.

startDate is the start date for statistics to include in the Summary Advisor execution.

endDate is the end date for statistics to include in the Summary Advisor execution.

`-C` specifies whether to use optimizer estimates. Specify `on` or `off`. The default is `off`.

`-M` specifies which measures to include in the recommendation. Specify `on` to include measures used in the workload. Specify `off` to include all measures in a logical fact table including those measures that were not used in the workload analyzed by Summary Advisor. The default is `off`.

`-K` specifies whether to use surrogate keys. Specify `on` or `off`. The default is `on`.

Examples

The following example shows how to correctly specify the *tupleInQuotes* parameter:

```
nQAggrAdvisor -d "AnalyticsWeb" -u "Administrator" -p "ADMIN" -o "C:\temp\aggr_
advisor.out.txt" -c "DW_Aggr"."Connection Pool","DW_Aggr".."AGGR",1000
```

The following example shows how to correctly specify the *gainThreshold*, *startDate*, and *endDate* parameters.

```
nQAggrAdvisor -d "AnalyticsWeb" -u "Administrator" -p "ADMIN" -o "C:\temp\aggr_
```

```
advisor.out.txt" -F "C:\temp\fact_filter.txt" -g 10 -c "TimesTen_
instance1"."Connection Pool","dbo",2000 -s "2011-05-02 08:00:00" -e "2011-05-07
18:30:00" -C on -M on -K off
```

Using the Aggregate Persistence Wizard to Generate the Aggregate Specification

You can use the Aggregate Persistence Wizard to create the SQL file that will be used to create and load aggregate tables and map them into the metadata. The resulting SQL file must be executed against a running Oracle BI Server.

It is strongly recommended that you use the Aggregate Persistence Wizard because it automatically enforces many of the constraints necessary when generating the aggregate specification. However, you can manually write the aggregate Logical SQL as an alternative to using the wizard. Make sure to follow the guidelines described in "Writing the Create Aggregates Specification Manually" if you choose to write your own aggregates specification.

Before you run the Aggregate Persistence Wizard, you must map the target database where you plan to create the aggregates into the Physical layer. To do this, manually create the necessary database, connection pool, and physical schema objects.

> **Note:** If you are running Oracle Business Intelligence on the Oracle Exalytics Machine, you can use the Summary Advisor feature instead of the Aggregate Persistence Wizard to identify which aggregates will increase query performance and to generate a script for creating the recommended aggregates. See "Using Oracle BI Summary Advisor to Identify Query Candidates for Aggregation" for more information.

To use the Aggregate Persistence Wizard:

1. It is recommended that you run Model Check Manager to ensure that your repository does not contain modeling problems that will affect aggregate creation and performance. See "Using Model Check Manager to Check for Modeling Problems" for more information.

   > **Note:** Because Model Check Manager runs queries against back-end data sources for some checks, it is recommended to run it during off-peak periods. In addition, it can take a long time to run Model Check Manager for large repositories. Use **Filtered by Statistics** (where available), or run it only for selected objects, to improve performance.

2. Open your repository in the Administration Tool, if it is not open already.

   You must run Model Check Manager in online mode. However, you can run the Aggregate Persistence Wizard in either online or offline mode.

3. Select **Tools > Utilities > Aggregate Persistence**, and then click **Execute**.

4. On the Select File Location screen, specify the complete path and file name of the aggregate creation script. You can specify a new or an existing file name.

   Typically, when you run the SQL script against the Oracle BI Server, it creates DDL and runs it against the target database schema to create the aggregate tables, then

loads them from the source, and finally creates the Oracle BI Server metadata so the aggregate navigation feature can use the new tables.

Alternatively, you can select **Generate target DDL in a separate file** if you want the DDL to be stored in a separate file from the Oracle BI Server SQL script. Selecting this option gives you the flexibility to alter the auto-generated DDL and run it independently of the Oracle BI Server. For example, you may want to alter the storage parameter or index settings.

When you select **Generate target DDL in a separate file**, two SQL scripts are generated in the directory you specify in the **Location** field:

- The create aggregates script (*script_name*)
- The prepare aggregates script (*script_name*_DDL)

After selecting **Generate target DDL in a separate file** and completing the wizard steps, you typically do the following:

a. Run the prepare aggregates script against the server. This action creates a DDL file at the following location:

```
ORACLE_INSTANCE\bifoundation\OracleBIServerComponent\coreapplication_obisn\
aggr
```

b. Run the generated DDL file against the target database to create the table.

c. Run the create aggregates script to populate the table.

Click **Next** after you have finished specifying options on the Select File Location screen.

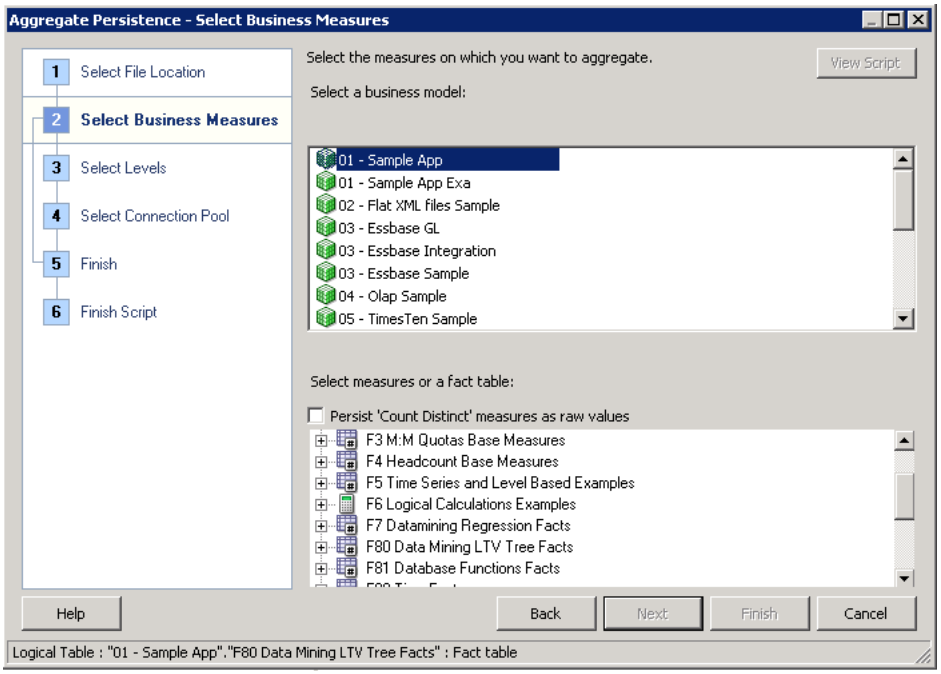
5. In the Select Business Measures screen, select the measures on which you want to aggregate. To do this, select a business model in the upper pane, then select a single fact table or a set of measures in the lower pane. You cannot select measures that span multiple fact tables. Use Ctrl-click to select multiple measures, or use Shift-click to select a range of consecutive measures.

   Optionally select **Persist 'Count Distinct' measures as raw values** to add the `as_raw_values` clause to all valid count distinct measures and to set an aggregation expression override on the corresponding logical column for each system-generated aggregate logical table source. Setting this option enables aggregate persistence to store actual values that are distinct-counted. If you do not select this option, then aggregate persistence stores pre-computed counts for the specified level combinations.

   Note that the **View Script** button is not available during the creation of the first aggregate table block.

   Figure 13–1 shows the Select Business Measures screen.

Figure 13–1 Aggregate Persistence Wizard: Select Business Measures Screen



Click **Next** after you have selected the appropriate measures.

6. In the Select Levels screen, specify the level of aggregation by selecting a logical level for one or more dimensions. You can specify a surrogate key to be used for the fact-dimension join.

   The default join option between the aggregated fact and dimension tables is the primary key defined in the logical level you selected. If the primary key of the level is large and complex, the join to the fact table is expensive, so using a surrogate key is recommended in this case. A surrogate key is an artificially generated key, usually a number. For example, a surrogate key in the level aggregate table would simplify this join, removing unnecessary (level primary key) columns from the fact table and resulting in a leaner fact table.

   Using a surrogate key only changes the query response time, not the logical results of the queries. However, generating the surrogate keys can have the side effect of increasing the aggregate table load time. Therefore, the recommended setting is as follows:

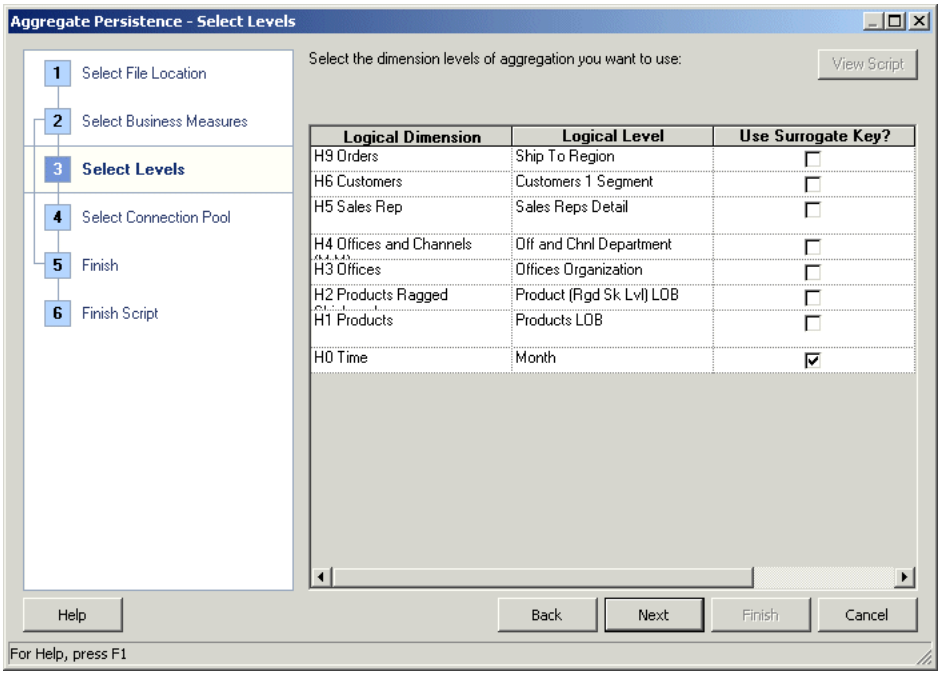
   – If the primary key for the logical level you have selected is already a single, numeric column, you typically should not select the **Use Surrogate Key** option since it may add to load processing time without producing a performance benefit.

   – If the primary key for the logical level you have selected is a text string, or consists of multiple logical columns, you typically should use a surrogate key to improve the performance of the queries that join to that aggregate dimension. However, keep in mind that generating the surrogate key can increase the load time for that aggregate dimension table.

   See "Adding Surrogate Keys to Dimension Aggregate Tables" for additional information about surrogate keys.

   Figure 13–2 shows the Select Levels screen.

Figure 13–2 Aggregate Persistence Wizard: Select Levels Screen



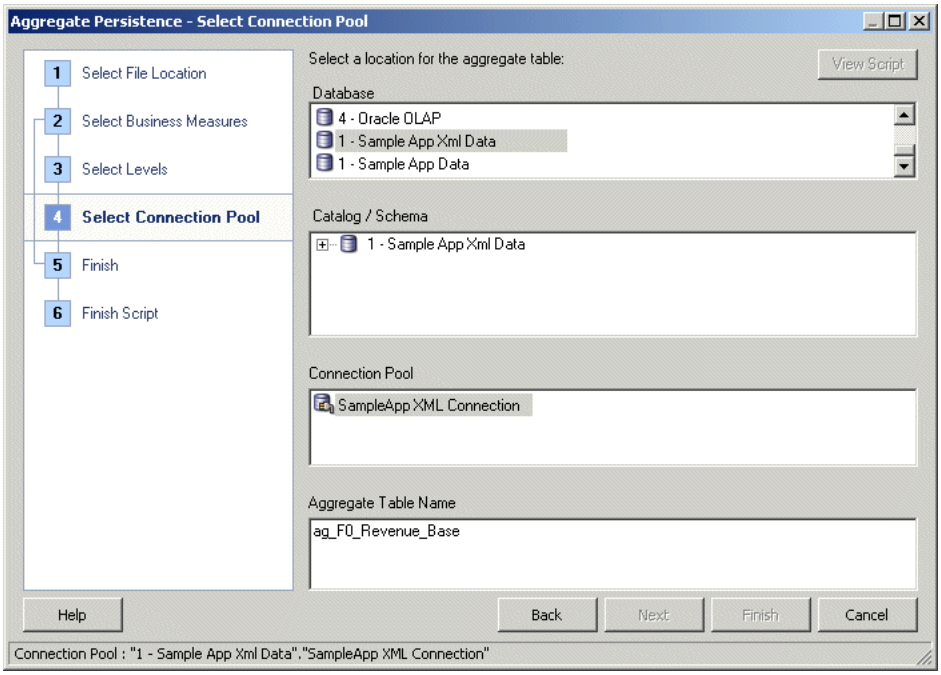
Click **Next** after you have selected the appropriate level of aggregation.

7. In the Select Connection Pool screen, select the appropriate items to specify a location for the aggregate table.

   A default aggregate table name is provided, and a prefix is added to the table name. The default prefix for the generated fact table is `ag`. For tables created for dimension (level) aggregates, the default prefix is `SA_` and can be changed by updating the `AGGREGATE_PREFIX` property in NQSConfig.INI. See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about changing configuration settings.

   Figure 13–3 shows the Select Connection Pool screen.

Figure 13–3 Aggregate Persistence Wizard: Select Connection Pool Screen



Click **Next** after you have provided connection pool information.

8. In the Finish screen, the **View Script** button becomes available for use, and the Logical SQL script appears for your review. Choose whether to define another aggregate (default) or end the wizard, and then click **Next**.
9. In the Finish Script screen, the complete path and file name appears. Click **Finish**.

See "Running the Aggregate Specification Against the Oracle BI Server" for information about using the SQL file to create aggregate tables.

Using Model Check Manager to Check for Modeling Problems

This section describes how to use Model Check Manager to check for modeling problems that might affect Oracle BI Summary Advisor and the aggregate persistence engine.

This section contains the following topics:

- About Model Check Manager
- Running Model Check Manager Using the Administration Tool
- Using the Model Check Manager Dialog
- Checking Models Using the validaterpd Utility

About Model Check Manager

You can use Model Check Manager to check your repository metadata for issues that might affect the success of Oracle BI Summary Advisor or the aggregate persistence engine.

Note that in Oracle BI EE 11*g* Release 1 (11.1.1.9), Oracle improved how Summary Advisor and aggregate persistence handle some issues that in releases previous to 11.1.1.9 prevented the creation of aggregates. See "Aggregate Persistence

Improvements" for more information.

Although the user experience of running Model Check Manager is very similar to running the Consistency Check Manager, there are three key differences between the two tools:

- Unlike the Consistency Check Manager, Model Check Manager requires access to the summary statistics table (when using **Filtered by Statistics**) and back-end data sources for some checks. Because some of the back-end queries can be expensive, it is recommended to run Model Check Manager during off-peak periods.
- Model Check Manager can only be run in online mode.
- Model Check Manager does not make any changes to repository metadata - it only flags possible problems.

Similar to the Consistency Check Manager, Model Check Manager returns both error and warning messages. You must fix errors identified by Model Check Manager, or Oracle BI Summary Advisor recommendations might be incorrect, and the aggregate persistence engine might fail to create aggregates. It is recommended that you fix warnings, but not required. Issues identified by warnings result in suboptimal recommendations from Oracle BI Summary Advisor, or suboptimal performance from the aggregate persistence engine.

Model Check Manager runs parallel queries against the database for better performance. By default, 24 threads are enabled. To change the default number of threads for model check manager, create and set an operating system environment variable called `MODEL_CHECKER_MAX_THREADS`. The maximum number of threads you can specify is 100.

Running Model Check Manager Using the Administration Tool

For Oracle BI Summary Advisor, run Model Check Manager after you have gathered Summary Advisor statistics, but before you run the Oracle BI Summary Advisor Wizard. For aggregate persistence, run Model Check Manager right before you run the Aggregate Persistence Wizard. Alternatively, you can run Model Check Manager to identify problems for selected objects after initial aggregate creation failure.

To run Model Check Manager globally using the Administration Tool, select the **File** menu, then select **Check Models**. Then, choose one of the following options from the submenu:

- **Complete:** Checks all objects in the Business Model and Mapping layer of the Oracle BI repository.
- **Filtered by Statistics:** Checks only fact table objects and associated dimensions in the Business Model and Mapping layer that have been actively queried according to the statistics table. Select this option to speed up the process for large repositories.

  This option is only available on the Oracle Exalytics Machine. If you attempt to filter by statistics on a non-Exalytics system, or if you attempt to filter when the statistics table is not available, a warning appears explaining that Model Check Manager cannot filter by statistics.

  See the following sections for information about setting up the Summary Advisor statistics table:

  – "Setting Up the Statistics Database"

  – "Turning On Usage Tracking"

– "Turning On Summary Advisor Logging"

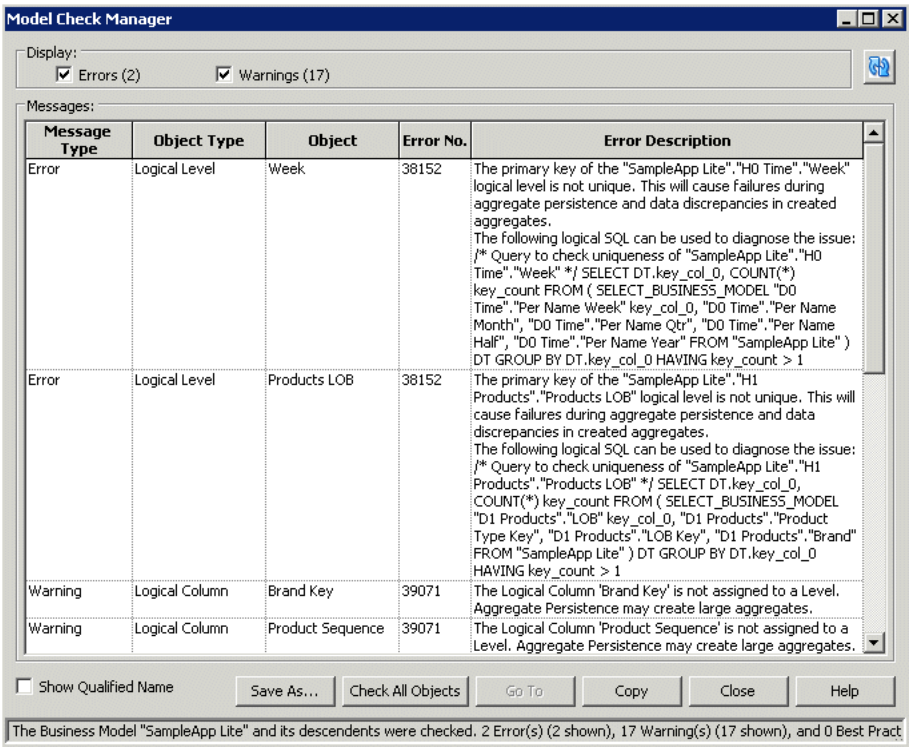
To run Model Check Manager for selected objects using the Administration Tool, right-click one or more business models, dimension objects, or logical fact tables and select **Check Model**. Then, choose **Complete** or **Filtered by Statistics** from the submenu, as described in the preceding list. Note that the **Filtered by Statistics** menu option is only available for fact table objects and business model objects.

When using Model Check Manager with large repositories, it is recommended that you use **Filtered by Statistics** (where available), or run it only for selected objects, to improve performance.

Using the Model Check Manager Dialog

When you run Model Check Manager for one or more objects, the Model Check Manager dialog appears, as shown in Figure 13–4.

Figure 13–4 Model Check Manager



To edit the repository to correct problems, double-click a row to open the properties dialog for that object, or select a row and click **Go To**. Then, correct the problem and click **OK**.

You can also perform the following actions in the Model Check Manager dialog:

- Use the options in the **Display** box to show only errors, only warnings, or both.
- Click the column headings to sort the rows of messages.
- Click **Save As** to save the Model Check Manager results in text, CSV, or XML format.
- Select one or more rows and click **Copy** to copy the messages so that you can paste them in another file such as a spreadsheet. Clicking **Copy** without any rows selected copies all messages.

- To check the model again, click **Check All Objects** to perform a global check. Or, click the **Refresh** button in the top right corner to check only the objects that had errors in the last check.
- Click **Show Qualified Name** to display the qualified names of objects in the Object column.
- View the status bar to see what objects were checked and to see a summary of all the rows displayed.

Checking Models Using the validaterpd Utility

You can check models from the command line using the Oracle BI Server utility `validaterpd` with the `-L` option. Running this utility with `-L` performs the same model checks as Model Check Manager in the Administration Tool. The `validaterpd` utility is available on both Windows and UNIX systems.

To run `validaterpd` in Model Check mode, you must specify the DSN of a running Oracle BI Server.

The location of the `validaterpd` utility is:

ORACLE_HOME/user_projects/domains/bi/bitools/bin

See "Using the validaterpd Utility to Check Repository Consistency" for information about running validaterpd in consistency check mode.

Syntax

The `validaterpd` utility takes the following parameters in Model Check mode:

```
validaterpd -L -D DSN_name -U DSN_user_name [-P DSN_password]
{-O output_txt_file_name |-C output_csv_file_name | -X output_xml_file_name} [-W]
[-S] [-8]
```

Where:

`-L`: Specifies Model Check mode.

`-D`: The DSN of a running Oracle BI Server.

`-U`: The user name for the Oracle BI Server DSN.

`-P`: The password for the Oracle BI Server DSN.

The password argument is optional. If you do not provide the password argument, you are prompted to enter the password when you run the command. To minimize the risk of security breaches, Oracle recommends that you do not provide password arguments either on the command line or in scripts. Note that the password argument is supported for backward compatibility only, and will be removed in a future release. For scripting purposes, you can pass the password through standard input.

`-O`: Use this option to output the results in a text file.

`-C`: Use this option to output the results in a CSV file.

`-X`: Use this option to output the results in an XML file.

`-8`: Use this option to specify UTF-8 output (optional).

`-W`: You can optionally include a whitelisted objects file. This text file specifies a limited number of logical objects that you want to check. Enter the fully-qualified name of each logical object on a single line. If `-W` is not specified, all logical objects are checked.

`-S`: Use this option to check only objects that have been actively queried according to the statistics table. If `-S` is not specified, all objects are checked. If `-W` is also specified, the whitelist file can only contain business models and logical fact tables (other objects are not checked). This option is only available on the Oracle Exalytics Machine.

Examples

```
validaterpd -L -D DSNName -U Username -O results.txt
Give password: my_dsn_password
```

The preceding example connects to an RPD using the DSNName connection, checks all models in the RPD, and writes output to results.txt.

```
validaterpd -L -D DSNName -U Username -O results.txt -W whitelist.txt -S
Give password: my_dsn_password
```

The preceding example connects to an RPD using the DSNName connection, performs a model check, and writes output to results.txt. Only objects listed in whitelist.txt are checked. Furthermore, because -S is specified, only objects that have been actively queried according to the statistics table are checked.

Note that when -W and -S are both specified, the whitelist can only contain business models and logical fact tables. Other objects are not checked.

Writing the Create Aggregates Specification Manually

If you choose not to use the Aggregate Persistence Wizard to create the script file, you can write the file manually. However, Oracle recommends that you use the Aggregate Persistence Wizard.

If you do not want the Oracle BI Server to modify your databases during aggregate creation, then you can specify this in the Aggregate Persistence Wizard by selecting the option **Generate target DDL in a separate file**. The Aggregate Persistence Wizard will create a DDL file (the "prepare aggregates" script) that you can use to create the empty aggregate tables. After this, you need to run the "create aggregates" script to populate the aggregate tables. This option provides some flexibility in case the database access to create tables is restricted. Note that you must run the prepare aggregates script before you run the create aggregates script.

This section contains the following topics:

What Constraints Are Imposed During the Create Process?

The following constraints are imposed during the create process:

- **Valid measures.** A valid measure must have a valid aggregation rule. The following constraints apply to level-based measures:
  - If the level is grand total alias, then that dimension must not be present in the list of levels for that aggregate specification.
  - Any other level defined for this measure must be present in the list of levels for that aggregate specification.

If the above constraints are not met, then the entire aggregate specification is discarded. In addition, a measure is ignored by the create process if any of the following conditions are true:

- Measure is mapped to a session or repository variable.
- Measure is a derived measure.
- Measure has a default aggregation rule of None.

Measures that are ignored do not necessarily affect the aggregate specification. The remaining measures are used to create the aggregate.

- **Valid levels.** A valid level must have a valid primary key. If a level is invalid, the aggregate specification is be discarded. Also, attributes of a level or its primary key are ignored if any of the following conditions are true:
  - Attribute is mapped to session or repository variables.
  - Attributes are not from the same logical table.
- **Valid aggregate specification.** A valid aggregate specification has the following properties:
  - Name length is between 1 and 18 characters (inclusive).
  - At least one valid level must be specified.
  - At least one valid measure must be specified.
  - Must have a valid connection pool.
  - Must have a valid output container (database/catalog/schema).
  - Connection pool and container must belong to the same database.
  - Only one level per dimension can be specified.
  - Measures can only be from the same fact table.
  - All logical components of the specification must be from the same subject area.

  An aggregate specification is ignored if the name already exists in the output container because level aggregates are scoped by the entire database. However, if different catalogs or schemas are specified for the same fact aggregate name, it is allowed to have multiple facts with the same name but different scope in the same database.

  Note that the aggregate specification is discarded if any dimension is not joined to a fact.

Writing the Create Aggregates Specification

All metadata names (except for logical fact columns) are fully qualified. There are two modes of operation: Create and Delete. It is strongly recommended that you place all aggregate specifications under a single Create Aggregates statement.

See "Adding Surrogate Keys to Dimension Aggregate Tables" for information about creating aggregates with surrogate keys.

Use the following guidelines when writing the aggregate specification.

Delete Statement for Aggregate Specification

Begin the script file with a Delete statement. It is essential to delete system-generated aggregates before creating new ones. This ensures that data is consistent and removes

invalid or incomplete aggregates before you run the Create operation. The following statement is the syntax for deleting aggregates:

```
Delete aggregates [list of fully qualified physical table names];
```

For example:

```
Delete aggregates "src".."INCR"."fact_1", "src".."INCR"."fact_2";
```

You can optionally include a comma-separated list of physical tables to delete. The tables you include must be system-generated (by a previous run of the aggregate creation script). Any dimension tables joined to listed fact tables are also deleted.

Note that if a dimension table is joined to more than one fact table, it will not be deleted unless the other joined table is also listed.

In addition to fact tables, you can also use the Delete statement to delete orphan dimension tables (that is, dimension tables that are not joined to any other fact table). Orphan dimension tables sometimes occur when aggregate creation fails.

The Delete statement also removes the logical key and logical column that were added to the time dimension's logical table when chronological keys were added for aggregate persistence.

Create Statement for Aggregate Specification

The Create statement should follow the Delete statement. The following is the syntax for creating aggregates:

```
Create|Prepare aggregates
aggr_name_1
for  logical_fact_table_1 [(logical_fact_column_1, logical_fact_column_2, count_
distinct_logical_fact_column_1 as_raw_values, ...)]
at levels (level_1, level_2, …)
using connection pool connection_pool_name_1
in schema_name_1
[ ,aggr_name_2
for logical_fact_table_3 [(logical_fact_column_5, logical_fact_column_2,…)]
at levels (level_3, level_2, …)
using connection pool connection_pool_name_2
in schema_name_2] ;
```

Note that `as_raw_values` must accompany a count-distinct measure with a simple aggregate rule. A simple aggregate rule has only one rule, which is not dimension-based.

Multiple Aggregates in Aggregate Specification

To specify multiple aggregates in a single Create Aggregates statement, follow these guidelines:

- Ensure that each of the multiple aggregate specifications are separated by a comma, and the entire aggregate creation script is terminated with a semicolon.
- In this file, only one Delete Aggregates statement should be specified at the beginning. Make sure that only one delete is issued per ETL run (unless a reset is called for).

Caution: Any aggregate scripts that are run after the first one should not have a Delete Aggregates statement, or all previously created aggregates are removed.

Where Clause for Aggregate Specification

You can add an optional Where clause to the Create statement. The Where clause filters the data that you want to aggregate and creates fragmented aggregates, or aggregates for only a fragment of data in the base fact table. The Where clause also sets the Fragmentation content field located on the Logical Table Source dialog. In most cases, the creation of fragmented aggregates maximizes query acceleration while minimizing the cost of creating and maintaining the aggregate.

Note the following examples of when you would use fragmented aggregates:

- If you are working with the time dimension and want your aggregates to include data only from the last three years.
- If your company reports primarily on revenue in the United States and wants the aggregates to include only United States data.

The following is an example of a valid Create statement with the Where clause:

```
Create Aggregates
Revenue_By_Year
for  "sales"."sales" at levels("sales".timedim.year)
where("sales"."time"."calendar year"=2007)
using connection pool aggrtarget.cp1
in aggrtarget..schema1
```

Logical Column Requirements

The logical column that you specify in the Where clause must meet the following requirements:

- The logical column must belong to a dimension.
- If the logical column belongs to a dimension included in the aggregate specification, then it must be at or above the level of the aggregate.
- If you use `operational_oper`, then the logical column's data type must match the constant's data type in `inlist`.

  If you use `inclusion_oper`, then the logical column's data type must match all the constants' data type in `inlist`.

Where Clause Grammar

The grammar for the Where clause in the create aggregate specification is a subset of the Where grammar for a Logical SQL filter. Therefore the grammar that Oracle BI Server supports for the create aggregate specification differs slightly from the Logical SQL filter.

Note the following Create statement and its Where clause:

```
create aggregate aggr1 for fact1 at levels(11,12...) where (filter_list) using
....
```

The following are the acceptable grammar rules:

filter_list ::= filter logical_oper filter_list

| *filter*

| *'(' filter_list ')'*

filter ::= logical_column relational_oper constant

| *logical_column inclusion_oper '(' inlist ')'*

relational_oper ::= '=' | '!=' | '<' | '>' | '<=' | '>=' | 'like'

inlist ::= constant ',' inlist

| *constant*

logical_oper ::= 'and' | 'or'

inclusion_oper ::= 'in' | 'not in'

Adding Surrogate Keys to Dimension Aggregate Tables

The join option default between fact and level aggregate tables uses primary keys from the level aggregate. If the primary key of the level is large and complex (composite of many columns), then the join to the fact table is expensive. A surrogate key is an artificially generated key, usually a number. A surrogate key, in the level aggregate table, simplifies this join and removes unnecessary columns (level primary key) from the fact table, resulting in a smaller-sized fact table. Adding surrogate keys to the dimension (level) aggregate tables can simplify joins to the fact tables and might improve query performance. Additionally, a surrogate key makes sure that each aggregate table has a unique identifier.

There might be cases in which a level is shared among multiple fact tables. One fact might use surrogate keys, and another might use primary keys from the dimension aggregate. The following are some options for resolving this issue:

- Set a metadata property for levels that indicates whether to use surrogate keys or primary keys.
- Always create a surrogate key for a level aggregate (relatively low cost operation). Then, decide later if the fact aggregate should join to it using a surrogate or primary key.

An alternative to specifying the join type for each dimension is to specify if surrogate keys should be used for the entire star. This would result in simpler syntax, but would also restrict the available user options and slow the aggregate creation process.

About the Create/Prepare Aggregates Syntax

The following syntax for create/prepare aggregates contains the change for [Using_ Surrogate_Key]. The surrogate key option can be specified for each level. If unspecified, the fact and dimension tables are joined using the primary key from the level aggregate.

```
Create|Prepare aggregates
aggr_name_1
[file output_file_name]
for  logical_fact_table_1 [(logical_fact_column_1, logical_fact_column_2,…)]
at levels (level_1 [Using_Surrogate_Key], level_2, …)
using connection pool connection_pool_name_1
in schema_name_1
[ ,aggr_name_2
for logical_fact_table_3 [(logical_fact_column_5, logical_fact_column_2,…)]
at levels (level_3, level_2, …)
using connection pool connection_pool_name_2
```

```
in schema_name_2] ;
```

About Surrogate Key Output from Create/Prepare Aggregates

The changes to the current process are restricted to the physical metadata layer in the repository and the database.

When you use the `Using_Surrogate_Key` join option, the following describes the results:

- For a level aggregate, the following occurs:
  - In the physical metadata, the following occurs:
    * The level aggregate table has a new column called *`levelName_upgradeIDSK`* (check for collisions). This is the surrogate key column for the dimension aggregate. Note that *`levelName`* is truncated if the total number of characters exceeds 18.
  - In the database, the following occurs:
    * The level aggregate table also has a corresponding column called *`levelName_upgradeIDSK`*. Again, *`levelName`* is truncated if the total number of characters exceeds 18.
    * It can be populated using `RCOUNT()`.
- For a fact aggregate, the following occurs:
  - In the physical metadata, the following occurs:
    * The fact aggregate table no longer contains columns from the level's primary keys.
    * Instead, a new column that corresponds to the level aggregate's surrogate key is added to the table.
    * The type of this column is identical to the level's surrogate key.
    * The column has the same name as that in the level aggregate (check for collisions).
    * The fact table and the level table are joined using this surrogate key only.
  - In the database, the following occurs:
    * The fact aggregate table also has the corresponding surrogate key.
    * It is populated using new capabilities to be available through Populate.

Running the Aggregate Specification Against the Oracle BI Server

This topics describes how to run the aggregate specification script against the Oracle BI Server. Before you run the script, you must create an ODBC DSN for the Oracle BI Server and ensure that the correct log level is set. Then, you can execute the script using `nqcmd` as a user who is a member of the BI Administrators group.

To run the aggregate specification script:

1. To run the aggregate creation script, you must connect directly to a DSN for a running Oracle BI Server and not to a clustered DSN. Note the following:
   - **Single-node cluster:** Because the DSN created upon install for each Oracle BI Server is clustered by default, even for a single-node deployment, you must

manually create a DSN for the Oracle BI Server to run the aggregate specification against.

- **Multi-node cluster:** You must run the aggregate specification directly against the master Oracle BI Server. Create a non-clustered DSN for the master Oracle BI Server to run the aggregate specification against. You can use the Cluster Manager in the Administration Tool in online mode to determine which Oracle BI Server is the master.

See "Integrating Other Clients with Oracle Business Intelligence" in *Oracle Fusion Middleware Integrator's Guide for Oracle Business Intelligence Enterprise Edition* for information about how to create an ODBC DSN for the Oracle BI Server.

2. It is recommended that you set an appropriate logging level before executing the script. Trace logs are logged to nqquery.log if the logging level is at least 2. The logging events include the aggregate execution plan and the order in which the aggregates are created and deleted. Higher logging levels provide more details about the query and execution plans - for example, specify logging level 4 or higher to see the queries being executed. Error logs are logged to nqquery.log if the logging level is at least 1, and to nqserver.log regardless of the logging level.

   First, set the logging level using one of the following methods:

   - Set the logging level in the repository user object for the user who will run the script. See "Managing the Query Log" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.
   - Create and set the LOGLEVEL system session variable. LOGLEVEL overrides logging levels set on individual users. See "Creating Session Variables" for more information.

   Then, you must also edit the aggregate creation script directly to set the logging level as a request variable in each delete aggregates or create aggregates statement. For example:

   ```
   set variable LOGLEVEL=7 : delete aggregates;

   set variable LOGLEVEL=7 : create aggregates... ;
   ```

   Note that you must use a colon as a delimiter when setting request variables using `nqcmd`.

3. As a user in the BI Administrators group, use `nqcmd` to connect to the non-clustered DSN for the Oracle BI Server that you created in step 1. Then, run the aggregate specification script.

   See "Using nqcmd to Test and Refine the Repository" for more information about running `nqcmd`.

Note: In a clustered environment, the aggregate specification script performs a rolling restart of the slave Oracle BI Servers in the background. It is a best practice to avoid making other configuration changes in Fusion Middleware Control or the configuration files while running the aggregate persistence script. Because only the slave servers are restarted in the rolling restart, a situation might result where the master Oracle BI Server has a different set of configuration settings loaded than the slave Oracle BI Servers. If this occurs, restart the master Oracle BI Server.

After executing the SQL script, aggregates are created and persisted in the Oracle BI Server metadata, as well as in the back-end databases.

Note that when a series of aggregates are being created, and the creation of one aggregate fails, the aggregate persistence engine skips creation of the failed aggregate and its dependencies and proceeds to the next aggregate in the list.

Check the log files to identify failed aggregates. If orphan aggregate dimensions (those not joined to any other fact table) were created, use the Delete Aggregates command to delete them.

Life Cycle Use Cases for Aggregate Persistence

Table 13–2 summarizes the user tasks to persist aggregates for different life cycle use cases.

Note that these use cases focus on operations against single or multiple aggregate persistence targets, and do not describe operations for single or multiple-node deployments. For the most part, user tasks are the same for both single-node deployments and multiple node deployments. The only difference is that in a clustered deployment, you must connect to the master Oracle BI Server, and a rolling restart of the slave servers is performed in the background. See "Running the Aggregate Specification Against the Oracle BI Server" for more information.

Table 13–2 Life Cycle Use Cases for Aggregate Persistence

Number	Use Case	Description
1	Creating aggregates for a single aggregate persistence target	To only create aggregates, modify the aggregate creation script to remove the delete aggregates statement at the beginning. Then, use `nqcmd` to run the script.

2	Deleting aggregates for a single aggregate persistence target	To delete aggregates, use `nqcmd` to run the delete aggregates statement directly, as follows:
		`Delete aggregates [list of fully qualified physical fact table names];`
		For example:
		`Delete aggregates;`
		or
		`Delete aggregates "src".."INCR"."fact_1", "src".."INCR"."fact_2";`

3	Refreshing aggregates for a single aggregate persistence target	Use `nqcmd` to run the aggregate creation script, which contains statements to first delete, then create the aggregates. Alternatively, you can manually delete the aggregates as described in use case 2, then create aggregates as shown in use case 1. This manual method is useful for situations where you want to delete all aggregates, but the aggregate creation script only specifies certain aggregates to be deleted.

Table 13–2 (Cont.) Life Cycle Use Cases for Aggregate Persistence

Number	Use Case	Description

4	Creating aggregates for multiple redundant aggregate persistence targets	To create aggregate clones on multiple targets, modify the aggregate creation script to copy the create aggregates statements as many times as you have targets.
		For example, say you have a script containing the following create aggregates statement:
		`set variable LOGLEVEL=7 : create aggregates "myfactaggr" for "FACT_1"("MEASURE_1") at levels ("INCR"."DIM1_LEVEL1Dim"."DIM1_LEVEL1 Detail") using connection pool "tgt1"."cp" in "tgt1".."double1";`
		You would then copy the block, paste it below the first block, and modify the connection pool and schema information for your second target. For example:
		`set variable LOGLEVEL=7 : create aggregates "myfactaggr" for "FACT_1"("MEASURE_1") at levels ("INCR"."DIM1_LEVEL1Dim"."DIM1_LEVEL1 Detail") using connection pool "tgt2"."cp" in "tgt2".."double2";`
		After you have copied and modified the block for all your targets, save the script. Then, use `nqcmd` to run the aggregate creation script.

5	Deleting aggregates for multiple aggregate persistence targets	To delete aggregates on multiple targets, use `nqcmd` to run the delete aggregates statement directly for the affected fact tables. For example:
		`set variable LOGLEVEL=7 : delete aggregates "tgt1".."double1"."myfactaggr";`
		`set variable LOGLEVEL=7 : delete aggregates "tgt2".."double2"."myfactaggr";`

6	Refreshing aggregates for multiple redundant aggregate persistence targets	See "Using Double Buffering to Refresh Highly Available Aggregates" for information about this use case.
7	Refreshing aggregates for multiple partitioned aggregate persistence targets	In some cases, you might have different aggregates partitioned across multiple targets. This approach maximizes memory use, but does not provide highly available aggregates. To refresh partitioned aggregates, use one of the following methods as appropriate for your deployment: ■ Run the Aggregate Persistence Wizard multiple times against the different targets to generate a set of aggregate creation scripts, then run the scripts. ■ If you are running Oracle Business Intelligence on the Oracle Exalytics Machine, run Oracle BI Summary Advisor and specify multiple targets in the Targets screen. Then, run the aggregate creation script.

Using Double Buffering to Refresh Highly Available Aggregates

When you have aggregate clones across multiple aggregate persistence targets, you can use double buffering to avoid downtime when refreshing the aggregates.

To set up double buffering, you manually call the aggregate create and delete SQL statements in a way that controls the refresh.

You start by deleting aggregates on the first target. Next, you create the aggregates on the first target, specifying the targets where aggregates have not yet been deleted as inactive schemas, so that the old data is not used in the refresh. Then, you repeat this process for each target. Note that you do not need to specify inactive schemas when refreshing the last target because by that point, the data in the other schemas has already been refreshed.

To specify inactive schemas, set the request variable INACTIVE_SCHEMAS before the create aggregates statement. For example:

```
set variable INACTIVE_SCHEMAS='"tgt2".."double2"' :
```

Only specify schemas that have not yet been refreshed as inactive schemas. Do not specify a schema that has already been refreshed or that you have just deleted.

To specify multiple inactive schemas, use a comma-separated list. Make sure there are no spaces in the list.

Example 13–1 illustrates how to use double buffering to refresh aggregates on two targets.

Example 13–1 Refreshing Aggregate Clones on Two Targets

Assume that you have the following aggregate clones on targets tgt1 and tgt2:

```
"myfactaggr"
for "FACT_1"("MEASURE_1")
at levels ("INCR"."DIM1_LEVEL1Dim"."DIM1_LEVEL1 Detail")
using connection pool "tgt1"."cp"
in "tgt1".."double1",

"myfactaggr"
for "FACT_1"("MEASURE_1")
at levels ("INCR"."DIM1_LEVEL1Dim"."DIM1_LEVEL1 Detail")
using connection pool "tgt2"."cp"
in "tgt2".."double2";
```

To make sure that at least one aggregate clone is available during the refresh, follow these steps:

1. Delete the aggregate clone for the first target:

   ```
   set variable LOGLEVEL=7 : delete aggregates "tgt1".."double1"."myfactaggr";
   ```

2. Create the aggregate for the first target, making sure to specify the second target as an inactive schema so that the data is not used in the refresh:

   ```
   set variable LOGLEVEL=7, INACTIVE_SCHEMAS='"tgt2".."double2"' : create
   aggregates
   "myfactaggr"
   for "FACT_1"("MEASURE_1")
   at levels ("INCR"."DIM1_LEVEL1Dim"."DIM1_LEVEL1 Detail")
   using connection pool "tgt1"."cp"
   in "tgt1".."double1";
   ```

3. Delete the aggregate clone for the second target:

   ```
   set variable LOGLEVEL=7 : delete aggregates "tgt2".."double2"."myfactaggr";
   ```

4. Create the aggregate for the second target. Because the first target has already been refreshed, do not specify any inactive schemas:

```
set variable LOGLEVEL=7 : create aggregates
"myfactaggr"
for "FACT_1"("MEASURE_1")
at levels ("INCR"."DIM1_LEVEL1Dim"."DIM1_LEVEL1 Detail")
using connection pool "tgt2"."cp"
in "tgt2".."double2";
```

Note: When you have aggregate clones across multiple aggregate persistence targets, the additional instances are hot-spares that take over the query load while the initial instance is being refreshed. The aggregate clones are not used for load balancing the incoming queries.

Creating Aggregates on TimesTen Sources

This section describes configuration steps and features related to aggregate creation on TimesTen sources. This section contains the following topics:

- Enabling PL/SQL for TimesTen
- Creating Aggregates with Compressed Tables
- Enabling Performance Enhancement Features for TimesTen

See also *Oracle Fusion Middleware Installation and Administration Guide for Oracle Exalytics In-Memory Machine* for specific instructions on setting up TimesTen sources on the Oracle Exalytics Machine.

Enabling PL/SQL for TimesTen

To create aggregates on TimesTen sources, you must ensure that PL/SQL is enabled for the instance, and that the PL/SQL first connection attribute PLSQL is set to 1.

You can enable PL/SQL at install time, or run the ttmodinstall utility to enable it post-install. See *Oracle TimesTen In-Memory Database Reference* for more information.

Creating Aggregates with Compressed Tables

If you want to create aggregates with compressed tables in TimesTen, turn on the database feature `COMPRESSED_COLUMNS` in the Features tab of the Database dialog in the Administration Tool.

See "Specifying SQL Features Supported by a Data Source" for more information about using the Features tab.

Enabling Performance Enhancement Features for TimesTen

If you are creating aggregates in TimesTen Release 11.2.2.3 or later, then you can enable features to improve performance.

The features that you can enable to improve performance are:

- Disable Redo Logging
- Create Indexes in Parallel
- Perform database checkpoints in the background

To enable these features:

1. Open opmn.xml for editing. You can find opmn.xml at:

   ORACLE_INSTANCE/config/OPMN/opmn/opmn.xml

2. Locate the ias-component tag for the Oracle BI Server process. For example:

   ```
   <ias-component id="coreapplication_obis1" inherit-environment="true>
   ```

3. Under the <environment> subtag, update the following TimesTen variables:

   ```
   <variable id="ORACLE_BI_TT_DISABLE_REDO_LOGGING" value="1"/>
   <!-- This disables redo-logging, enabling faster creation of aggregates.
   '0' if you wish to disable this feature. -->

   <variable id="ORACLE_BI_TT_BACKGROUND_CHECKPOINT_INTERVAL" value="10"/>
   <!-- This changes how often TimesTen will flush its data to disk. If this
   element is missing, the default is every 10 seconds. If explicitly set, it
   will flush to disk every N seconds, 10 in this example. '0' will disable
   background flushing. Enabling background flushing speeds up creation of
   aggregates, by avoiding a large blocking flush at the end of the aggregate
   creation process. -->
   ```

4. Save and close the file.
5. Restart OPMN.
6. Repeat these steps on each computer that runs the BI Server process. If you are running multiple BI Server instances on the same computer, then be sure to update the ias-component tag appropriately for each instance in opmn.xml (for example, ias-component id="coreapplication_obis1", ias-component id="coreapplication_obis2", and so on).

If you are creating aggregates in TimesTen Release 11.2.2.5 or later, then you can enable parallel insert. Parallel insert allocates multiple threads to accommodate the insertion of rows into a single table. This enhancement improves the performance of aggregate creation in TimesTen. Perform the following procedure in addition to the preceding procedure.

To enable the parallel insert feature:

1. Open opmn.xml for editing. You can find opmn.xml at:

   ORACLE_INSTANCE/config/OPMN/opmn/opmn.xml

2. Locate the ias-component tag for the Oracle BI Server process. For example:

   ```
   <ias-component id="coreapplication_obis1" inherit-environment="true>
   ```

3. Under the <environment> subtag, update the following TimesTen variables:

   ```
   <variable id="ORACLE_BI_AGGR_PARALLEL_INSERT" value="ENABLE"/>
   <!-- This enables parallel insert using TimesTen's default thread count
   allocations, providing faster creation of aggregates. Set the value to an
   integer >=2 to specify a custom thread count for TimesTen. AggrPersist
   will request that TimesTen allocate the input number of threads for use in
   performing parallel inserts. If the number of thread counts exceeds
   TimesTen's maximum number, then TimesTen and AggrPersist will use the
   maximum number instead. -->

   <variable id="ORACLE_BI_AGGR_DIMENSION_TO_FACT_RATIO = 1:4"/>
   <!-- Optional. This specifies the ratio of threads to dedicate to a
   dimension table versus a fact table. Using a ratio allows thread allocation
   to scale to the size of the TimesTen ThreadPool. There is one thread pool
   per TimesTen connection pool. If this variable is omitted or is invalid,
   ```

```
the value defaults to 1:4. Suppose you specifiy a 1:4 ratio and there is
one Dimension and one Fact table in an Aggregate Persistence batch of
tables. When a TimesTen Connection Pool's thread pool has 5 threads, then
one Dimension table will have one thread and one Fact table will have four
threads. And, when a TimesTen Connection Pool's thread pool has ten
threads, then one Dimension table will have two threads and one Fact table
will have eight threads. -->
```

4. Save and close the file.
5. Restart OPMN.
6. Repeat these steps on each computer that runs the BI Server process. If you are running multiple BI Server instances on the same computer, then be sure to update the ias-component tag appropriately for each instance in opmn.xml (for example, ias-component id="coreapplication_obis1", ias-component id="coreapplication_obis2", and so on).

Troubleshooting Aggregate Persistence

This section describes how to troubleshoot the Summary Advisor and aggregate persistence process.

This section includes the following topics:

- About Aggregate Persistence Errors
- Avoiding Aggregate Persistence Errors

About Aggregate Persistence Errors

Occurrences such as a network failure, no disk space on the database, or a bad aggregate request, will result in aggregate persistence errors.

When a series of aggregates are being created, and the creation of one aggregate fails, the aggregate persistence engine skips creation of the failed aggregate and its dependencies and proceeds to the next aggregate in the list. Check the log files to identify failed aggregates.

If there are errors, you must remove the failed aggregates in one of the following ways:

- Manually remove the aggregates from the metadata and the database. To identify the aggregate metadata, you can query the repository using the **Is System Generated** filter for physical tables and logical table sources. See "Querying the Repository" for more information.
- Automatically remove the failed aggregates using the Delete Aggregates specification. In particular, use this technique to remove any orphan aggregate dimensions (those not joined to any other fact table).

Avoiding Aggregate Persistence Errors

Run Model Check Manager to ensure that your repository does not contain modeling problems that will affect Oracle BI Summary Advisor and aggregate persistence performance and results. See "Using Model Check Manager to Check for Modeling Problems" for more information.

14

Applying Data Access Security to Repository Objects

This chapter provides information about the different types of data access security available for Oracle BI repository objects and explains how to apply them.

Data access security controls rights to view and modify data. You can use several different methods of data access security with Oracle Business Intelligence: row-level security (implemented either in the repository or in the database), object permissions, and query limits.

Other security tasks, including setting up SSL connections, managing users, groups, and application roles, setting up custom LDAP servers, and managing custom authenticators, are covered in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*. Note that you must create users and application roles before you can implement data access security.

You should plan to implement data access security in the Administration Tool in online mode. If you must perform data access security tasks in offline mode, be sure to read "About Applying Data Access Security in Offline Mode" first.

Data access security auditing is covered by the Oracle Business Intelligence usage tracking feature. See "Managing Usage Tracking" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

This chapter contains the following topics:

About Data Access Security

After developing your metadata repository, you need to set up your data security architecture to control access to source data.

Data access security accomplishes the following goals:

- To protect business data queried from databases
- To protect your repository metadata (such as measure definitions)
- To prevent individual users from hurting overall system performance

Oracle Business Intelligence supports three types of data security: row-level security, object permissions, and query limits (governors). Object permissions and query limits are set up in the repository and are enforced only by the Oracle BI Server. Row-level data security, however, can be implemented and enforced in both the repository, and in the database.

Even if you choose to implement row-level security in the database, you should still set up object permissions and query limits in the repository. Although it is possible to provide database-level object restrictions on individual tables or columns, objects to which users do not have access are still visible in all clients, even though queries against them will fail. It is better to set up object permissions in the repository, so that objects to which users do not have access are hidden in all clients.

Because a variety of clients can connect to the Oracle BI Server, you cannot implement or enforce data security in Oracle BI Presentation Services. Oracle BI Presentation Services provides an extensive set of security controls that let you set up privileges to access functionality in the Oracle Business Intelligence user interface, as well as dashboards and analyses objects. However, Oracle BI Presentation Services does not provide data access security. If you only implement security controls in Oracle BI Presentation Services, you will be exposed to SQL injection hacker attacks and other security vulnerabilities. You must provide object-level security in the repository to create rules that apply to all incoming clients.

See *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition* for more information about the security controls available in Oracle BI Presentation Services.

Where to Find Information About Security Tasks

Oracle Business Intelligence security tasks are covered in this guide, in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*, and in other sources.

Table 14–1 summarizes the Oracle Business Intelligence security tasks and where to go for more information.

Table 14–1 Security Tasks in Oracle Business Intelligence

Task	Location
Setting up user authentication with the default authentication provider or an alternative authentication provider	"Managing Security Using the Default Security Configuration" in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*
Creating and managing users and groups in the default authentication provider	"Managing Users and Groups in the Embedded WebLogic LDAP Server" in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*

Table 14–1 (Cont.) Security Tasks in Oracle Business Intelligence

Task	Location
Creating application roles and managing policies in the default policy store	"Managing the Policy Store" in *Oracle Fusion Middleware Application Security Guide*
Viewing and understanding the default Oracle Business Intelligence permissions used with application roles in the policy store	"Default Permissions" in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*
Applying data access security in offline mode and setting up placeholder application roles	"About Applying Data Access Security in Offline Mode"
Setting up row-level data security	"Setting Up Row-Level Security"
Setting repository object permissions	"Setting Up Object Permissions"
Setting query limits (governors)	"Setting Query Limits"
Viewing users in the Administration Tool	"About the List of Users in the Administration Tool"
Setting up single sign-on (SSO)	"Enabling SSO Authentication" in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*
Enabling SSL communication	"SSL Configuration in Oracle Business Intelligence" in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*
Managing custom authenticators	"Authenticating by Using a Custom Authenticator Plug-In" in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*

Setting Up Row-Level Security

You can choose to set up row-level security in the repository, or in the database.

Implementing row-level security in the repository provides many benefits, including the following:

- All users share the same database connection pool for better performance
- All users share cache for better performance
- You can define and maintain security rules that apply across many federated data sources

Implementing row-level security in the database, in contrast, is good for situations where multiple applications share the same database. Note that even when you design and implement row-level security in the database, you should still define and apply object permissions in the repository.

Although it is possible to set up row-level security in both the repository and in the database, you typically do not enforce row-level security in both places unless you have a particular need to do so.

This section contains the following topics:

- Setting Up Row-Level Security (Data Filters) in the Repository

- Setting Up Row-Level Security in the Database

Setting Up Row-Level Security (Data Filters) in the Repository

Data filters are a security feature that provide a way to enforce row-level security rules in the repository. Data filters are set up in the repository using the Administration Tool and are applied for a particular application role.

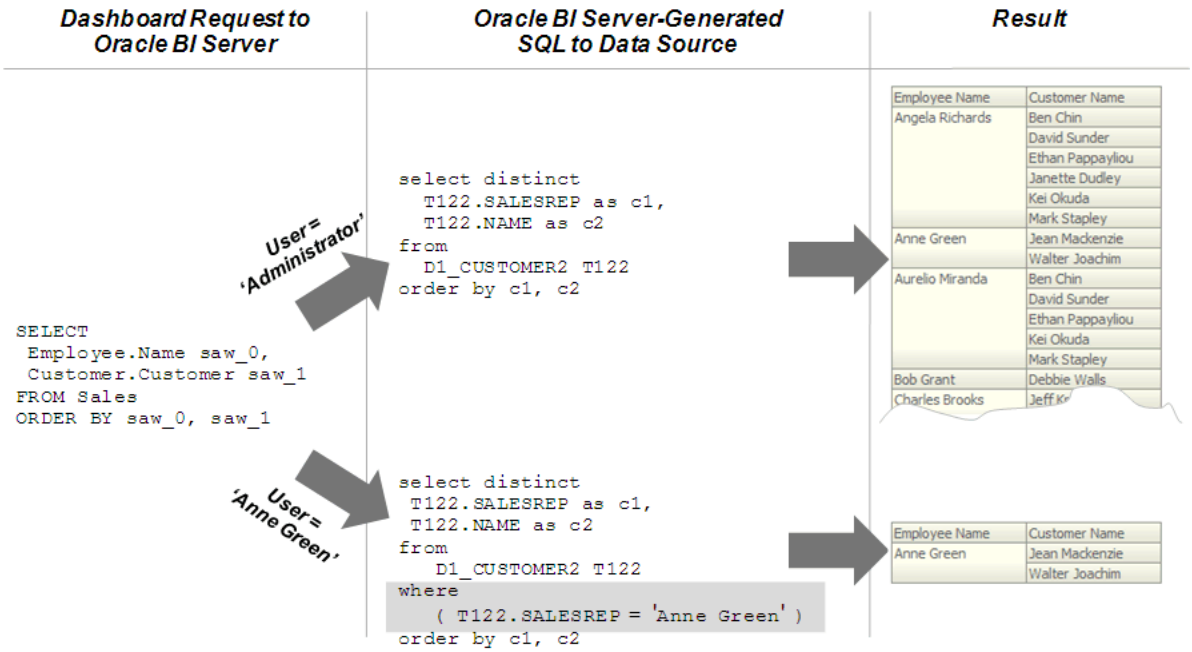
You typically do not set up data filters if you have implemented row-level security in the database, because in this case, your row-level security policies are being enforced by the database rather than the Oracle BI Server.

Data filters can be set for objects in both the Business Model and Mapping layer and the Presentation layer. Applying a filter on a logical object impacts all Presentation layer objects that use the object. If you set a filter on a Presentation layer object, it is applied in addition to any filters that might be set on the underlying logical objects.

Figure 14–1 illustrates how data filter rules are enforced in the Oracle BI Server. The security rules are applied to all incoming clients and cannot be breached, even when the Logical SQL query is modified.

In this example, a filter has been applied to an application role. When Anne Green, who is a member of that role, sends a request, the return results are limited based on the filter. Because no filters have been applied to the application roles for the Administrator user, all results are returned. The Oracle BI Server-generated SQL takes into account any data filters that have been defined.

Figure 14–1 Row-Level Security Enforcement in the Oracle BI Server



You should always set up data filters for particular application roles rather than for individual users.

To set up data filters to apply row-level authorization rules for queries:

1. Open your repository in the Administration Tool.
2. Select **Manage**, then select **Identity**.

3. In the Identity Manager dialog, in the tree pane, select **BI Repository**.
4. In the right pane, select the Application Roles tab, then double-click the application role for which you want to set data filters.

   Note that if you are in offline mode, no application roles appear in the list unless you have first modified them in online mode. See "About Applying Data Access Security in Offline Mode" for more information.
5. In the Application Role dialog, click **Permissions**.
6. In the User/Application Role Permissions dialog, click the Data Filters tab.

   To create filters, you first add objects on which you want to apply the filters. Then, you provide the filter expression information for the individual objects.
7. To add objects on which you want to apply filters, perform one of the following steps:
   - Click the **Add** button. Then, browse to locate the object you want, select it, and then click **Select**.
   - Click the **Name** field for an empty row. Then, browse to locate the object you want, select it, and then click **Select**.
8. To enter the filter expression for individual objects, perform one of the following steps:
   - Select the data filter, then click the **Expression Builder** button. Create the filter expression in Expression Builder, then click **OK**.
   - Click the **Data Filter** field for the appropriate filter, then type the filter expression.

   For example, you might want to define a filter like `"Sample Sales"."D2 Market"."M00 Mkt Key" > 5` to restrict results based on a range of values for another column in the table.

   You can also use repository and session variables in filter definitions. Use Expression Builder to include these variables to ensure the correct syntax.
9. Optionally, select a status for each filter from the Status list. You can choose one of the following options:
   - **Enabled:** The filter is applied to any query that accesses the object.
   - **Disabled:** The filter is not used and no other filters applied to the object at higher levels of precedence (for example, through an application role) are used.
   - **Ignored:** The filter is not in use, but any other filters applied to the object (for example, through a different application role) are used. If no other filters are enabled, no filtering occurs.
10. In addition to defining new filters, you can perform other operations in the Data Filters tab. Table 14–2 lists and describes the other buttons and options.

Table 14–2 Data Filters Tab: Buttons and Options

Option Name	Description
Subject Area	Select a subject area to only view data filters for that individual subject area, or select **All** to view all filters.
Total Filters	Lists the total number of data filters that have been defined for this particular user or application role.

Table 14–2 (Cont.) Data Filters Tab: Buttons and Options

Option Name	Description
Add 	Click **Add** to open the Browse dialog to add objects on which you want to apply data filters.
Delete 	Select a row and click **Delete** to remove a filter.
Browse 	Select a row and click **Browse** to change the object on which the filter is applied.
Edit Expression (Expression Builder) 	Select a row and click **Edit Expression** to add or change a filter expression for a particular object. You must first add an object before you can apply a filter expression to the row.
Find 	Enter text in the **Find** field and click **Find Down** or **Find Up** to find a particular string.

11. Click **OK**, then click **OK** again to return to the Identity Manager.

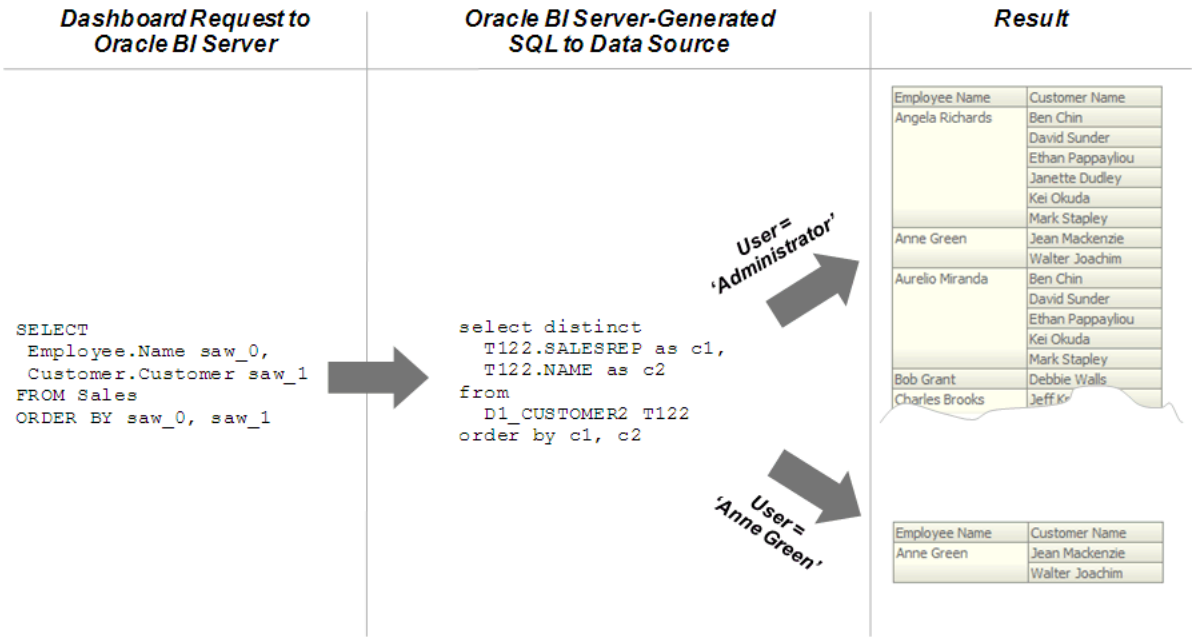
Setting Up Row-Level Security in the Database

To set up Oracle Business Intelligence for row-level security that has been implemented in the database, you can configure your connection pools so that the Oracle BI Server passes the credentials for each user to the database. The database then uses the credentials to apply its own row-level security rules to user queries.

Note that the row-level database security described in this section is different from database *authentication*, a topic discussed in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*. Rather, row-level database security provides database *authorization*. In other words, it applies access security to particular rows in the database.

Figure 14–2 illustrates how row-level security is enforced in the database for Oracle Business Intelligence queries. The security rules are applied to all incoming clients and cannot be breached, even when the Logical SQL query is modified. In this example, the results returned are different depending on which user generated the query, even though the SQL query generated by the Oracle BI Server is the same. The returned results are based on rules created and enforced in the database.

Figure 14–2 Row-Level Security Enforcement in the Database



In addition to setting up Oracle Business Intelligence for row-level security in the database, you must define your set of users, permissions, and security policies in the database itself. Refer to your database documentation for more information.

Note the following about this configuration:

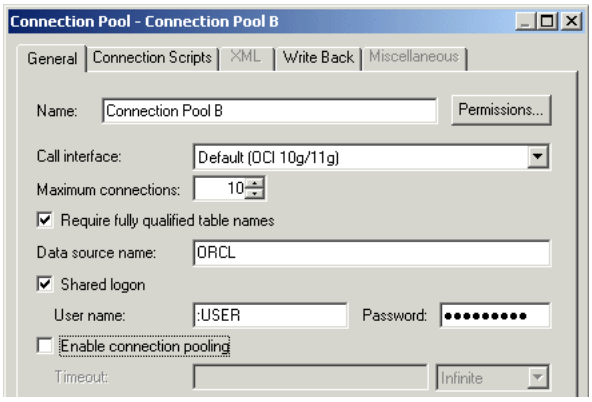
- This approach will not work when SSO is being used, or for any cases that involve impersonation (such as Delivers), because the password for the end user is not available to the Oracle BI Server.
- A connection script can be used to achieve the same functionality for Oracle Database data sources.
- For Essbase or Hyperion Financial Management data sources, the connection pool displays an additional option to implement SSO.

To set up Oracle Business Intelligence for row-level access security in the database:

1. Open your repository in the Administration Tool.
2. Double-click the connection pool associated with the database for which you want to set up database-level security.
3. In the General tab of the Connection Pool dialog, select **Shared logon**, and then enter `:USER` and `:PASSWORD` in the **User name** and **Password** fields. The `:USER` and `:PASSWORD` syntax automatically passes the value of user credentials upon login to the database. Note that the `:USER` and `:PASSWORD` syntax does not refer to session variables.

   Figure 14–3 shows the General tab of the Connection Pool dialog.

Figure 14–3 Entering Credentials for Database-Level Security in the Connection Pool



Note: Alternatively, you can use the database session context to pass end user identity to the database. Use a connection pool script to set up session context. Note that this approach does not rely on database authentication.

4. Click **OK** in the Connection Pool dialog.
5. Double-click the database object for which you want to set up database-level security.
6. In the Database dialog, select **Virtual Private Database**. Selecting this option ensures that the Oracle BI Server protects cache entries for each user.
7. Click **OK** in the Database dialog.

After you have set up row-level security in the database, you still need to set up object permissions in the repository for Presentation layer or other objects. You can also set query limits (governors). See "Setting Up Object Permissions" and "Setting Query Limits" for more information.

Setting Up Object Permissions

You can set up object permissions in your repository to control access to Presentation layer and Business Model and Mapping layer objects. You set object permissions using the Administration Tool.

There are two approaches to setting object permissions: you can set permissions for particular application roles in the Identity Manager, or you can set permissions for individual objects in the Presentation layer.

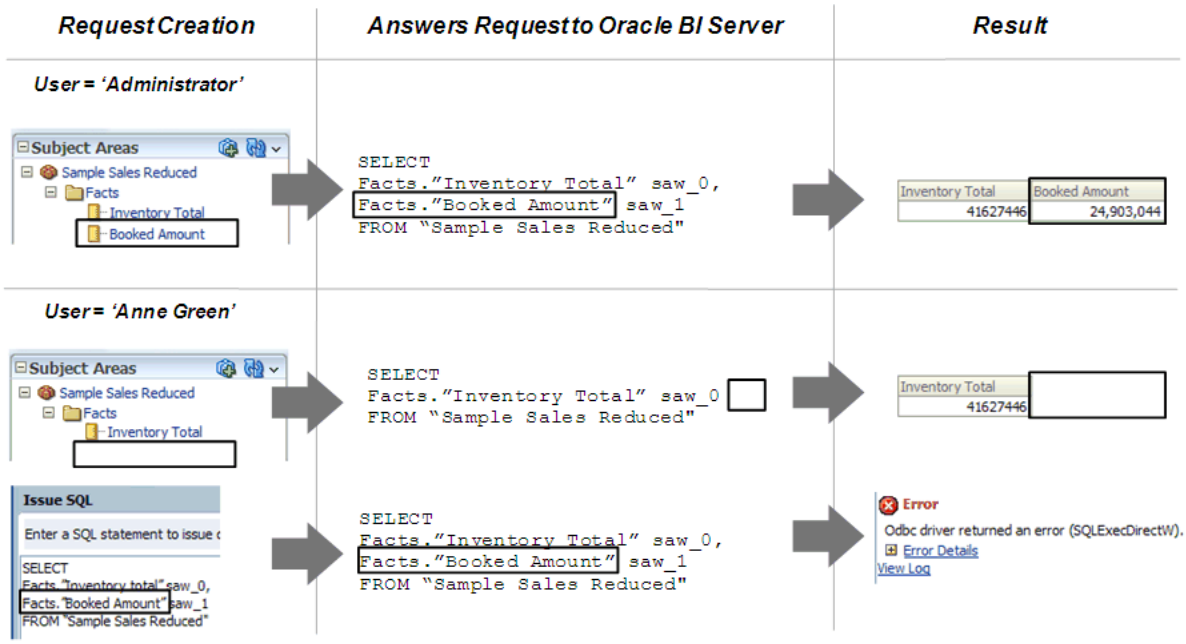
This section explains how to set up object permissions for application roles in the Identity Manager. See "Setting Permissions for Presentation Layer Objects" for information about setting object permissions for individual Presentation layer objects.

Setting up object permissions for particular application roles is useful when you want to define permissions for a large set of objects at one time. You should always set up object permissions for particular application roles rather than for individual users.

Figure 14–4 shows how object permissions restrict what users can see. The security rules are applied to all incoming clients and cannot be breached, even when the Logical SQL query is modified. In this example, an application role to which the Administrator belongs has been granted access to the Booked Amount column, so the

Administrator can view the returned results. The user Anne Green is not a member of an application role with access to this object and cannot see the column in the Subject Area pane in Answers. Even if the request SQL is modified, results are not returned for this column because of the application role-based object permissions that have been set.

Figure 14–4 Object Permission Enforcement in the Oracle BI Server



Note the following:

- If an application role is granted or disallowed permissions on an object from multiple sources (for example, explicitly and through one or more additional application roles), the permissions are applied based on the order of precedence.
- If you explicitly deny access to an object that has child objects, users who are members of the individual application role are denied access to the child objects. For example, if you explicitly deny access to a particular logical table, you are implicitly denying access to all of the logical columns associated with that table.
- Object permissions do not apply to repository and session variables, so values in these variables are not secure. Anybody who knows or can guess the name of the variable can use it in an expression in Answers or in a Logical SQL query. Because of this, do not put sensitive data like passwords in session or repository variables.
- You can control what level of privilege is granted by default to the AuthenticatedUser application role, which is the default application role associated with new repository objects. To do this, set the DEFAULT_PRIVILEGES parameter in the NQSConfig.INI file. See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.
- The AuthenticatedUser application role means "any authenticated user." This role is internal to the Oracle BI repository. It appears in the Permissions dialog for connection pools and Presentation layer objects, but it does not appear in the list of application roles in the Identity Manager.

To set up object permissions for individual application roles:

1. Open your repository in the Administration Tool.
2. Select **Manage**, then select **Identity**.
3. In the Identity Manager dialog, in the tree pane, select **BI Repository**.
4. In the right pane, select the Application Roles tab, then double-click the application role for which you want to set object permissions.

   Note that if you are in offline mode, no application roles appear in the list unless you have first modified them in online mode. See "About Applying Data Access Security in Offline Mode" for more information.
5. In the Application Role dialog, click **Permissions**.
6. In the User/Application Role Permissions dialog, in the Object Permissions tab, select an object by performing one of the following steps:
   - Click the **Add** button. Then, browse to locate the object you want, select it, and then click **Select**.
   - Click the **Name** field for an empty row. Then, browse to locate the object you want, select it, and then click **Select**.
7. Assign the appropriate permission for each object. You can choose one of the following options:
   - **Read:** Only allows read access to this object.
   - **Read/Write:** Provides both read and write access to this object.
   - **No Access:** Explicitly denies all access to this object.
8. Click **OK**, then click **OK** again to return to the Identity Manager.

About Permission Inheritance for Users and Application Roles

Users can have explicitly granted permissions. They can also have permissions granted through membership in application roles, that in turn can have permissions granted through membership in other application roles, and so on.

Permissions granted explicitly to a user have precedence over permissions granted through application roles, and permissions granted explicitly to the application role take precedence over any permissions granted through other application roles.

If there are multiple application roles acting on a user or application role at the same level with conflicting security attributes, then the user or application role is granted the least restrictive security attribute. Oracle currently requires that the application role with access to an object also have access to the object's container. For example, if ApplicationRole 1 has permission to access Column A, which is part of Table B, then ApplicationRole1 must also have permission to access Table B. Any explicit permissions acting on a user take precedence over any permissions on the same objects granted to that user through application roles.

In previous releases, the application role did not require access to an object's container, as described above. To revert to this behavior, go to the Oracle BI Server machine and create environment variable OBIS_SECURITY_10g_COMPATIBLE and set it to 1.

Filter definitions, however, are always inherited. For example, if User1 is a member of Role1 and Role2, and Role1 includes a filter definition but Role2 does not, the user inherits the filter definition defined in Role1.

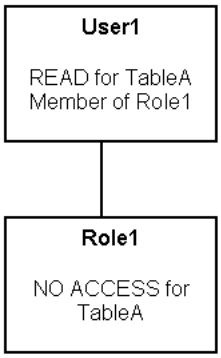
Note that you should always define object permissions for application roles rather than for individual users.

Example 14–1 Permission Inheritance 1

You might have a user (User1) who is explicitly granted permission to read a given table (TableA). Suppose also that User1 is a member of Role1, and Role1 explicitly denies access to TableA. The resultant permission for User1 is to read TableA, as shown in Figure 14–5.

Because permissions granted directly to the user take precedence over those granted through application roles, User1 has the permission to read TableA.

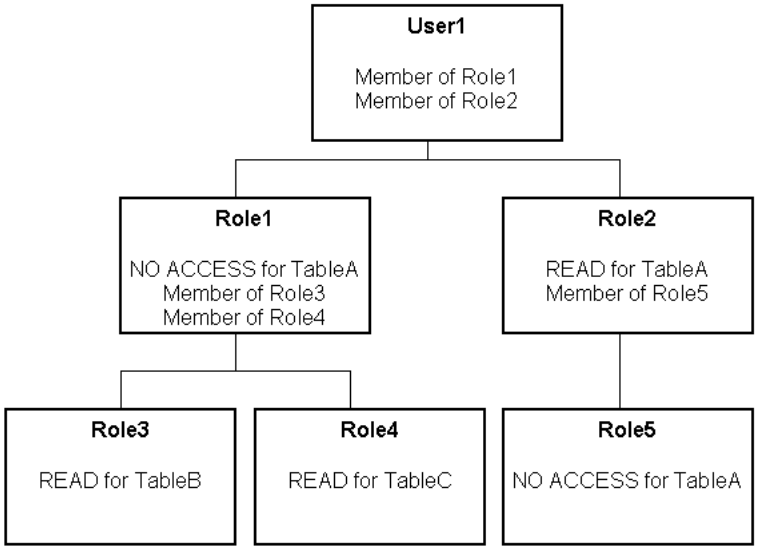
Figure 14–5 User Permissions and Application Role Permissions



Example 14–2 Permission Inheritance 2

Consider the situation shown in Figure 14–6.

Figure 14–6 Permissions Example



These are the resulting permissions:

- User1 is a direct member of Role1 and Role2, and is an indirect member of Role3, Role4, and Role5.
- Because Role5 is at a lower level of precedence than Role2, its denial of access to TableA is overridden by the `READ` permission granted through Role2. The result is that Role2 provides `READ` permission on TableA.

- The resultant permissions from Role1 are `NO ACCESS` for TableA, `READ` for TableB, and `READ` for TableC.
- Because Role1 and Role2 have the same level of precedence and because the permissions in each cancel the other out (Role1 denies access to TableA, Role2 allows access to TableA), the less restrictive level is inherited by User1. In other words, User1 has `READ` access to TableA.
- The total permissions granted to User1 are `READ` access for TableA, TableB, and TableC.

Overview of User and Application Role Commands

You can use the following commands to update application roles and users stored in the repository and the Oracle BI Presentation Catalog:

- Rename Application Role Command
- Delete Application Role Command
- Rename Users Command
- Delete Users Command

Note that the remaining upload and update commands (for example, Upload Repository Command and Update Repository Variables Command) update the repository, only.

The application roles and users update commands use two plugins: the RPD plugin, which updates the application roles and users in the repository, and the WEBCAT plugin, which updates application roles and the users in the Oracle BI Presentation Catalog.

These plugins function separately, and therefore the failure of one does not impact the other. In the event of a partial failure, or one of the two plugins failing, Oracle recommends that you address the root cause of the failure and then re-execute the command as you initially ran it. Note that re-applying the successful plugin has no impact on the results, but re-executing the command re-runs the failed plugin.

By default, the application roles and users update commands run the two plugins, and the order in which they are run is RPD and then WEBCAT. However, the commands include the `-L` option which allows you to specify an individual plugin or to reverse the default order in which the plugins are run. In most cases you will run the commands in the default order, but in some cases you may need to run only one plugin or reverse the order of the plugins.

Rename Application Role Command

Use the rename application role command `renameapproles` to upload a JSON file containing information about the application roles that you want to rename for a specific server instance.

You execute the utility through a launcher script, data-model-cmd.sh on UNIX and data-model-cmd.cmd on Windows. You can find the launcher script at the following location:

Oracle_Home`/user_projects/domains/bi/bitools/bin`

See "What You Need to Know Before Using the Command" for more information.

See "Overview of User and Application Role Commands" for information about the RPD and WEBCAT plugins used by this command.

Syntax

The `renameapproles` command takes the following parameters:

```
renameapproles -T <inputfile.json> [-L <plugin list>] -SI <service
instance> -U <cred username> [-P <cred password>] [-S <hostname>] [-N
<port number>] [-SSL] [-H]
```

Where

`T` specifies the name of the JSON input file containing the application role name changes for the server instance. See "Creating a JSON Rename Application Role Input File" for information about the correct syntax for the application role input file.

`SI` specifies the name of the service instance.

`L` specifies a single plugin to run or to reverse the default plugin execution order. The plugins determine where the system applies the updates: to the repository, the Oracle BI Presentation Catalog, or both. For RPD and WEBCAT plugin usage information, see "Overview of User and Application Role Commands."

Note the following options for `L`:

- `RPD`: Specify this option to rename application roles in the repository, only.
- `WEBCAT`: Specify this option to rename application roles in the Oracle BI Presentation Catalog, only.
- `WEBCAT,RPD`: Specify this option to reverse the default plugin run order. Note that the default plugin run order is repository (RPD) and then Oracle BI Presentation Catalog (WEBCAT).
- Omit this option to execute the plugins in their default order, which is repository (RPD) and then Oracle BI Presentation Catalog (WEBCAT).

`U` specifies a valid user's name to be used for Oracle BI EE authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you do not supply the password, then you will be prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you are using automated scripting to run the command.

`S` specifies the Oracle BI EE host name. Only include this option when you are running the command from a client installation.

`N` specifies the Oracle BI EE port number. Only include this option when you are running the command from a client installation.

`SSL` specifies to use SSL to connect to the WebLogic Server to run the command. Only include this option when you are running the command from a client installation.

`H` displays the usage information and exits the command.

Example

```
data-model-cmd.sh renameapproles -T approlenames.json -SI bi -U weblogic
-P password -S server1.example.com -N 7777 -SSL
```

Creating a JSON Rename Application Role Input File

Use the following syntax to create the JSON rename application role input file.

```
{
    "Title":"Target Application Roles",
    "App-Roles":[
```

```
            { "oldname":"<current_approle1>", "newname":"<new_approle1>" },
            { "oldname":"<current_approle2>", "newname":"<new_approle2>" },
            { "oldname":"<current_approle3>", "newname":"<new_approle3>" }
   ]
}
```

Delete Application Role Command

Use the delete application role command `deleteapproles` to upload a JSON file containing a list of application roles that you want to delete from a specific server instance.

You execute the utility through a launcher script, data-model-cmd.sh on UNIX and data-model-cmd.cmd on Windows. You can find the launcher script at the following location:

Oracle_Home`/user_projects/domains/bi/bitools/bin`

See "What You Need to Know Before Using the Command" for more information.

See "Overview of User and Application Role Commands" for information about the RPD and WEBCAT plugins used by this command.

Syntax

The `deleteapproles` command takes the following parameters:

```
deleteapproles -T <inputfile.json> [-L <plugin list>] -SI <service_
instance> -U <cred_username> [-P <cred_password>] [-S <hostname>] [-N
<port_number>] [-SSL] [-H]
```

Where

`T` specifies the name of the JSON input file containing the application roles to be deleted from the server instance. See "Creating a JSON Delete Application Role Input File" for information about the correct syntax for the application role input file.

`L` specifies a single plugin to run or to reverse the default plugin execution order. The plugins determine where the system applies the updates: to the repository, the Oracle BI Presentation Catalog, or both. For RPD and WEBCAT plugin usage information, see "Overview of User and Application Role Commands."

Note the following options for `L`:

- `RPD`: Specify this option to delete application roles in the repository, only.
- `WEBCAT`: Specify this option to delete application roles in the Oracle BI Presentation Catalog, only.
- `WEBCAT,RPD`: Specify this option to reverse the default plugin run order. Note that the default plugin run order is repository (RPD) and then Oracle BI Presentation Catalog (WEBCAT).
- Omit this option to execute the plugins in their default order, which is repository (RPD) and then Oracle BI Presentation Catalog (WEBCAT).

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for Oracle BI EE authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you do not supply the password, then you will be prompted for the password when the command is run. For security purposes, Oracle recommends that you include a

password in the command only if you are using automated scripting to run the command.

`S` specifies the Oracle BI EE host name. Only include this option when you are running the command from a client installation.

`N` specifies the Oracle BI EE port number. Only include this option when you are running the command from a client installation.

`SSL` specifies to use SSL to connect to the WebLogic Server to run the command. Only include this option when you are running the command from a client installation.

`H` displays the usage information and exits the command.

Example

```
data-model-cmd.sh deleteapproles -T approlenames.json -SI bi -U weblogic -P
password -S server1.example.com -N 7777 -SSL
```

Creating a JSON Delete Application Role Input File

Use the following syntax to create the JSON delete application role input file.

```
{
    "Title":"Target Application Roles",
    "App-Roles":[
         { "name":"<approle1>" },
         { "name":"<approle2>" },
         { "name":"<approle3>" }
   ]
}
```

Rename Users Command

Use the rename user command `renameusers` to upload a JSON file containing a list of information about the users that you want to rename for a specific server instance.

You execute the utility through a launcher script, data-model-cmd.sh on UNIX and data-model-cmd.cmd on Windows. You can find the launcher script at the following location:

Oracle_Home`/user_projects/domains/bi/bitools/bin`

See "What You Need to Know Before Using the Command" for more information.

See "Overview of User and Application Role Commands" for information about the RPD and WEBCAT plugins used by this command.

Syntax

The `renameusers` command takes the following parameters:

```
renameusers -T < usernames.json> [-L <plugin list>] -SI <service instance>
-U <cred username> [-P <cred password>] [-S <hostname>] [-N <port number>]
[-SSL] [-H]
```

Where

`T` specifies the name of the JSON input file containing the user name changes for the server instance. See "Creating a JSON Rename Users Input File" for information about the correct syntax for the application role input file.

`L` specifies a single plugin to run or to reverse the default plugin execution order. The plugins determine where the system applies the updates: to the repository, the Oracle

BI Presentation Catalog, or both. For RPD and WEBCAT plugin usage information, see "Overview of User and Application Role Commands."

Note the following options for `L`:

- `RPD`: Specify this option to rename users in the repository, only.
- `WEBCAT`: Specify this option to rename users in the Oracle BI Presentation Catalog, only.
- `WEBCAT,RPD`: Specify this option to reverse the default plugin run order. Note that the default plugin run order is repository (RPD) and then Oracle BI Presentation Catalog (WEBCAT).
- Omit this option to execute the plugins in their default order, which is repository (RPD) and then Oracle BI Presentation Catalog (WEBCAT).

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for Oracle BI EE authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you do not supply the password, then you will be prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you are using automated scripting to run the command.

`S` specifies the Oracle BI EE host name. Only include this option when you are running the command from a client installation.

`N` specifies the Oracle BI EE port number. Only include this option when you are running the command from a client installation.

`SSL` specifies to use SSL to connect to the WebLogic Server to run the command. Only include this option when you are running the command from a client installation.

`H` displays the usage information and exits the command.

Example

```
data-model-cmd.sh renameusers -T usernames.json -SI bi -U weblogic -P
password -S server1.example.com -N 7777 -SSL
```

Creating a JSON Rename Users Input File

Use the following syntax to create the JSON rename users input file.

```
{
    "Title":"Target Users",
    "Users":[
          { "oldname":"<current_user1>", "newname":"<new_user1>" },
          { "oldname":"<current_user2>", "newname":"<new_user2>" },
          { "oldname":"<current_user3>", "newname":"<new_user3>" }
    ]
}
```

Delete Users Command

Use the delete users command `deleteusers` to upload a JSON file containing a list of users that you want to delete from a specific server instance.

You execute the utility through a launcher script, data-model-cmd.sh on UNIX and data-model-cmd.cmd on Windows. You can find the launcher script at the following location:

Oracle_Home/user_projects/domains/bi/bitools/bin

See "What You Need to Know Before Using the Command" for more information.

See "Overview of User and Application Role Commands" for information about the RPD and WEBCAT plugins used by this command.

Syntax

The deleteusers command takes the following parameters:

```
deleteusers -T <usernames.json> [-L <plugin list>] -SI <service_instance>
-U <cred_username> [-P <cred_password>] [-S <hostname>] [-N <port_number>]
[-SSL] [-H]
```

Where

T specifies the name of the JSON input file containing the users to be deleted from the server instance. See "Creating a JSON Delete Users Input File" for information about the correct syntax for the application role input file.

L specifies a single plugin to run or to reverse the default plugin execution order. The plugins determine where the system applies the updates: to the repository, the Oracle BI Presentation Catalog, or both. For RPD and WEBCAT plugin usage information, see "Overview of User and Application Role Commands."

Note the following options for L:

- RPD: Specify this option to delete users in the repository, only.
- WEBCAT: Specify this option to delete users in the Oracle BI Presentation Catalog, only.
- WEBCAT,RPD: Specify this option to reverse the default plugin run order. Note that the default plugin run order is repository (RPD) and then Oracle BI Presentation Catalog (WEBCAT).
- Omit this option to execute the plugins in their default order, which is repository (RPD) then Oracle BI Presentation Catalog (WEBCAT).

SI specifies the name of the service instance.

U specifies a valid user's name to be used for Oracle BI EE authentication.

P specifies the password corresponding to the user's name that you specified for U. If you do not supply the password, then you will be prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you are using automated scripting to run the command.

S specifies the Oracle BI EE host name. Only include this option when you are running the command from a client installation.

N specifies the Oracle BI EE port number. Only include this option when you are running the command from a client installation.

SSL specifies to use SSL to connect to the WebLogic Server to run the command. Only include this option when you are running the command from a client installation.

H displays the usage information and exits the command.

Example

```
data-model-cmd.sh deleteusers -T usernames.json -SI bi -U weblogic -P
password -S server1.us.example.com -N 777 -SSL
```

Creating a JSON Delete Users Input File

Use the following syntax to create the JSON delete users input file.

```
{
    "Title":"Target Users",
    "Users":[
         { "name":"<user1>" },
         { "name":"<user2>" },
         { "name":"<user3>" }
    ]
}
```

Setting Query Limits

You can manage the query environment by setting query limits (governors) in the repository for particular application roles.

You can limit queries by the number of rows received, by maximum run time, and by restricting to particular time periods. You can also allow or disallow direct database requests or the Populate privilege.

You should always set query limits for particular application roles rather than for individual users.

This section contains the following topics:

Accessing the Query Limits Functionality in the Administration Tool

Follow the steps in this section to access the Query Limits tab of the User/Application Role Permissions dialog.

To access the query limits functionality in the Administration Tool for a particular application role:

1. Open your repository in the Administration Tool.
2. Select **Manage**, then select **Identity**.
3. In the Identity Manager dialog, in the tree pane, select **BI Repository**.
4. In the right pane, select the Application Roles tab, then double-click the application role for which you want to set query limits.

   Note that if you are in offline mode, no application roles appear in the list unless you have first modified them in online mode. See "About Applying Data Access Security in Offline Mode" for more information.
5. In the Application Role dialog, click **Permissions**.
6. In the User/Application Role Permissions dialog, click the Query Limits tab.

Limiting Queries By the Number of Rows Received

You can control runaway queries by limiting queries to a specific number of rows.

> **Note:** Any query limits you set should exceed the Presentation Server settings for 'Maximum Number of Rows Processed when Rendering a Table View' and 'Maximum Number of Rows to Download' by at least 500 to avoid error messages. If you choose to impose data source rows limits on certain users or Application Roles using the repository Max Rows query limits setting, then those users may receive 'Max Row Limit Exceeded' messages.
>
> For information about setting Presentation Server row limits, see "Using Fusion Middleware Control to Set Configuration Options for Data in Tables and Pivot Tables" and "Using Fusion Middleware Control to Set the Maximum Number of Rows Processed to Render a Table" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

To limit queries by the number of rows received:

1. Follow the steps in "Accessing the Query Limits Functionality in the Administration Tool" to access the Query Limits tab.
2. In the **Max Rows** column, type the maximum number of rows for users to retrieve from each source database object.
3. In the **Status Max Rows** field, select one of the following options for each database:
   - **Enable:** This limits the number of rows to the value specified. If the number of rows exceeds the **Max Rows** value, the query is terminated.
   - **Disable:** Disables any limits set in the **Max Rows** field.
   - **Warn:** Does not enforce limits, but logs queries that exceed the set limit in the Query log.
   - **Ignore:** Limits are inherited from the parent application role. If there is no row limit to inherit, no limit is enforced.
4. Click **OK**, then click **OK** again to return to the Identity Manager.

Limiting Queries By Maximum Run Time and Restricting to Particular Time Periods

You can forbid queries during certain time periods, or you can specify the maximum time a query can run on a database.

If you do not select a particular time period, access rights remain unchanged. If you allow or disallow access explicitly in one or more application roles, users are granted the least restrictive access for the defined time periods. For example, if a user is a member of an application role that is explicitly allowed access all day on Mondays, but that user also belongs to another application role that is disallowed access during all hours of every day, then the user has access on Mondays only.

To limit queries by maximum run time, or restrict queries to particular time periods:

1. Follow the steps in "Accessing the Query Limits Functionality in the Administration Tool" to access the Query Limits tab.

2. To specify the maximum time a query can run on a database, in the **Max Time (Minutes)** column, enter the maximum number of minutes you want queries to run on each database object. Then, in the **Status Max Time** field, select one of the following options for each database:
   - **Enable:** This limits the time to the value specified.
   - **Disable:** Disables any limits set in the **Max Time** field.
   - **Warn:** Does not enforce limits, but logs queries that exceed the set time limit in the Query log.
   - **Ignore:** Limits are inherited from the parent application role. If there is no time limit to inherit, no limit is enforced.
3. To restrict access to a database during particular time periods, in the **Restrict** column, click the **Ellipsis** button. Then, in the Restrictions dialog, perform the following steps:
   a. To select a time period, click the start time and drag to the end time.
   b. To explicitly grant access, click **Allow**.
   c. To explicitly deny access, click **Disallow**.
   d. Click **OK**.
4. Click **OK**, then click **OK** again to return to the Identity Manager.

Allowing or Disallowing Direct Database Requests

You can allow or disallow the ability to execute direct database requests for a particular application role.

For the selected role, this privilege overrides the property **Allow direct database requests by default** for the database object in the Physical layer.

To set the ability to execute direct database requests:

1. Follow the steps in "Accessing the Query Limits Functionality in the Administration Tool" to access the Query Limits tab.
2. For each database object, in the **Execute Direct Database Requests** field, select one of the following options:
   - **Allow:** Explicitly grants the ability to execute direct database requests for this database.
   - **Disallow:** Explicitly denies the ability to execute direct database requests for this database.
   - **Ignore:** Limits are inherited from the parent application role. If there is no limit to inherit, then direct database requests are allowed or disallowed based on the property **Allow direct database requests by default** for the database object.
3. Click **OK**, then click **OK** again to return to the Identity Manager.

Allowing or Disallowing the Populate Privilege

When a criteria block is cached, the Populate stored procedure writes the Cache/Saved Result Set value to the database. You can grant or deny this Populate privilege to particular application roles.

For the selected application role, this privilege overrides the property **Allow populate queries by default** for the database object in the Physical layer.

Any Oracle Marketing Segmentation user who writes a cache entry or saves a result set must be a member of an application role that has been assigned the `POPULATE` privilege for the target database. For more information about marketing cache, see the topic about setting up cache for target levels in the documentation for the Oracle Marketing Segmentation application.

To allow or disallow the Populate privilege:

1. Follow the steps in "Accessing the Query Limits Functionality in the Administration Tool" to access the Query Limits tab.
2. For each database object, in the **Populate Privilege** field, select one of the following options:
   - **Allow:** Explicitly grants the Populate privilege for this database. For all Marketing data warehouses, select **Allow**.
   - **Disallow:** Explicitly denies the Populate privilege for this database.
   - **Ignore:** Limits are inherited from the parent application role. If there is no limit to inherit, then the Populate privilege is allowed or disallowed based on the property **Allow populate queries by default** for the database object.
3. Click **OK**, then click **OK** again to return to the Identity Manager.

About Applying Data Access Security in Offline Mode

It is strongly recommended that you perform data access security tasks in the Administration Tool in online mode. If you must apply data access security in offline mode, be aware that users and application roles do not appear in the Administration Tool in offline mode unless you have first modified them in the Administration Tool in online mode.

For example, if you open the Administration Tool in offline mode without first making any changes in online mode, you will see zero users and application roles defined. However, if you first modify the users and application roles in online mode (for example, applying object permissions or setting query limits), they will subsequently be available in the Administration Tool in offline mode.

In online mode, you can retrieve the latest list of application roles from the policy store at any time by selecting **Action**, then selecting **Synchronize Application Roles** in the Identity Manager.

Setting Up Placeholder Application Roles for Offline Repository Development

Application roles are created and managed in the policy store using the Oracle WebLogic Administration Console and Fusion Middleware Control. These application roles are displayed in the Administration Tool in online mode so that you can use them to set data filters, object permissions, and query limits for particular roles. The application roles in the policy store are retrieved by the Oracle BI Server when it starts.

In some cases, you may want to proceed with setting up data access security in your repository for application roles that have not yet been defined in the policy store. You can do this by creating placeholder application roles in the Administration Tool, then proceeding with setting up data access security in the repository.

If you create placeholder application roles in the Administration Tool, you must eventually add them to the policy store. Run a consistency check in online mode to

identify application roles that have been defined in the Administration Tool, but that have not yet been added to the policy store. Be sure to use the same name in the policy store that you used for the placeholder role in the Administration Tool.

> **Note:** Use caution when defining and using placeholder roles. If you make changes to a role in offline mode that also exists in the policy store, the changes will be overwritten the next time you connect to the Oracle BI Server.

To create placeholder application roles in the Administration Tool:

1. Open your repository in the Administration Tool.
2. Select **Manage**, then select **Identity**.
3. In the Identity Manager dialog, select **Action > New > Application Role**.
4. In the Application Role dialog, provide the following information:
   - **Name:** Provide a name for the role.
   - **Display Name:** Enter the display name for the role.
   - **Description:** Optionally, provide a description of this application role.
   - **Members:** Use the **Add** and **Remove** buttons to add or remove users and other application roles as appropriate.
   - **Permissions:** Set object permissions, data filters, and query limits for this application role as appropriate. Refer to the other sections in this chapter for detailed information.
5. Click **OK** to return to the Identity Manager.

To check for application roles that need to be added to the policy store:

1. Open your repository in online mode in the Administration Tool.
2. Select **File**, then select **Check Global Consistency**.
3. Note any entries related to application roles, then add the appropriate roles to the policy store as appropriate. See *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition* for information about adding application roles to the policy store.
4. Optionally, select individual rows and click **Copy** to copy the entries to a text file.

Alternatively, you can check an individual application role by right-clicking the application role in the Identity Manager dialog and then selecting **Check Consistency**.

About the List of Users in the Administration Tool

The Identity Manager in the Administration Tool provides a list of users that have been defined for your system. The list of users is retrieved from your authentication provider. The set of users is refreshed when the Oracle BI Server is restarted.

To see the user list, select BI Repository in the Identity Manager navigation tree, and then select the Users tab in the right pane.

In online mode, by default, no users are retrieved, because the list of users might be very large. Select **Action**, then select **Set Online User Filter** to specify the set of users you want to retrieve.

The filter is empty by default, which means that no users are retrieved. Enter * to retrieve all users, or enter a combination of characters for a specific set of users, such as A* to retrieve all users whose names begin with the letter A. The filter is not case-sensitive.

In offline mode, users do not appear in the list unless you have first modified them in the Administration Tool in online mode. Because of this, you might not see any users in the Administration Tool in offline mode.

Double-click a user in the Users list to open the User dialog. You can do the following in this dialog:

- In the User tab, you can view the name, display name, and description for the user, as well as the application roles to which this user belongs. You can also set the query logging level for this user. See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about setting the query logging level.
- In the Logons tab, you can provide a list of data source-specific logons. In this tab, you can provide a mapping of credentials that you want to be passed to data sources for this user.

  This feature is used when you set up a data source connection with no shared connection pool, so that individual user names are passed directly to data sources. Rather than passing the Oracle Business Intelligence user credentials to the data source, you can map individual users to separate data source-specific credentials.

Important: Do not set object permissions, data filters, or query limits for individual users using the Permissions button. Always use application roles rather than individual users to secure data.

15

Completing Oracle BI Repository Setup

This chapter explains how to perform final Oracle BI repository setup tasks like configuring for Oracle Scorecard and Strategy management, saving the repository and checking consistency, testing the repository, and uploading the repository using the upload repository command.

This chapter contains the following topics:

Configuring the Repository for Oracle Scorecard and Strategy Management

If your organization licensed Oracle Scorecard and Strategy Management and if you have the appropriate privileges, then you can use this functionality as part of a default installation with no additional configuration. Some features, however, such as comments and status overrides, require repository configuration in order to work.

Note: See the following sections in *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition* for more information about these features:

- "About Comments"
- "About Status Overrides"

Configuring the Repository for Comments and Status Overrides

Oracle Scorecard and Strategy Management provides the capability to add comments (that is, annotations) or to override the status that is associated with specific dimension values for KPIs, Objectives, and Initiatives. KPI Watchlists offer the capability to add comments or to override statuses for KPIs. To enable these features, you must configure the repository to include a database object for storing the comment and status override information.

The database that you installed for use with Oracle Business Intelligence contains the Business Intelligence Platform schema, which includes required Oracle Scorecard and Strategy Management schema tables. For more information about installing a database for Oracle Business Intelligence and running the Repository Creation Assistant (RCU) to create the required schemas, see *Oracle Fusion Middleware Installation Guide for Oracle Business Intelligence*.

To configure the Oracle BI repository for comments and status overrides:

1. In the Administration Tool, open the repository in online mode.

   Online mode is strongly recommended for performing data access security tasks, such as the task described in Step 12 of this procedure.

2. In the Physical layer, right-click and select **New Database**. The Database dialog is displayed.
3. For **Name**, enter BSC.
4. For **Database**, select the type of database that you have installed for use with Oracle Business Intelligence (typically Oracle 11g).
5. Select the Connection Pool tab and click the **Add** button. The Connection Pool dialog is displayed.
6. For **Name**, enter BSC.
7. Select the **Call interface** appropriate for the database (for example, OCI 10g/11g for Oracle Database).
8. For **Data source name**, provide the information that is appropriate for the database that you have installed and configured for use with Oracle Business Intelligence. For example, for Oracle Database, enter a connection string similar to the following:

   ```
   (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=192.168.1.100)(PORT=1521))(CONNECT_
   DATA=(SERVER=DEDICATED)(SERVICE_NAME=KPIOracle)(SID=KPIOracl)))
   ```

   When connecting to an Oracle Database data source, you can include the entire connect string, or you can use the net service name defined in the tnsnames.ora file. If you choose to enter only the net service name, then you must set up a tnsnames.ora file in the following location within the Oracle Business Intelligence environment, so that the Oracle BI Server can locate the entry:

   ORACLE_HOME/network/admin

9. Select **Shared logon** and enter values for **User name** and **Password**. In this step, you provide the user/schema name and password that you created when you used the Repository Creation Utility (RCU) to populate the Business Intelligence Platform schema in the Oracle Business Intelligence database.

   Ensure that the user that you provide has read/write privileges for the ANNOTATIONS and ASSESSMENT_OVERRIDES tables in the Business Intelligence Platform schema.

10. Click **OK** in the Connection Pool dialog.
11. Click **OK** in the Database dialog.
12. Use the Identity Manager in the Administration Tool to allow the BISystem application role to execute direct database requests by default for the BSC database object. See "Allowing or Disallowing Direct Database Requests" for more information.

13. Save and close the repository.
14. Restart the Oracle BI Server.

Saving the Repository and Checking Consistency

In offline editing, remember to save your repository from time to time. You can save a repository in offline mode even though the business models may be inconsistent.

To determine if business models are consistent, use the Check Consistency command to check for compilation errors. You can check for errors in the whole repository by choosing **File > Check Global Consistency**, or in a particular logical business model by selecting a business model and then selecting **Check Consistency** from the right-click menu.

The consistency check analyzes the repository for certain kinds of errors and inconsistencies. For example, the consistency check finds any logical tables that do not have logical sources configured or any logical columns that are not mapped to physical sources, checks for undefined logical join conditions, determines whether any physical tables referenced in a business model are not joined to the other tables referenced in the business model, and checks for existence of a subject area for each business model.

Note: Passing a consistency check does not guarantee that a business model is constructed correctly, but it does rule out many common problems.

When you check for consistency, any errors or warnings that occur are displayed in a dialog. Correct any errors and check for consistency again, repeating this process until there are no more errors. An error message indicates a problem that must be corrected. A warning message identifies a possible problem. Refer to "Checking the Consistency of a Repository or a Business Model" for more information.

After upgrading from a previous software version and checking the consistency of your repository, you might observe messages that you had not received in previous consistency checks. This typically indicates inconsistencies that had been undetected before the upgrade, not new errors.

Using nqcmd to Test and Refine the Repository

When your repository is complete, you can run sample queries against it to test that it is created properly. Correct any problems you find and test again, repeating this process until you are satisfied with the results.

You can use the Oracle BI Server utility `nqcmd` to run test queries against the repository. The utility connects using an Oracle BI Server ODBC DSN. The Oracle BI Server must be running to use `nqcmd`.

The `nqcmd` utility is available on both Windows and UNIX systems.

This utility is intended for sanity testing. For heavier load testing, use Answers or another client. Queries with many thousands of rows will not work with `nqcmd`.

Although you can use `nqcmd` to run queries against other ODBC data sources, this section only describes how to use this utility to query the Oracle BI Server.

For example, on Windows:

1. Launch `nqcmd` from the following location:

   ORACLE_HOME/user_projects/domains/bi/bitools/bin

2. At `nqcmd`, type the desired options. For example:

```
nqcmd -dmy_dsn -umy_username [-pmy_password] -ssql_input_file -omy_result_file
```

You can pass a text file with SQL statements to the utility (script mode), or you can enter SQL at the command line (interactive mode). Queries are run against the default subject area, unless the object names used in the query are fully qualified.

Table 15–1 lists the command-line arguments for `nqcmd`.

Table 15–1 Command-Line Arguments for nqcmd

Argument	Description
-?	Lists the available command-line arguments.
-d*data_source_name*	The ODBC data source name for the Oracle BI Server to which you want to connect. If you omit this parameter, you are prompted at the command line to enter the DSN. **Tip:** On Windows, you can see the available local ODBC data source names by going to **Control Panel > Administrative Tools > Data Sources (ODBC)**. Click the System DSN tab to see a list of the available DSNs (for example, AnalyticsWeb_ coreapplication).
-u*user_name*	A valid Oracle Business Intelligence user name.
-p*password*	The corresponding Oracle Business Intelligence user password. The password argument is optional. If you do not provide a password argument, you are prompted to enter a password when you run the command. To minimize the risk of security breaches, Oracle recommends that you do not provide a password argument either on the command line or in scripts. Note that the password argument is supported for backward compatibility only, and will be removed in a future release. For scripting purposes, you can pass the password through standard input.
-s*sql_input_file_name*	The name and path of a text file that includes your test SQL queries.
-o*output_result_file_name*	The name and path of a file to which the utility will write the query results. This option is only used with -s.
-D*delimiter*	The delimiter used in the SQL input file (for example, semicolon (;) or colon (:)). This option is only used with -s.
-a	Enables asynchronous processing. This option is typically used with -s, when you are passing a SQL input file with multiple SQL statements.
-z	Enables UTF8 output instead of ACP in the output result file. You might need to include this option to display international characters in query results.
-utf16	Enables UTF16 instead of ACP for communication between `nqcmd` and the Oracle BI ODBC driver. You might need to include this option to display international characters in query results.

Table 15–1 (Cont.) Command-Line Arguments for nqcmd

Argument	Description
-NotForwardCursor	Disables the ODBC forward only cursor. Including this argument overrides the setting specified in the ODBC DSN.
-v	Displays the version of the `nqcmd` utility.
-SessionVar *session_variable_name=session_variable_value*	Includes the specified session variable and sets it to the specified value.

Although -C, -R, -f, -H, -q, and -NoFetch are listed by the utility as available arguments, these options are not typically used.

If you run `nqcmd` in interactive mode rather than script mode (or in other words, if you do not pass a SQL input file), `nqcmd` shows a menu of options after you provide the data source name and user credentials. Although many options are shown, you typically only use Q, T, and C against the Oracle BI Server.

Enter Q to type a query at the command line. You must enter the query on a single line, and you cannot use a semicolon as a delimiter. Pressing Enter sends the SQL to the Oracle BI Server.

Enter T to browse presentation tables, or C to browse presentation columns. The utility prompts you for catalog pattern, user pattern, table pattern, and table type pattern before returning results.

For catalog pattern, enter the subject area that contains the tables you want to see. For table pattern, enter the specific table. You can enter percent (%) to see all subject areas or all tables, use % with other characters to replace a set of characters, or use underscore (_) with other characters to replace a single character.

User pattern and table type pattern are not used in queries against the Oracle BI Server, so enter % for these options.

You can also enter D to view a static list of data types supported by the Oracle BI Server.

Upload Repository Command

Use the upload repository command `uploadrpd` to upload the repository to Oracle BI Server. Uploading the repository to Oracle BI Server allows BI Server to load the repository into memory upon startup and makes the repository available for queries.

> **Note:** You can only upload the repository to a specific service instance.

Oracle provides the `downloadrpd` and `uploadrpd` commands for offline repository diagnostic and development purposes such as testing, only. Note that in all other repository development and maintenance situations, you should use BAR to utilize BAR's repository upgrade and patching capabilities and benefits.

You can use this command to upload the Oracle BI Repository in RPD format. You cannot use this command to upload a repository composed of MDS XML documents.

You execute the utility through a launcher script, data-model-cmd.sh on UNIX and data-model-cmd.cmd on Windows. You can find the launcher script at the following location:

Oracle_Home/user_projects/domains/bi/bitools/bin

See "What You Need to Know Before Using the Command" for more information.

Syntax

The uploadrpd command takes the following parameters:

```
uploadrpd -I <RPDname> [-W <RPDpwd>] -SI <service_instance> -U <cred_
username> [-P <cred_password>] [-S <hostname>] [-N <port_number>] [-SSL]
[-H]
```

Where

I specifies the name of the repository that you want to upload.

W is the repository's password. If you do not supply the password, then you will be prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you are using automated scripting to run the command.

SI specifies the name of the service instance.

U specifies a valid user's name to be used for Oracle BI EE authentication.

P specifies the password corresponding to the user's name that you specified for U. If you do not supply the password, then you will be prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you are using automated scripting to run the command.

S specifies the Oracle BI EE host name. Only include this option when you are running the command from a client installation.

N specifies the Oracle BI EE port number. Only include this option when you are running the command from a client installation.

SSL specifies to use SSL to connect to the WebLogic Server to run the command. Only include this option when you are running the command from a client installation.

H displays the usage information and exits the command.

Example

```
data-model-cmd.sh uploadrpd -I repository.rpd -SI bi -U weblogic -S
server1.example.com -N 7777 -SSL
```

Making the Repository Available for Queries

After you build a repository and it is consistent, you need to upload the repository using the "Upload Repository Command" so that all Oracle BI Server instances can access it. Uploading the repository allows the Oracle BI Server to load the repository into memory upon startup and makes the repository available for queries.

When the repository is uploaded and you can connect to it, run sample queries against it to test that it is created properly. Correct any problems you find and test again, repeating this process until you are satisfied with the results.

Note: You must upload an Oracle BI repository in RPD format. You cannot upload a repository composed of MDS XML documents.

Creating Data Source Connections to the Oracle BI Server for Client Applications

If you want to enable end user client applications to connect to the new repository, you must define an ODBC data source connection to the Oracle BI Server for each application.

Note that Oracle BI Presentation Services has the same relationship to the Oracle BI Server as any other client application.

See "Integrating Other Clients with Oracle Business Intelligence" in *Oracle Fusion Middleware Integrator's Guide for Oracle Business Intelligence Enterprise Edition* for information about creating ODBC data source connections for the Oracle BI Server.

Publishing to the User Community

After testing is complete, notify the user community that the data sources are available for querying. Presentation Services users only need to know the URL to type in their browser. Client/server users (for example, users accessing the Oracle BI Server with a query tool or report writer client application) need to know the subject area names, the computer on which the server is running, and their user IDs and passwords. They also need to have the ODBC DSN for the Oracle BI Server installed on their computers, and they may need to know the logical names of repositories if multiple repositories are used and the data source name (DSN) being created does not point to the default repository.

16

Setting Up Data Sources on Linux and UNIX

This chapter describes how to set up data sources for use with Oracle Business Intelligence when the Oracle BI Server is running on Linux or UNIX.

Most repository development is performed on Windows, because the Administration Tool runs only on Windows. When you move to a production system, however, you can choose to run the Oracle BI Server on a Linux or UNIX platform.

See "System Requirements and Certification" for information about supported Linux and UNIX platforms.

This chapter contains the following topics:

About Setting Up Data Sources on Linux and UNIX

When the Oracle BI Server is running on Linux or UNIX, most data source connections are for query-only access. The Administration Tool is used for importing objects and is a Windows-only tool. Because of this, data source connections for import must be set up on Windows.

Note that some data source connections on Linux and UNIX do support write operations for special functions, like data source connections for write-back, usage tracking, and annotations for Oracle Scorecard and Strategy Management.

When the Oracle BI Server is running on Linux or UNIX and you need to update database object settings (such as the database type) or connection pool settings, you can copy the repository file to a Windows computer, make the changes using the Administration Tool on Windows, and then copy the repository file back to the Linux or UNIX computer.

There are three types of data source connections on Linux and UNIX platforms:

- Native data source gateway connections, such as OCI for Oracle Database or DB2 CLI for IBM DB2

- ODBC connections using the DataDirect Connect ODBC drivers that are bundled with Oracle Business Intelligence
- Native ODBC connections using external drivers, such as for Teradata data sources

Note that you can have a single repository that contains both DataDirect Connect ODBC connections and native ODBC connections. In this situation, both the native ODBC drivers and DataDirect ODBC drivers need to be managed by the same DataDirect ODBC driver manager. For example, the Teradata ODBC drivers include their required ODBC driver managers. When the Teradata ODBC driver is used with Oracle BI EE, the driver must be managed by the DataDirect ODBC driver manager bundled with Oracle BI EE.

Configuring Data Source Connections Using Native Gateways

You can connect to both Oracle Database and DB2 using native gateways (OCI and DB2 CLI, respectively).

For Oracle Database, note the following:

- The Oracle BI Server uses the Oracle Call Interface (OCI) to connect to the database. OCI is installed by default with Oracle BI Enterprise Edition. You must use the bundled version to connect.
- In the tnsnames.ora file, the Oracle Database alias (the defined entry name) must match the Data Source Name used in the repository connection pools of all physical Oracle databases.

  When connecting to an Oracle Database data source, you can include the entire connect string, or you can use the net service name defined in the tnsnames.ora file. If you choose to enter only the net service name, you must set up a tnsnames.ora file in the following location within the Oracle Business Intelligence environment, so that the Oracle BI Server can locate the entry:

  BI_DOMAIN/config/fmwconfig/bienv/core
- Edit the obis.properties file to set environment variables for the database client.

For DB2, note the following:

- Install the appropriate database client on the computer running the Oracle BI Server, then edit the obis.properties file to set environment variables for the database client.
- For Windows, you can set environment variables for DB2 in the obis.properties file. For example, if configuring DB2 CLI, then you must modify obis.properties to include the DB2 executable path.
- You need to create a catalog associated with each database so that the client connects to the database by catalog name. To create a catalog associated with each database, enter and run the following command:

```
db2 catalog tcpip node <DB2 database> remote <hostname> server <port number>;
db2 catalog database <DB2 database> as <DB2 database> at node <DB2 database>;
connect to <DB2 database> user db2admin using welcome1
```

To edit the obis.properties file to set environment variables for Oracle Database or DB2:

1. Open the obis.properties file. You can find this file at:

   BI_DOMAIN/config/fmwconfig/bienv/obis

2. Include the appropriate environment variable settings for the database client of your choice. Ensure that you point to the appropriate libraries, depending on whether you are using a 32-bit or 64-bit database.

```
DB2INSTANCE=db2user
IBM_DB_LIB=/scratch/db2user/sqllib/lib
IBM_DB_DIR=/scratch/db2user/sqllib
LD_LIBRARY_PATH=/scratch/db2user/sqllib/lib64:/scratch/db2user/sqllib/lib32
PATH=$PATH:/scratch/db2user/sqllib/bin:/scratch/db2user/sqllib/adm:/scratch/db2
user/sqllib/misc
DB2_HOME=/scratch/db2user/sqllib
IBM_DB_INCLUDE=/scratch/db2user/sqllib/include
DB2LIB=/scratch/db2user/sqllib/lib
```

   See Example 16–1 for sample values.

3. Save and close the file.

4. Restart OBIS1.

Example 16–1 Sample obis.properties Entries for Oracle Database and DB2 (32-Bit)

This example shows sample entries in obis.properties for Oracle Database and DB2 on various platforms.

```
################################################################
# Linux: Oracle BI 32 bit mode
################################################################
#set +u

# Oracle Parameters
#---------------------------
# Make sure that Oracle DB 32 bit Client is installed
#ORACLE_HOME=/export/home/oracle/10g
#export ORACLE_HOME
#TNS_ADMIN=$ORACLE_HOME/network/admin
#export TNS_ADMIN
#PATH=$ORACLE_HOME/bin:/opt/bin:$PATH
#export PATH
#LD_LIBRARY_PATH=$ORACLE_HOME/lib:$LD_LIBRARY_PATH
#export LD_LIBRARY_PATH

# If you have Linux 64 bit Platform, and would like to run Oracle BI 32 bit
# then you must install Oracle DB 64 bit client, and this client comes with
# 32 bit libraries under $ORACLE_HOME/lib32. The LD_LIBRARY_PATH in this case
# shall be like this:
#LD_LIBRARY_PATH=$ORACLE_HOME/lib32:$LD_LIBRARY_PATH
#export LD_LIBRARY_PATH

# DB2 Parameters
#---------------------------
#make sure the /DB2ISTANCE/sqllib/lib points to 32 lib file
#. /DB2ISTANCE/sqllib/db2profile
#---------------------------

################################################################
# Solaris: Oracle BI 64 bit mode
################################################################
#set +u

# Oracle Parameters
#---------------------------
```

```
# Make sure to install Oracle DB 64 bit Client
#ORACLE_HOME=/export/home/oracle/10g
#export ORACLE_HOME
#TNS_ADMIN=$ORACLE_HOME/network/admin
#export TNS_ADMIN
#PATH=$ORACLE_HOME/bin:/opt/bin:$PATH
#export PATH
#LD_LIBRARY_PATH_64=$ORACLE_HOME/lib:$LD_LIBRARY_PATH_64:/opt/j2se/jre/lib/sparc
#export LD_LIBRARY_PATH_64
#---------------------------

# DB2 Parameters
#---------------------------
#make sure the /DB2ISTANCE/sqllib/lib points to 64 lib file
#. /DB2ISTANCE/sqllib/db2profile
#LD_LIBRARY_PATH_64=/DB2ISTANCE/sqllib/lib:$LD_LIBRARY_PATH_64
#export LD_LIBRARY_PATH_64
#---------------------------

################################################################
# HPUX Itanium: Oracle BI 64 bit mode
################################################################
#set +u

# Oracle Parameters
#---------------------------
#ORACLE_HOME=/export/home/oracle/10g
#export ORACLE_HOME
#TNS_ADMIN=$ORACLE_HOME/network/admin
#export TNS_ADMIN
#PATH=$ORACLE_HOME/bin:/opt/bin:$PATH
#export PATH
#SHLIB_PATH=$ORACLE_HOME/lib:$SHLIB_PATH:/opt/j2se/jre/lib/hp700
#export SHLIB_PATH
#---------------------------

# DB2 Parameters
#---------------------------
#make sure the /DB2ISTANCE/sqllib/lib points to 64 lib file
#. /DB2ISTANCE/sqllib/db2profile
#SHLIB_PATH=/DB2ISTANCE/sqllib/lib:$SHLIB_PATH
#export SHLIB_PATH
#---------------------------

################################################################
# AIX: Oracle BI 64 bit mode
################################################################
#set +u

# Oracle Parameters
#---------------------------
#ORACLE_HOME=/export/home/oracle/10g
#export ORACLE_HOME
#TNS_ADMIN=$ORACLE_HOME/network/admin
#export TNS_ADMIN
#PATH=$ORACLE_HOME/bin:/opt/bin:$PATH
#export PATH
#LIBPATH=$ORACLE_HOME/lib:$LIBPATH:/opt/j2se/jre/lib/sparc
#export LIBPATH
#---------------------------
```

```
# DB2 Parameters
#---------------------------
#make sure the /DB2ISTANCE/sqllib/lib points to 64 lib file
#. /DB2ISTANCE/sqllib/db2profile
#---------------------------
```

Note that the shell script excerpts shown are examples only and are not recommendations for particular software platforms. See "System Requirements and Certification" for information about supported software platforms.

Troubleshooting OCI Connections

There are several reasons why you might have trouble connecting to an Oracle Database using OCI.

Check to ensure that the following conditions are true:

- The computer running the Oracle BI Server must use Oracle Call Interface (OCI) to connect to the database.
- If you choose not to use the entire connect string in the repository connection pool, you must ensure that a valid tnsnames.ora file is set up in the following location within the Oracle Business Intelligence environment, so that the Oracle BI Server can locate the entry:

  BI_DOMAIN/config/fmwconfig/bienv/core
- If you choose not to use the entire connect string in the repository connection pool, ensure that the net service name in the tnsnames.ora file matches the Data Source Name used in the connection pool.

  For example, in the following example of a tnsnames.ora entry, the corresponding Oracle BI repository connection pool Data Source Name is ITQA2.

```
ITQA2 =
   (DESCRIPTION =
      (ADDRESS_LIST =
         (ADDRESS = (PROTOCOL = TCP)(HOST = ITQALAB2)(PORT = 1521))
   (CONNECT_DATA =
      (SERVICE_NAME = ITQALAB2.corp)
      )
   )
```

The following procedure shows how to check repository database and connection pool settings against the Oracle tnsnames.ora settings.

To check that the repository database and connection pool settings are correct:

1. Open the repository in the Administration Tool.
2. In the Physical layer, double-click the database you want to check to display the Database dialog.
3. On the General tab, in the **Data source definition: Database** field, ensure that the appropriate Oracle Database version is selected. Then, click **OK**.
4. Open the Connection Pool dialog for this data source. You might need to expand the database object in the Physical layer to see the connection pool object.
5. In the Connection Pool dialog, check that the following is true:
   - The **Call interface** field displays the appropriate value for the release of the Oracle Database you are using.

- The **Data source name** field displays the Oracle Database net service name that you defined in the tnsnames.ora entry.
- The **User name** and **password** fields contain the correct values.

Change the values if necessary, then click **OK**.

6. In the Oracle Business Intelligence environment, open the tnsnames.ora file located in the following directory:

   BI_DOMAIN/config/fmwconfig/bienv/core

7. Check that a valid net service name exists with the following characteristics:
   - Matches the connection pool settings for the Data Source Name
   - Specifies the targeted Oracle physical database

About Updating Row Counts in Native Databases

This topic applies if both of the following are true:

- You are using the Update Rowcount functionality in the Administration Tool in offline mode.
- You are running a heterogeneous environment, such as the Oracle BI Server on UNIX, while remote administrators run the Administration Tool on Windows computers.

When using the Update Rowcount functionality in offline mode, the Administration Tool uses local data source connection definitions on the client computer, *not* the server data sources. Therefore, Oracle Database or DB2 clients must be configured on the Windows computer running the Administration Tool so that the following conditions are true:

- Data sources point to the same database identified in the Oracle Business Intelligence obis.properties file on the UNIX server.
- The name of the local data source must also match the name of the data source defined in the Connection Pool object in the physical layer of the Oracle BI repository (.rpd) file.

If these conditions are not true, and if the server and client data sources are pointing at different databases, then erroneous updated row counts or incorrect results appear.

Using DataDirect Connect ODBC Drivers on Linux and UNIX

Oracle Business Intelligence provides DataDirect Connect ODBC drivers and driver managers for Linux and UNIX operating systems for connectivity to Microsoft SQL Server, Sybase ASE, Informix, Hive, and Impala databases.

After Oracle Business Intelligence is installed, the DataDirect Connect ODBC drivers are installed in *ORACLE_HOME*/bi/common/ODBC/Merant/7.1.4/drivers.

You do not need to set the ODBCINI environment variable to set up the DataDirect Connect ODBC drivers. This variable is set automatically during installation.

Refer to "System Requirements and Certification" for information about supported operating systems, databases, and driver versions for the DataDirect Connect ODBC drivers.

This section contains the following topics:

- Configuring the DataDirect Connect ODBC Driver for Microsoft SQL Server Database
- Configuring the DataDirect Connect ODBC Driver for Sybase ASE Database
- Configuring the DataDirect Connect ODBC Driver for Informix Database
- Configuring the DataDirect Connect ODBC Driver for Cloudera Impala Database
- Configuring the DataDirect Connect ODBC Driver for Apache Hive Database

Configuring Oracle Business Intelligence to Use DataDirect

When you install Oracle BI EE, the required DataDirect 7.1.4 drivers are installed and automatically configured. You can define the default settings in obis.properties and odbc.ini files.

You will need to modify your existing database configurations to use the DataDirect drivers. For information about modifying your existing database configuration, see the following procedures:

- Configuring the DataDirect Connect ODBC Driver for Microsoft SQL Server Database
- Configuring the DataDirect Connect ODBC Driver for MySQL Database
- Configuring the DataDirect Connect ODBC Driver for Sybase ASE Database
- Configuring the DataDirect Connect ODBC Driver for Informix Database
- Configuring the DataDirect Connect ODBC Driver for Cloudera Impala Database
- Configuring the DataDirect Connect ODBC Driver for Apache Hive Database

Additional DataDirect Configuration for Oracle Essbase

This configuration requires you to modify the essbase.cfg file and the epmsys_registry.bat utility (epmsys_registry.sh for Linux and UNIX) so that Essbase connects to the DataDirect 7.1.4 drivers.

To modify the essbase.cfg file:

1. Open essbase.cfg for editing. You can find essbase.cfg at:

   BI_DOMAIN/config/fmwconfig/biconfig/essbase

2. Locate the BPM_ORACLE_DriverDescriptor entry and change the value to "DataDirect 7.1.4 Oracle Wire Protocol".

3. Use Fusion Middleware Control to restart Essbase.

To modify the Essbase Server registry:

1. Open a command prompt and enter and run the command to view the ESSBASE_SERVER registry properties.

   For Windows:

   ```
   C:\OracleBI\instances\instance1\config\foundation\11.1.2.0>epmsys_registry.bat view ESSBASE_SERVER
   ```

   For Linux/UNIX:

   ```
   C:/OracleBI/instances/instance1/config/foundation/11.1.2.0>epmsys_registry.sh view ESSBASE_SERVER
   ```

2. Locate and copy the ID property.

3. Enter and run the following command to update the DataDirect version listed in the `BPM_Oracle_DriverDescriptor` property.

   For Windows:

   ```
   epmsys_registry.bat updateproperty #your_componentID/@BPM_Oracle_
   DriverDescriptor "DataDirect 7.1 Oracle Wire Protocol"
   ```

   For Linux/UNIX:

   ```
   epmsys_registry.sh updateproperty #your_componentID/@BPM_Oracle_
   DriverDescriptor "DataDirect 7.1 Oracle Wire Protocol"
   ```

4. If you scaled your Oracle BI EE environment, then repeat the above steps for all instances.
5. Use Fusion Middleware Control to restart Essbase.

Configuring the DataDirect Connect ODBC Driver for Microsoft SQL Server Database

The name of the DataDirect ODBC driver file to connect to a Microsoft SQL Server database is ARsql27.so. See "System Requirements and Certification" for supported versions of Microsoft SQL Server.

To configure the DataDirect Connect ODBC Driver to connect to Microsoft SQL Server:

1. Open the obis.properties file. You can find this file at:

   ```
   BI_DOMAIN/config/fmwconfig/bienv/obis
   ```

2. Locate the `LD_LIBRARY_PATH` variable. Note the following:
   - For Solaris, Linux, and HP-UX, the library path variable is `LD_LIBRARY_PATH`.
   - For AIX, the library path variable is `LIBPATH`.

   For example, to set the library path variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$ORACLE_HOME/bifoundation/server/bin,
   $ORACLE_HOME/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bifoundation/odbc/lib,
   $ORACLE_INSTANCE,
   $ORACLE_HOME/lib
   ```

3. If necessary, update the `LD_LIBRARY_PATH` variable to include the DataDirect driver path. For example, to update the variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$ORACLE_HOME/common/ODBC/Merant/7.1.4/drivers,
   $ORACLE_HOME/bifoundation/server/bin,
   $ORACLE_HOME/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bifoundation/odbc/lib,
   $ORACLE_INSTANCE,
   $ORACLE_HOME/lib
   ```

4. In obis.properties, locate the `PATH` variable and if necessary, include the DataDirect driver path.
5. Save and close the file.
6. Open the odbcinst.ini file. You can find this file at:

   ```
   BI_DOMAIN/config/fmwconfig/bienv/core
   ```

7. Create an entry for the database, ensuring that the ODBC connection name is identical to the data source name specified in the connection pool defined in the repository. Ensure that you set the Driver parameter to the file name and location of the DataDirect Connect driver for Microsoft SQL Server. In the following example, the Driver parameter is set to the DataDirect Connect driver, and the data source name is SQLSERVER_DB.

```
[SQLSERVER_DB]
Driver=/ORACLE_HOME/bi/common/ODBC/Merant/7.1.4/drivers/ARsqls27.so
Description=DataDirect 7.1 SQL Server Wire Protocol
Address=111.111.111.111,1433
AlternateServers=
AnsiNPW=Yes
ConnectionRetryCount=0
ConnectionRetryDelay=3
Database=dbschema_name
LoadBalancing=0
LogonID=
Password=
QuoteID=No
ReportCodePageConversionErrors=0
```

8. Save and close the odbc.ini file.
9. Restart OBIS1.
10. Open the repository in the Administration Tool on a Windows computer.
11. In the Physical layer, double-click the database object for the Microsoft SQL Server database.
12. Click **OK**.
13. Save and close the repository.
14. On the Linux or UNIX computer, shut down Oracle Business Intelligence.
15. Copy the repository from the Windows computer to the Linux or UNIX computer.
16. Start Oracle Business Intelligence on the Linux or UNIX computer.

Configuring the DataDirect Connect ODBC Driver for MySQL Database

The name of the DataDirect ODBC driver file to connect to a MySQL database is ARmysql27.so. See "System Requirements and Certification" for information about supported versions of MySQL.

To configure the DataDirect Connect ODBC Driver to connect to MySQL Database:

1. Open the obis.properties file. You can find this file at:

   BI_DOMAIN/config/fmwconfig/bienv/obis

2. Locate the LD_LIBRARY_PATH variable. Note the following:

   - For Solaris, Linux, and HP-UX, the library path variable is LD_LIBRARY_PATH.
   - For AIX, the library path variable is LIBPATH.

   For example, to set the library path variable for the driver on Linux:

```
LD_LIBRARY_PATH=$ORACLE_HOME/bifoundation/server/bin,
$ORACLE_HOME/bifoundation/web/bin,
$ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
```

```
$ORACLE_HOME/bifoundation/odbc/lib,
$ORACLE_INSTANCE,
$ORACLE_HOME/lib
```

3. If necessary, update the `LD_LIBRARY_PATH` variable to include the DataDirect driver path. For example, to update the variable for the driver on Linux:

```
LD_LIBRARY_PATH=$ORACLE_HOME/bi/common/ODBC/Merant/7.1.4/drivers,
$ORACLE_HOME/bifoundation/server/bin,
$ORACLE_HOME/bifoundation/web/bin,
$ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
$ORACLE_HOME/bifoundation/odbc/lib,
$ORACLE_INSTANCE$:$ORACLE_HOME/lib
```

4. In obis.properties, locate the `PATH` variable and if necessary, include the DataDirect driver path.
5. Save and close the file.
6. Open the odbc.ini file. You can find this file at:

```
BI_DOMAIN/config/fmwconfig/bienv/core
```

7. Create an entry for the database, ensuring that the ODBC connection name is identical to the data source name specified in the connection pool defined in the repository. Ensure that you set the Driver parameter to the file name and location of the DataDirect Connect driver for MySQL Database. For NetworkAddress, provide the IP address or fully qualified host name and the port number.

   In the following example, the Driver parameter is set to the DataDirect Connect driver, and the data source name is MySQL_DB.

```
[MYSQL_DB]
Driver=/ORACLE_HOME/bi/common/ODBC/Merant/7.1.4/drivers/ARmysql27.so
Description=DataDirect 7.1.4 MySQL Wire Protocol
ApplicationUsingThreads=1
ConnectionRetryCount=0
ConnectionRetryDelay=3
Database=default
DefaultLongDataBuffLen=1024
EnableDescribeParam=0
HostName=localhost
InteractiveClient=0
LoadBalancing=0
LogonID=my_id
Password=my_password
PortNumber=1526
ReportCodepageConversionErrors=0
TreatBinaryAsChar=0
```

8. Save and close the odbc.ini file.
9. Restart OBIS1.

Configuring the DataDirect Connect ODBC Driver for Sybase ASE Database

The name of the DataDirect ODBC driver file to connect to a Sybase ASE database is ARase27.so. See "System Requirements and Certification" for information about supported versions of Sybase ASE.

To configure the DataDirect Connect ODBC Driver to connect to Sybase ASE Database:

1. Open the obis.properties file. You can find this file at:

   BI_DOMAIN/config/fmwconfig/bienv/obis

2. Locate the LD_LIBRARY_PATH variable. Note the following:

   - For Solaris, Linux, and HP-UX, the library path variable is LD_LIBRARY_PATH.
   - For AIX, the library path variable is LIBPATH.

   For example, to set the library path variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$ORACLE_HOME/bifoundation/server/bin,
   $ORACLE_HOME/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bifoundation/odbc/lib,
   $ORACLE_INSTANCE$:$ORACLE_HOME/lib
   ```

3. If necessary, update the LD_LIBRARY_PATH variable to include the DataDirect driver path. For example, to update the variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$ORACLE_HOME/common/ODBC/Merant/7.1.4/drivers,
   $ORACLE_HOME/bifoundation/server/bin,
   $ORACLE_HOME/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bifoundation/odbc/lib,$ORACLE_INSTANCE$:$ORACLE_HOME/lib
   ```

4. In obis.properties, locate the PATH variable and if necessary, include the DataDirect driver path.
5. Save and close the file.
6. Open the odbc.ini file. You can find this file at:

   BI_DOMAIN/config/fmwconfig/bienv/core

7. Create an entry for the database, ensuring that the ODBC connection name is identical to the data source name specified in the connection pool defined in the repository. Ensure that you set the Driver parameter to the file name and location of the DataDirect Connect driver for Sybase ASE Database. For NetworkAddress, provide the IP address or fully qualified host name and the port number.

   In the following example, the Driver parameter is set to the DataDirect Connect driver, and the data source name is SybaseASE_DB.

   ```
   [SybaseASE_DB]
   Driver=/ORACLE_HOME/bi/common/ODBC/Merant/7.1.4/drivers/ARase27.so
   Description=DataDirect 7.1 Sybase Wire Protocol
   AlternateServers=
   ApplicationName=
   ApplicationUsingThreads=1
   ArraySize=50
   AuthenticationMethod=0
   Charset=
   ConnectionRetryCount=0
   ConnectionRetryDelay=3
   CursorCacheSize=1
   Database=Paint
   DefaultLongDataBuffLen=1024
   EnableDescribeParam=0
   EnableQuotedIdentifiers=0
   EncryptionMethod=0
   GSSClient=native
   HostNameInCertificate=
   ```

```
InitializationString=
Language=
LoadBalancing=0
LogonID=my_id
NetworkAddress=111.111.111.111,5005
OptimizePrepare=1
PacketSize=0
Password=
RaiseErrorPositionBehavior=0
ReportCodePageConversionErrors=0
SelectMethod=0
ServicePrincipalName=
TruncateTimeTypeFractions=0
TrustStore=
TrustStorePassword=
ValidateServerCertificate=1
WorkStationID=
```

8. Save and close the odbc.ini file.
9. Restart OBIS1.

Configuring the DataDirect Connect ODBC Driver for Informix Database

The name of the DataDirect ODBC driver file to connect to an Informix database is ARifcl27.so. See "System Requirements and Certification" for information about supported versions of Informix.

To configure the DataDirect Connect ODBC Driver to connect to Informix:

1. Open the obis.properties file. You can find this file at:

   BI_DOMAIN/config/fmwconfig/bienv/obis

2. Locate the LD_LIBRARY_PATH variable. Note the following:
   - For Solaris, Linux, and HP-UX, the library path variable is LD_LIBRARY_PATH.
   - For AIX, the library path variable is LIBPATH.

   For example, to set the library path variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$ORACLE_HOME/bifoundation/server/bin,
   $ORACLE_HOME/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bifoundation/odbc/lib,
   $ORACLE_INSTANCE$:$ORACLE_HOME/lib
   ```

3. If necessary, update the LD_LIBRARY_PATH variable to include the DataDirect driver path. For example, to update the variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$ORACLE_HOME/common/ODBC/Merant/7.1.4/lib,
   $ORACLE_HOME/bifoundation/server/bin,
   $ORACLE_HOME/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bifoundation/odbc/lib,
   $ORACLE_INSTANCE$:$ORACLE_HOME/lib
   ```

4. In obis.properties, locate the PATH variable and if necessary, include the DataDirect driver path.
5. Save and close the file.

6. Open the odbc.ini file. You can find this file at:

   BI_DOMAIN/config/fmwconfig/bienv/core

7. Create an entry for the database, ensuring that the ODBC connection name is identical to the data source name specified in the connection pool defined in the repository. Ensure that you set the Driver parameter to the file name and location of the DataDirect Connect driver for Informix. Also, you must specify the HostName parameter (you can use the fully qualified host name or the IP address) and the PortNumber parameter.

   In the following example, the Driver parameter is set to the DataDirect Connect driver, and the data source name is Informix_DB.

```
[Informix_DB]
Driver=/ORACLE_HOME/bi/common/ODBC/Merant/7.1.4/drivers/ARifcl27.so
Description=DataDirect Informix Wire Protocol
AlternateServers=
ApplicationUsingThreads=1
CancelDetectInterval=0
ConnectionRetryCount=0
ConnectionRetryDelay=3
Database=
HostName=111.111.111.111
LoadBalancing=0
LogonID=informix
Password=mypassword
PortNumber=1526
ReportCodePageConversionErrors=0
ServerName=
TrimBlankFromIndexName=1
```

8. Save and close the odbc.ini file.
9. Restart OBIS1.

Configuring the DataDirect Connect ODBC Driver for Cloudera Impala Database

The name of the DataDirect ODBC driver file to connect to a Cloudera Impala database is ARimpala27.so. See "System Requirements and Certification" for information about supported versions of Cloudera Impala.

To configure the DataDirect Connect ODBC Driver to connect to Cloudera Impala:

1. Open the obis.properties file. You can find this file at:

   BI_DOMAIN/config/fmwconfig/bienv/obis

2. Locate the `LD_LIBRARY_PATH` variable. Note the following:
   - For Solaris, Linux, and HP-UX, the library path variable is `LD_LIBRARY_PATH`.
   - For AIX, the library path variable is `LIBPATH`.

   For example, to set the library path variable for the driver on Linux:

```
LD_LIBRARY_PATH=$ORACLE_HOME/bifoundation/server/bin,
$ORACLE_HOME/bifoundation/web/bin,
$ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
$ORACLE_HOME/bifoundation/odbc/lib,
$ORACLE_INSTANCE,
$ORACLE_HOME/lib
```

3. If necessary, update the `LD_LIBRARY_PATH` variable to include the DataDirect driver path. For example, to update the variable for the driver on Linux:

```
LD_LIBRARY_PATH=$ORACLE_HOME/common/ODBC/Merant/7.1.4/drvers,
$ORACLE_HOME/bifoundation/server/bin,
$ORACLE_HOME/bifoundation/web/bin,
$ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
$ORACLE_HOME/bifoundation/odbc/lib,
$ORACLE_INSTANCE,
$ORACLE_HOME/lib
```

4. In obis.properties, locate the `PATH` variable and if necessary, include the DataDirect driver path.
5. Save and close the file.
6. Open the odbc.ini file. You can find this file at:

```
BI_DOMAIN/config/fmwconfig/bienv/core
```

7. Create an entry for the database, ensuring that the ODBC connection name is identical to the data source name specified in the connection pool defined in the repository. Ensure that you set the Driver parameter to the file path of the DataDirect Connect driver for Cloudera Impala. Also, you must specify the HostName parameter (you can use the fully qualified host name or the IP address) and the PortNumber parameter.

   In the following example, the Driver parameter is set to the DataDirect Connect driver, and the data source name is Impala_DB.

```
[Impala_DB]
Driver=/ORACLE_HOME/bi/common/ODBC/Merant/7.1.4/lib/ARimpala27.so
Description=Oracle 7.1 Cloudera Impala Wire Protocol
ArraySize=16384
Database=default
DefaultLongDataBuffLen=1024
DefaultOrderByLimit=-1
EnableDescribeParam=0
HostName=localhost
LoginTimeout=30
MaxVarcharSize=2000
PortNumber=21050
RemoveColumnQualifiers=0
StringDescribeType=12
TransactionMode=0
UseCurrentSchema=0
WireProtocolVersion=2
```

8. Save and close the odbc.ini file.

   Note that if you are using Impala 1.3.*x*, then you must complete the steps in the following "Configure Impala 1.3.x to Include a LIMIT Clause" section. If you are using Impala 1.4 (CDH 5.1) or later, then you do not need to complete the additional steps.

9. Restart OBIS1.
10. Go to the Administration Tool to import the Cloudera Impala metadata. See "Importing Cloudera Impala Metadata Using the Windows ODBC Driver" for more information.

Configure Impala 1.3.*x* to Include a LIMIT Clause

Impala 1.3.*x* requires that queries with an ORDER BY clause contain a LIMIT clause. Note that there are three methods to specify this clause in the configuration, but Oracle recommends "Method 1: Modify the Impala daemon's default query options."

Specifying a default order by limit using any of the following methods returns a maximum of 2,000,000 rows for queries with an ORDER by clause.

Method 1: Modify the Impala daemon's default query options:

Oracle recommends this method of specifying the LIMIT clause.

If you specify the LIMIT clause using this method and your queries include an ORDER BY clause, then Impala returns a maximum of 2,000,000 rows. If this limit is exceeded, then Impala throws an exception.

1. Go to the Cloudera Manager's home page and click the **Impala** service. The Impala service page appears.
2. In the Impala service page, click **Configuration**, and the select **View and Edit**. The Configuration page appears.
3. In the Configuration page, select **Impala Daemon Default Group**. Locate Impala Daemon Query Options Advanced (also called default_query_options) and add the following entries:

   ```
   default_order_by_limit=2000000
   abort_on_default_limit_exceeded=true
   ```

4. Click **Save Changes**.
5. In the Cloudera Manager's home page, restart the Impala service.

For queries over 2,000,000 rows, specify a higher `default_order_by_limit` value.

Method 2: Modify the Impala daemon's default query options when Impala is not managed by Cloudera Manager:

If your Impala environment is not managed by Cloudera Manager, then use the Impala product documentation to help you modify the LIMIT clause. To complete this task, use the "Configuring Impala Startup Options Through the Command Line" topic, located in the *CDH 5 Installation Guide*. This topic is available at the following location:

http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH5/5.0/Impala/Installing-and-Using-Impala/ciiu_config_options.html?scroll=config_options

Using the "Configuring Impala Startup Options Through the Command Line" topic, add the following entry in `IMPALA_SERVER_ARGS`:

```
-default_query_options 'default_order_by_limit=2000000;abort_on_default_limit_
exceeded=true'
```

If you specify the LIMIT clause using this method and your queries include an ORDER BY clause, then Impala returns a maximum of 2,000,000 rows. If this limit is exceeded, then Impala throws an exception.

Method 3: Modify the DefaultOrderByLimit parameter in the odbc.ini Impala DSN entry:

Note: Note that the Default Order By Limit can also be specified by the client instead of the Impala server.

Use this method if you do not have rights to modify the Impala daemon using the previous methods. If you use this method, then Impala *silently truncates* your value to 2,000,000 rows.

1. Open the odbc.ini file. You can find this file at:

   BI_DOMAIN/config/fmwconfig/bienv/core

2. Locate the Impala_DB database entry, and then locate the `DefaultOrderByLimit` parameter.
3. Update the value to 2000000. For example:

   ```
   DefaultOrderByLimit=2000000
   ```

4. Save and close the odbc.ini file.

If you need your query to return more than 2,000,000 rows, then specify a higher `DefaultOrderByLimit` value.

Configuring the DataDirect Connect ODBC Driver for Apache Hive Database

The name of the DataDirect ODBC driver file to connect to a Apache Hive is libARhive27.so. See "System Requirements and Certification" for information about supported versions of Apache Hive. See "Limitations on the Use of Apache Hive with Oracle Business Intelligence" for information about using Apache Hive data sources.

To configure the DataDirect Connect ODBC Driver to connect to Apache Hive:

1. Open the obis.properties file. You can find this file at:

   BI_DOMAIN/config/fmwconfig/bienv/obis

2. Locate the `LD_LIBRARY_PATH` variable. Note the following:
   - For Solaris, Linux, and HP-UX, the library path variable is `LD_LIBRARY_PATH`.
   - For AIX, the library path variable is `LIBPATH`.

   For example, to set the library path variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$ORACLE_HOME/bifoundation/server/bin,
   $ORACLE_HOME/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bifoundation/odbc/lib,
   $ORACLE_INSTANCE,
   $ORACLE_HOME/lib"
   ```

3. If necessary, update the `LD_LIBRARY_PATH` variable to include the DataDirect driver path. For example, to update the variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$ORACLE_HOME/common/ODBC/Merant/7.1.4/drivers,
   $ORACLE_HOME/bifoundation/server/bin,
   $ORACLE_HOME/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bifoundation/odbc/lib,
   $ORACLE_INSTANCE,
   $ORACLE_HOME/lib
   ```

4. In obis.properties, locate the `PATH` variable and if necessary, include the DataDirect driver path.
5. In obis.properties file, either above or below the LD_LIBRARY_PATH variable, create the HADOOP_DLL variable to point to the DataDirect driver.

   For example:

   ```
   HADOOP_DLL=ORACLE_HOME/bi/common/ODBC/Merant/7.1.4/drivers/ARhive27.so
   ```

6. Save and close the file.
7. Open the odbc.ini file. You can find this file at:

   ```
   BI_DOMAIN/config/fmwconfig/bienv/core
   ```

8. Create an entry for the database, ensuring that the ODBC connection name is identical to the data source name specified in the connection pool defined in the repository. Ensure that you set the Driver parameter to the file name and location of the DataDirect Connect driver for Hive. Also, you must specify the HostName parameter (you can use the fully qualified host name or the IP address) and the PortNumber parameter.

   In the following example, the Driver parameter is set to the DataDirect Connect driver, and the data source name is Hive.

   ```
   [Hive]
   Driver=/ORACLE_HOME/bi/common/ODBC/Merant/7.1.4/drivers/ARhive27.so
   Description=Oracle 7.1 Apache Hive Wire Protocol
   ArraySize=16384
   Database=default
   DefaultLongDataBuffLen=1024
   EnableDescribeParam=0
   HostName=localhost
   LoginTimeout=30
   MaxVarcharSize=2000
   PortNumber=10000
   RemoveColumnQualifiers=0
   StringDescribeType=12
   TransactionMode=0
   UseCurrentSchema=0
   ```

9. Save and close the odbc.ini file.
10. Restart OBIS1.
11. Create a soft link `libARicu27.so` pointing to `libARicu26.so`.

    For example:

    ```
    cd //ORACLE_HOME/common/ODBC/Merant/7.1.4/drivers/
    ln -s libARicu26.so libARicu27.so
    ```

Configuring Database Connections Using Native ODBC Drivers

Oracle Business Intelligence bundles UNIX ODBC drivers for some data sources, but not all. For these data sources, including Teradata and Oracle TimesTen In-Memory Database, you must install your own ODBC driver, then update the obis.properties and odbc.ini files to configure the data source.

If you are using Teradata, see also "Avoiding Spool Space Errors for Queries Against Teradata Data Sources" for related information.

To configure a database connection using a native ODBC driver:

1. Open the obis.properties file. You can find this file at:

   BI_DOMAIN/config/fmwconfig/bienv/obis

2. Locate the LD_LIBRARY_PATH variable. Note the following:

   - For Solaris, Linux, and HP-UX, the library path variable is LD_LIBRARY_PATH.
   - For AIX, the library path variable is LIBPATH.

   For example, to set the library path variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$ORACLE_HOME/opt/teradata/client/14.10/odbc_64/lib,
   $ORACLE_HOME/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bifoundation/odbc/lib,
   $ORACLE_INSTANCE,
   $ORACLE_HOME/lib
   ```

3. If necessary, update the LD_LIBRARY_PATH variable to include the driver path. For example, to update the variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$ORACLE_HOME/opt/teradata/client/14.10/odbc_64/lib,
   $ORACLE_HOME/bifoundation/server/bin,
   $ORACLE_HOME/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bifoundation/odbc/lib,
   $ORACLE_INSTANCE,
   $ORACLE_HOME/lib
   ```

4. In obis.properties, locate the PATH variable and if necessary, include the DataDirect driver path.
5. Save and close the file.
6. Open the odbc.ini file. You can find this file at:

   BI_DOMAIN/config/fmwconfig/bienv/core

7. Create an entry for the database, ensuring that the ODBC connection name is identical to the data source name specified in the connection pool defined in the repository. Ensure that you set the Driver parameter to the file name and location of the native ODBC driver for the database, with the library suffix that is appropriate for the operating system (for example, .so for Solaris and AIX, or .sl for HP-UX).

   The following example provides details for a Teradata data source on Linux, with a data source name of Tera_Northwind.

   ```
   [Tera_Northwind]
   Driver=/opt/teradata/client/14.10/odbc_64/lib/tdata.so
   Description=NCR 3600 running Teradata V2R6.2
   DBCName=10.345.67.899
   astUser=
   Username=northwind
   Password=northwind
   Database=northwind
   DefaultDatabase=northwind
   NoScan=no
   ```

Note that the DefaultDatabase parameter can be left empty only if you have selected the option **Require fully qualified table names** in the General tab of the Connection Pool dialog for this data source in the Administration Tool.

8. Still in the odbc.ini file, add an entry to the section [ODBC Data Sources] with the details appropriate for the data source. The following example provides details for a Teradata data source with a data source name of Tera_Northwind.

   ```
   Tera_Northwind=tdata.so
   ```

9. Restart OBIS1.

10. Using the Administration Tool, open the repository and add the new DSN you created as the Connection Pool **Data source name** for the appropriate physical databases. See "Creating or Changing Connection Pools" for more information.

Setting Up Oracle TimesTen In-Memory Database on Linux and UNIX

To set up Oracle TimesTen In-Memory Database data sources, first follow the instructions in "Configuring TimesTen Data Sources" to set up the TimesTen data source. Ensure that you go to the section "Configuring Database Connections Using Native ODBC Drivers" to obtain the correct steps for Linux and UNIX systems.

Next, review the best practices described in "Improving Use of System Memory Resources with TimesTen Data Sources" and implement them as needed.

Finally, if the user that starts OBIS1 does not have the path to the TimesTen DLL ($TIMESTEN_HOME/lib) in their operating system `LD_LIBRARY_PATH` variable (or `SHLIB_PATH` and `LIBPATH` on HP-UX and AIX, respectively), you must add the TimesTen DLL path as a variable in the obis.properties file.

To update obis.properties to include TimesTen variables on Linux and UNIX:

1. Open obis.properties for editing. You can find obis.properties at:

   BI_DOMAIN`/config/fmwconfig/bienv/obis`

2. Add the required TimesTen variable `TIMESTEN_DLL`, and also update the `LD_LIBRARY_PATH` variable (or equivalent), as shown below:

   ```
   TIMESTEN_DLL=$TIMESTEN_HOME/lib/libttclient.so
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$TIMESTEN_HOME/lib
   ```

3. Save and close the file.

4. Repeat these steps on each computer that runs the Oracle BI Server process. If you are running multiple Oracle BI Server instances on the same computer, then ensure that you update the ias-component tag appropriately for each instance in obis.properties (for example, ias-component id="coreapplication_obis1", ias-component id="coreapplication_obis2", and so on).

5. Restart OBIS1.

Configuring Oracle RPAS ODBC Data Sources on AIX UNIX

You can access Oracle RPAS ODBC data sources when the Oracle BI Server is running on an AIX UNIX platform. To configure this database connection, first update the odbc.ini file to configure the Oracle RPAS ODBC data source, then use the rdaadmin tool to define dimension tables as not normalized at run time.

See "Setting Up Oracle RPAS Data Sources" for information about configuring Oracle RPAS ODBC data sources on Windows.

To configure Oracle RPAS ODBC as a data source on AIX UNIX:

1. Log on as a separate telnet session.
2. Open the odbc.ini file. You can find this file at:

   BI_DOMAIN/config/fmwconfig/bienv/core

3. In the RPAS data source section, edit the values. For example:

   ```
   [RPAS Sample]
   Data Source Name=RPAS Sample
   Driver=[client RPASClient/lib/raix/oaodbc.so
   DriverUnicodeType=1
   Description=OpenRDA DSN
   ```

   The `Data Source Name` you provide must match the value entered for `DATABASE:` in Step 3 of the following procedure. Also, you must add the line `DriverUnicodeType=1` as shown in the preceding example.

To use the rdaadmin client tool to define dimension tables as not normalized at run time:

1. Locate the rdaadmin client tool in the following location:

   `/bin/rdaadmin`

2. Run the rdaadmin client tool by typing the following command:

   `rdaadmin`

3. Enter appropriate text when prompted, as follows:

   `DATABASE: [`*Oracle_RPAS_database_name*`]`

   The database name must match the name given for the Data Source Name in the previous task (for example, RPAS Sample).

   `ADDRESS: [`*ip_address*`]`

   `PORT: [`*port_number*`]`

   An example port number value is 1707.

   `CONNECT_STRING: [NORMALIZE_DIM_TABLES=NO]`

   This value treats dimension tables as not normalized at run time.

   `TYPE: []`

   `SCHEMA_PATH: []`

   `REMARKS: []`

4. The RPAS environment variable `OPENRDA` should be declared in the Oracle BI Server session on UNIX. For example, declare the variable as follows using the 64 bit rdaadmin client tool:

   ```
   OPENRDA_INI=/rpasclient64/config/raix/openrda.ini export OPENRDA_INI
   ```

Configuring Essbase Data Sources on Linux and UNIX

The Oracle BI Server uses the Essbase client libraries to connect to Essbase data sources. The Essbase client libraries are installed by default with Oracle BI Enterprise

Edition. No additional configuration is required to enable Essbase data source access for full installations of Oracle BI Enterprise Edition.

However, for HP-UX Itanium systems, the following additional steps are required:

1. Define ESSLANG and LANG.

   For example:

   ```
   ESSLANG=English_UnitedStates.UTF-8@Binary
   export ESSLANG
   LANG=en_US.utf8
   export LANG
   ```

2. Comment out LOCALE, SORT_ORDER_LOCALE, and SORT_TYPE in the NQSConfig.ini file. For example:

   ```
   [ GENERAL ]
   // Localization/Internationalization parameters.
   // LOCALE="English-usa";
   // SORT_ORDER_LOCALE="English-usa";
   // SORT_TYPE="binary";
   ```

Configuring DB2 Connect on IBM z/OS and s/390 Platforms

IBM DB2 Connect does not support the option of automatically disconnecting when an application using it receives an interrupt request.

When the native database uses DB2 Connect workstation, then you must change the setting of the parameter INTERRUPT_ENABLED. This parameter must be changed on any Oracle Business Intelligence computer if the database or any data source resides on IBM DB2 on a mainframe running z/OS or s/390 platforms.

> **Note:** If IBM DB2 is used, DB2 Connect must be installed on the Oracle BI Server computer. The version of DB2 Connect must match the most recent DB2 instance that was configured as a data source.

To configure the INTERRUPT_ENABLED parameter:

1. Configure a database alias to be used as the native CLI Data Source Name. For example, create a new database entry using DB2 Configuration Assistant.
2. Using the database alias created and the name of the actual target DB2 database, set the INTERRUPT_ENABLED parameter using the following syntax:

   ```
   uncatalog dcs db local_dcsname
   catalog dcs db local_dcsname as target_dbname parms \",,INTERRUPT_ENABLED\"
   ```

   where:

   – local_dcsname represents the local name of the host or database (database alias name)

   – target_dbname represents the name of database on the host or database system

   > **Note:** Ensure that you use backslashes to pass the quotation marks as part of the string.

The following example uses an OS390 DB2 instance:

```
uncatalog dcs db DB2_390
catalog dcs db DB2_390 as Q10B parms \",,INTERRUPT_ENABLED,,,,,\"
catalog database DB2_390 as DB2_390 at node NDE1EF20 authentication dcs
```

17

Managing Oracle BI Repository Files

This chapter describes tasks related to managing your Oracle BI repository files, including comparing and merging repositories, equalizing objects, querying and managing metadata, and changing the repository password.

This chapter contains the following topics:

Comparing Repositories

This section explains how to compare all repository objects in two different repositories.

If you are using an Oracle BI Applications repository and have customized its content, you can use this feature to compare your customized repository to a new version of the repository received with Oracle BI Applications.

See "Merging Repositories" for more information about merging your customized Oracle BI Applications repository with a new version of the repository.

This section contains the following topics:

Comparing Repositories Using the Compare Dialog

This section explains how to use the Compare dialog in the Administration Tool.

To compare two repositories:

1. In the Administration Tool, open a repository in offline mode.

   The repository that you open in this step is referred to as the current repository. See "Using Online and Offline Repository Modes" for instructions on opening a repository.

2. From the **File** menu, select **Compare**.

3. In the Select Original Repository dialog, select the repository you want to compare to the open repository. Select **Repository** from the submenu to select a binary repository file in RPD format, or select **XML** to select a set of MDS XML documents.
4. In the Open Offline dialog, enter the repository password and click **OK**.
5. Use the Compare repositories dialog to review the differences between the two repositories. Figure 17–1 shows the Compare repositories dialog.

Figure 17–1 Compare Repositories Dialog

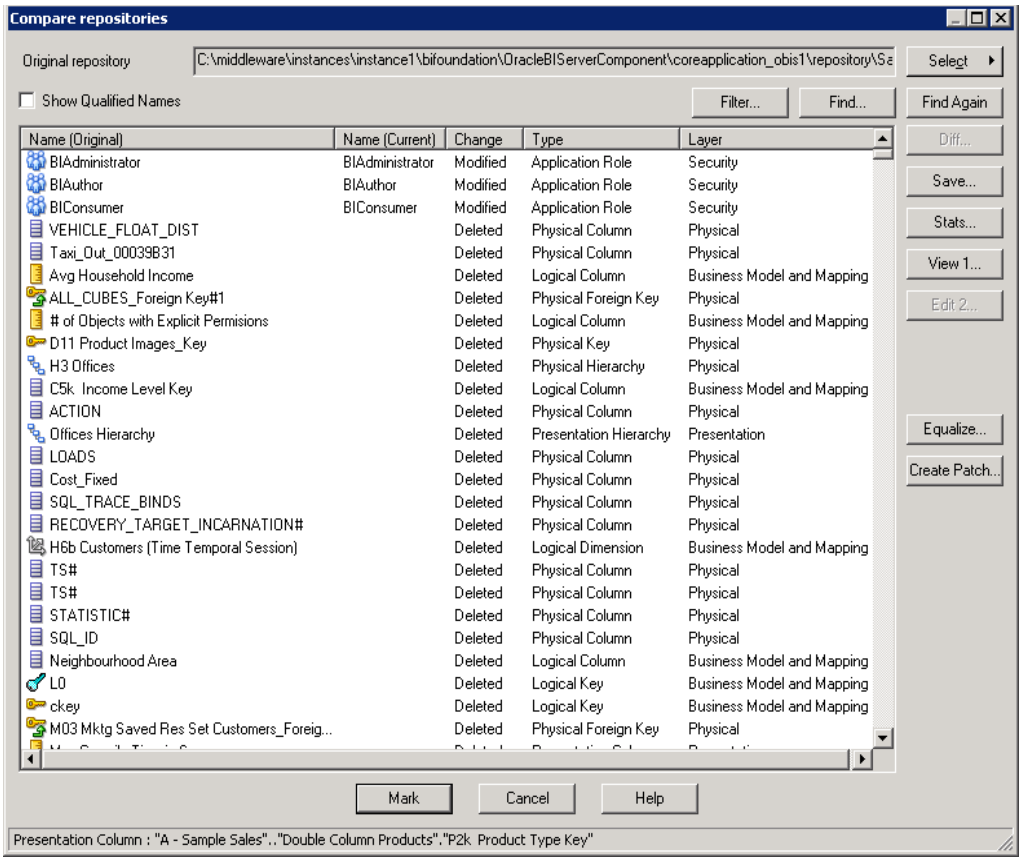


Table 17–1 lists and describes the values in the **Change** column.

Table 17–1 Compare Repositories Dialog: Change Column

Change	Description
Created	Object was created in the current repository and does not exist in the original repository.
Deleted	Object exists in the original repository but has been deleted from the current repository.
Modified	Object exists in the original repository but has been modified in the current repository. Objects marked as Modified display a value in the Name (Current) column. By comparing this column to the Name (Original) column, you can quickly determine if the object has been renamed in the current repository.

Table 17–2 lists and describes some of the buttons in the Compare repositories dialog.

Table 17–2 Compare Repositories Dialog: Buttons

Button	Description
Filter	Opens the Comparison Filter dialog to enable you to filter the objects that appear in the Compare repositories dialog by type of change and type of object. You can specify what you want to appear and what you want to be hidden. If you select **Group created and deleted objects**, the tool filters out the child objects of created and deleted objects, so that only the parent objects are shown. By default, all items are shown.
Find	Search by an object Name and Type (such as Initialization Block).
Select	Enables you to select a repository to compare with the current repository. Select **Repository** from the submenu to select a binary repository file in RPD format, or select **XML** to select a set of MDS XML documents.
Find Again	Search again for the most recent Find value.
Diff	Differences between the selected objects in the current repository and the original repository.
Save	Saves a list of the differences between the two repositories. The differences are saved into a CSV file.
Stats	Provides the number of changes by Change type.
View 1	Opens an object in the original repository in read-only mode.
Edit 2	Opens an object in the current repository in read/write mode.
View 2	Opens an object in the current repository in read-only mode. This button is only displayed when viewing MUD history, or when the current repository is open in read-only mode.
Equalize	Opens the Equalize Objects dialog so that you can model changes to the Upgrade ID of the objects. See "Equalizing Objects" for more information.
Create Patch	Creates a patch file (XML) that contains the differences between the repositories. See "Performing Patch Merges" for more information.
Mark	Closes the Compare Repositories dialog and applies icon overlays to the created and modified objects in the repository's layers. To remove the icon overlays from the objects, from the File menu, choose Turn off Compare Mode. If you used the Filter option to filter the objects that appear in the Compare repositories dialog and then select the Mark button, then only the filtered objects are highlighted in the repository's layers.

Comparing Repositories Using comparerpd

You can also compare repositories and create patch files using the `comparerpd` utility. This feature is especially useful when you want to compare repositories or generate patches on Linux and UNIX systems where the Administration Tool is not available. The compare utility is available on both Windows and UNIX systems.

When you run the `comparerpd` utility, the system also equalizes the repository objects. The equalizing process outputs the *my_current_rpd*_equalized.rpd file, which is an informational file containing the output of the equalization done between the

compared repositories. Equalizing ensures that objects with the same qualified names have the same unique identifiers (UIDs). Objects with the same name but different UIDs are automatically updated to have the same UID (or equalized). Note that even if there are no objects that need to be equalized, the *my_current_rpd*_equalized.rpd file is generated. The *my_current_rpd*_equalized.rpd file is saved to the original repository's location.

The location of the `comparerpd` utility is:

ORACLE_HOME/user_projects/domains/bi/bitools/bin

Syntax

The `comparerpd` utility takes the following parameters:

```
comparerpd [-P modified_rpd_password] -C modified_rpd_pathname
[-W original_rpd_password] -G original_rpd_pathname {-O output_csv_file_name |
-D output_patch_file_name | -M output_mds_xml_directory_name} -A -E -8
```

Where:

`-P` *`modified_rpd_password`* is the repository password for the modified repository, also called the customer or customized repository.

`-C` *`modified_rpd_pathname`* is the name and location of the modified repository.

`-W` *`original_rpd_password`* is the repository password for the original repository.

`-G` *`original_rpd_pathname`* is the name and location of the original repository.

`-O` *`output_csv_file_name`* is the name and location of a csv file where you want to store the repository object comparison output.

`-D` *`output_patch_file_name`* is the name and location of an XML patch file where you want to store the differences between the two repositories.

`-M` *`output_mds_xml_directory_name`* is the top-level directory where you want to store diff information in MDS XML format. Note that a list of removed XML files is stored in the directory tree under the top-level directory at:

```
oracle\bi\server\base\DeletedFiles.txt
```

You can specify an output CSV file using `-O`, an XML patch file using `-D`, or an MDS XML directory tree using `-M`. You cannot specify more than one output type at the same time.

Note that if the patch contains passwords, such as connection pool passwords, the patch file is encrypted using the repository password from the current repository. The current repository password effectively becomes the patch file password. You might need to supply this patch file password when applying the patch, if it is different from the repository password for the original repository.

`-A` is an optional argument that ensures the XML patch file does not contain encrypted passwords for the connection pools. This parameter is used in conjunction with the `-D` parameter.

`-E` is an optional argument that causes UIDs to be used to compare expression text. Using this parameter ensures objects that have the same name but different UIDs are given the same UID. This action makes sure that objects are compared as modified instead of being reported as created and deleted. If `-E` is not specified, strings are used.

`-8` specifies UTF-8 encoding.

Note: The arguments for the *modified_rpd_password* and *original_rpd_password* are optional. If you do not provide password arguments, you are prompted to enter any required passwords when you run the command. To minimize the risk of security breaches, Oracle recommends that you do not provide password arguments either on the command line or in scripts. Note that the password arguments are supported for backward compatibility only, and will be removed in a future release. For scripting purposes, you can pass the password through standard input.

For example:

```
comparerpd -C customer.rpd -G original.rpd -O diff.csv
Give password for customer repository: my_cust_password
Give password for original repository: my_orig_password
```

This example generates a comparison diff file in CSV format called diff.csv from the customer.rpd and original.rpd repositories.

```
comparerpd -C customer.rpd -G original.rpd -D my_patch.xml
Give password for customer repository: my_cust_password
Give password for original repository: my_orig_password
```

This example generates an XML patch file called my_patch.xml from the customer.rpd and original.rpd repositories.

Turning Off Compare Mode

This option enables you to remove marks applied to objects while using the Compare Repositories and Merge Repositories options. The **Turn off Compare Mode** option is only available after you have clicked **Mark** during the **File > Compare** action. If no repository object is marked, this option is not available.

To enable the Turn off Compare Mode option:

- In the Administration Tool, select **File**, then select **Turn off Compare Mode**.

Equalizing Objects

If you have objects in two repositories that have the same name but different Upgrade IDs, you may want to treat them as the same object. To accomplish this, you can use the `equalizerpds` utility to equalize the objects by giving them both the same Upgrade ID. Alternatively, you can equalize objects as part of the merge process.

You can also use the Equalize Objects dialog (available from the Compare repositories dialog) to preview what the repository will look like after you run the `equalizerpds` utility.

This section contains the following topics:

About Equalizing Objects

Objects may need to be equalized because the Administration Tool tracks the history of each repository object using the Upgrade ID of the object. The Upgrade ID is a unique identifier for each object.

Sometimes, the Upgrade ID can change because of user actions or during merge. When this occurs, and a subsequent comparison is done, the Administration Tool treats the new Upgrade ID as a new object, and the missing original Upgrade ID as a deleted object.

For example, assume you have two identical repositories. In one repository, delete a presentation column, then re-create it again. When you compare the two repositories using the Compare repositories dialog, there are two entries for the presentation column: one that shows the old object as deleted, and one that shows the new object as created. Without using the Compare repositories dialog, it is hard to tell that this action occurred, because the Administration Tool typically shows only the object name and properties, not the underlying Upgrade ID.

Note that the Upgrade IDs are not unique, and in rare cases, the repositories that you want to merge might contain the same Upgrade ID. Running the `equalizerpds` utility on the repositories that you want to merge will correct this duplicate Upgrade ID issue and prevent an error while performing the merge.

It is very useful run the `equalizerpds` utility on your repositories before merging them to equalize your changes. Equalizing any opposing changes (such as a column that has been duplicated, and then renamed) cleans up the underlying Upgrade IDs and prevents unintended renaming during the merge.

When you equalize objects, you can lose track of object renames because legitimate object renames become different objects. In other words, intentional renames you did in the repository might be changed to different Upgrade IDs, so subsequent merges erroneously treat the renamed object as a new object. To avoid this situation, enter the before and after names of intentionally renamed objects in a rename map file that you then pass to the utility. The `equalizerpds` utility uses the information in the file to ensure that the original IDs are used in the renamed current objects.

Tip: You can view the Upgrade ID for repository objects using the Query Repository dialog. To do this, follow these steps:

1. Select **Tools**, then select **Query Repository**.
2. Run a query. See "Querying and Managing Repository Metadata" for details.
3. Click **Columns**.
4. Select **Upgrade ID** from the list. You can use the Find button to help locate the Upgrade ID.
5. Click **OK**. A new column showing the Upgrade IDs appears in the Results list.

Note that **Upgrade ID** is not available as a column option unless you have selected **Show Upgrade ID in Query Repository** in the General tab of the Options dialog. See "Setting Administration Tool Options" for more information.

Using the Equalize Objects Dialog

The Equalize Objects dialog provides a preview of what your repository will look like if you run the `equalizerpds` utility on it. The Equalize Objects dialog provides a

convenient way to compare changes related to objects that have the same name, but it does not persist any of the changes.

> **Note:** Using the Equalize Objects dialog can be a very slow process for larger repositories.

To view and use the Equalize Objects dialog:

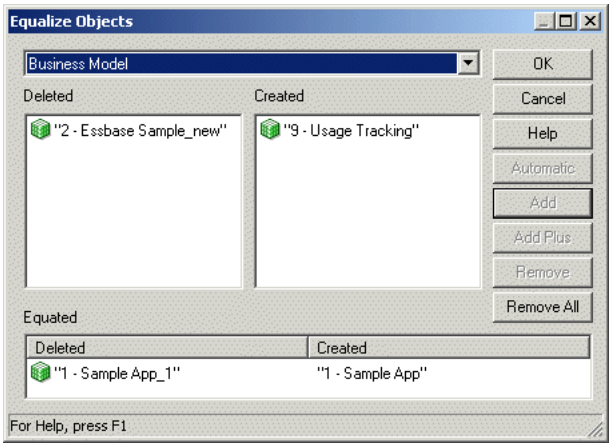
1. In the Administration Tool, open your repository in offline mode.
2. From the **File** menu, select **Compare**.
3. In the Select Original Repository dialog, select the repository you want to compare to the open repository (typically the original repository). Select **Repository** from the submenu to select a binary repository file in RPD format, or select **XML** to select a set of MDS XML documents.
4. In the Open Offline dialog, enter the repository password and click **OK**. The Compare repositories dialog is displayed.
5. Click **Equalize** to display the Equalize Objects dialog.
6. The Equalize Objects dialog shows a list of changes where you may want to consider objects with different Upgrade IDs to be the same object. You can use the following options to model how the changes might get equalized:
   - Click **Automatic** to automatically equalize changes related to objects that have the same name. The changes appear in the Equated table.

     If no changes can be automatically equalized, nothing appears in the table, and the **OK** button remains disabled.
   - Select an object in the Deleted list, then select the equivalent object in the Created list and click **Add** or **Add Plus** to equate the objects. **Add Plus** adds the object along with its child objects to the Equated table, while **Add** simply adds the selected object. For example, if you select a Subject Area and click **Add Plus**, the underlying Presentation Tables and Presentation Columns are added as well.

     After you make a manual selection, the **Automatic** button is disabled.
   - Select a row in the Equated table and select **Remove** or **Remove All** to remove objects from the Equated table. **Remove All** removes the object along with its child objects, while **Remove** simply removes the selected object

     The **Automatic** button is enabled after all manual selections are removed.
7. When you are finished modeling the changes, click **OK**. The changes appear in the Compare Repositories dialog, but the changes do not persist after you close the dialog.

Figure 17–2 Equalize Objects Dialog



Using the equalizerpds Utility

You can use the `equalizerpds` utility to equalize the Upgrade ID of objects in two separate repositories. If objects have the same Upgrade ID, they are considered to be the same object. The utility compares Upgrade IDs from the first repository (typically the original repository) with Upgrade IDs from the second repository (typically the modified repository). Then, the utility equalizes the Upgrade IDs of objects with the same name, using the Upgrade ID from the original repository.

The `equalizerpds` utility is available on both Windows and UNIX systems. However, you can only use `equalizerpds` with binary repositories in RPD format.

The location of the `equalizerpds` utility is:

ORACLE_HOME/user_projects/domains/bi/bitools/bin

Syntax

The `equalizerpds` utility takes the following parameters:

```
equalizerpds [-B original_repository_password] -C original_repository_name
{-E modified_repository_password]|-G} -F modified_repository_name [-I input_UDML_
script_name][-J rename_map_file]
[-O output_repository_name] [-R output_apply_result_file] [-U output_id_map_
file][-Y equalStringSet]
```

Where:

`G` specifies to use the original repository password for the modified repository password.

`rename_map_file` is a text file containing a list of objects that were renamed and that you want to equalize. The format is a tab-separated file with the following columns:

```
TypeName      Name1      Name2
```

For example, to include a logical column in the map file that was renamed from Name1 to Name2, provide the following:

```
ATTRIBUTE     "BusinessModel"."Table"."Name1"     "BusinessModel"."Table"."Name2"
```

Do not cut and paste this example as the foundation for your own file, because the tab separators might not get copied properly. Create a new file with proper tabs.

See "About Values for TypeName" for more information about valid TypeName values.

`equalStringSet` is a set of characters that you want to treat as equal.

Note that the *`original_repository_password`* and *`modified_repository_password`* arguments are optional. If you do not provide these password arguments, you are prompted to enter the passwords when you run the command (password1 and password2). To minimize the risk of security breaches, Oracle recommends that you do not provide password arguments either on the command line or in scripts. Note that the password arguments are supported for backward compatibility only, and will be removed in a future release. For scripting purposes, you can pass the password through standard input.

For example:

```
equalizedrpds -C original.rpd -F modified.rpd -O modified_equalized.rpd
password1: my_original_rpd_password
password2: my_modified_rpd_password
```

In this example, original.rpd is compared with modified.rpd, the Upgrade IDs are equalized using the Upgrade IDs from original.rpd, and the final result is written to modified_equalized.rpd.

> **Note:** Be sure to provide the full pathnames to your repository files, both the input files and the output file, if they are located in a different directory.

About Values for TypeName

Table 17–3 shows the available object types and their corresponding values for TypeName.

Table 17–3 TypeName Values

Object Type	Value for TypeName
Database	DATABASE
Connection Pool	CONNECTION POOL
Physical Catalog	CATALOG
Physical Schema	SCHEMA
Physical Display Folder	PHYSICAL DISPLAY FOLDER
Physical Table	TABLE
Physical Key	TABLE KEY
Physical Foreign Key	FOREIGN KEY
Physical Column	COLUMN
Physical Complex Join	JOIN
Physical Hierarchy	HIERARCHY
Physical Level	PHYSICAL LEVEL
Cube Column	COLUMN
Cube Table	CUBE TABLE
LDAP Server	LDAP SERVER
Custom Authenticator	CUSTOM AUTHENTICATOR
Variable	VARIABLE

Table 17–3 (Cont.) TypeName Values

Object Type	Value for TypeName
Application Role	SECURITY ROLE
User	USER
User Database Signon	USER DATABASE SIGNON
Project	PROJECT
Business Model	SUBJECT AREA
Logical Dimension	DIMENSION
Logical Level	LEVEL
Logical Display Folder	LOGICAL DISPLAY FOLDER
Logical Table	LOGICAL TABLE
Logical Source Folder	LOGICAL SOURCE FOLDER
Logical Table Source	LOGICAL TABLE SOURCE
Logical Column	ATTRIBUTE
Logical Join	ROLE RELATIONSHIP
Logical Key	LOGICAL KEY
Logical Foreign Key	LOGICAL FOREIGN KEY
Presentation Catalog	CATALOG FOLDER
Presentation Table	ENTITY FOLDER
Presentation Column	FOLDER ATTRIBUTE
Presentation Hierarchy	PRESENTATION HIERARCHY
Presentation Level	PRESENTATION LEVEL
Catalog Link	CATALOG LINK
Target Level	CUSTOMER TYPE
List Catalog	LIST CATALOG
Qualified Item	QUALIFIED ITEM
Qualifying Key	QUALIFYING KEY
Sampling Table	SAMPLING TABLE
Segmentation Catalog	SEGMENTATION CATALOG

Merging Repositories

You can use the Merge Repository Wizard in the Administration Tool to merge Oracle BI repositories. You can merge repositories in binary (RPD) format, or MDS XML format. There are three types of merges:

- Full merges are typically used during development processes, when there are two different repositories that need to be merged. The Administration Tool provides a three-way merge feature that lets you merge two repositories that have both been derived from a third, original repository. Full merges can also be used to import objects from one repository into another.
- Patch merges are used when you are applying the differential between two versions of the same repository. For example, you might want to use a patch

merge to apply changes from the development version of a repository to your production repository, or to upgrade your Oracle BI Applications repository.

By default, the `patchrpd` utility's merge functionality uses patch mode. If you want to set up the Merge Repository Wizard to match the `patchrpd` utility's default merge functionality, then you must set up the Merge Repository Wizard to run in "patch" mode.

- Multiuser development merges are used when you are publishing changes to projects using a multiuser development environment. See "About the Multiuser Development Merge Process" for more information.

See also Appendix D, "Merge Rules" for additional information about how repository objects are merged.

This section contains the following topics:

Performing Full Repository Merges

You can use the Administration Tool to merge different repositories. This section describes how to use the full (standard) repository merge feature in the Administration Tool.

This section contains the following topics:

About Full Repository Merges

The merge process typically involves three versions of an Oracle BI repository: the **original** repository, **modified** repository, and **current** repository. The original repository is the original unedited repository (the parent repository), while the modified and current repository are the two changed repositories you want to merge. The current repository is the one currently open in the Administration Tool.

During the merge process, the Administration Tool compares the original repository with the modified repository and the original repository with the current repository. Conflicts occur when there are conflicting changes resulting from the two comparisons. For example, a conflict occurs if you rename object A to B in the modified repository, but you rename object A to C in the current repository.

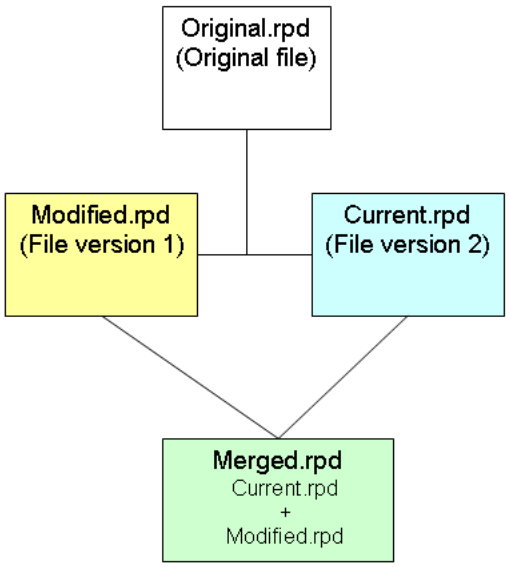
The Merge Repository feature lets you decide on an object-by-object basis which changes you want to keep in the final merged repository. If there are no conflicts, merging is automatic.

There are two types of full merge:

- **Common Parent.** This merge, also called a three-way merge, is useful when you have a common parent repository and two derived repositories. There is a parent (original) repository, and two derived repositories. After the merge, a fourth merged repository is created.

  The example shown in Figure 17–3 assumes you are merging binary (RPD) repositories, but you can also perform this type of merge with MDS XML repositories.

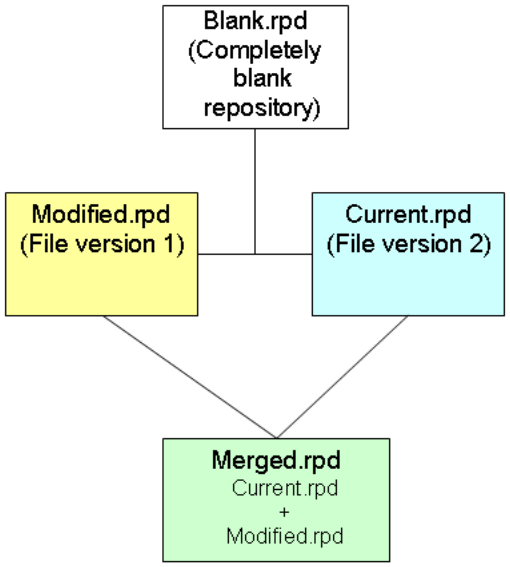
Figure 17–3 Example of Full Merge With a Common Parent with Binary Repositories



- **No Common Parent.** This merge, also called a two-way merge, is useful when you want to import objects from one repository to another. In this case, objects are merged from two different repositories with no common parent. To accomplish this, you perform a three-way merge in the Administration Tool with a completely blank repository as the original repository (see Figure 17–4). This functionality replaces the Import from Repository feature that was deprecated in an earlier release.

  The example shown in Figure 17–4 assumes you are merging binary (RPD) repositories, but you can also perform this type of merge with MDS XML repositories.

Figure 17–4 Example of Full Merge Without a Common Parent with Binary Repositories



Performing Full Repository Merges With a Common Parent

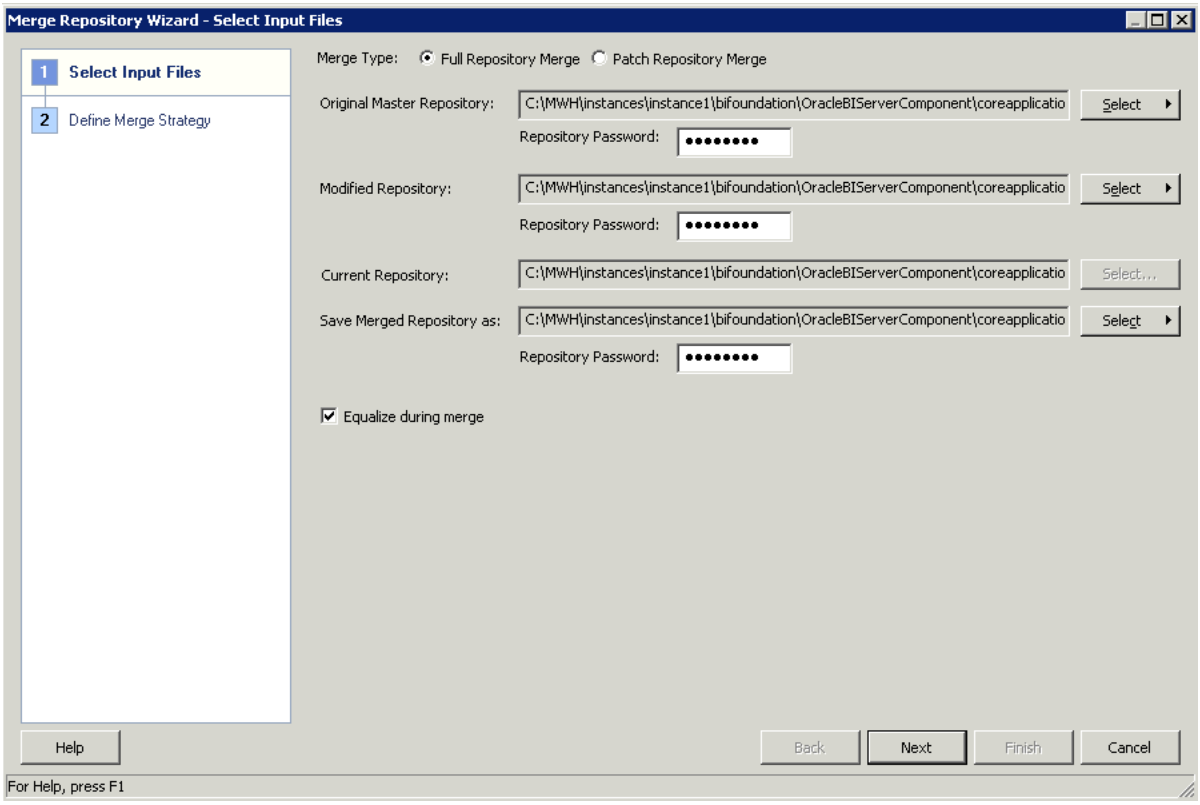
This section explains how to use the Administration Tool to perform a full repository merge with a common parent. Typically, this approach is used when you have an **original** parent repository and would like to merge the changes made to objects in two modified repository versions (**current** and **modified**). Objects that do not exist in the current repository are created as new objects.

To merge two versions of an Oracle BI repository with a common parent:

1. In the Administration Tool, open the **current** repository in offline mode.
2. From the Administration Tool menu, select **File**, then select **Merge**. The Merge Repository Wizard appears.

   Figure 17–5 shows the Merge Repository Wizard.

Figure 17–5 Merge Repository Wizard: Select Input Files Screen (Full Merge)



3. In the Select Input Files screen, for **Merge Type**, select **Full Repository Merge**.
4. Select the original parent repository by clicking **Select** next to **Original Master Repository**. Select **Repository** from the submenu to select a binary repository file in RPD format, or select **XML** to select a set of MDS XML documents.

   For binary repositories, browse to select the original repository, and then click **Open**. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.
5. Provide the password for the original repository in the appropriate **Repository Password** field.

6. Select the modified repository by clicking **Select** next to the **Modified Repository** field. Select **Repository** from the submenu to select a binary repository file in RPD format, or select **XML** to select a set of MDS XML documents.

   For binary repositories, browse to select the modified repository, and then click **Open**. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.

7. Provide the password for the modified repository in the appropriate **Repository Password** field.

8. Optionally, you can change the default name and location of the saved (merged) repository by clicking **Select** next to the **Save Merged Repository as** field. Select **Repository** from the submenu to save the merged repository as a binary repository file in RPD format, or select **XML** to save the merged repository as a set of MDS XML documents.

   For binary repositories, provide a new name and location, and then click **Save**. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.

9. It is a good practice to equalize your changes to clean up underlying object IDs before merging. If you have not yet equalized your changes, select **Equalize during merge** to equalize objects as part of the merge process. Selecting this option may affect merge performance.

   See "Equalizing Objects" for more information about equalizing.

10. Click **Next**. If there are any conflicts, the Define Merge Strategy screen of the Merge Repository Wizard appears. If there are no conflicts, the Merge Repository Wizard closes.

11. The Define Merge Strategy screen displays a decision table that shows conflicts for this merge. See Table 17–4 for details about the elements in this screen.

    To make decisions about whether to include or exclude objects from the merged repository, choose **Current** or **Modified** from the **Decision** list. Choose **Current** to keep the change for the selected object in the current repository, or choose **Modified** to keep the change for the selected object in the modified repository.

    When you select an object in the decision table, the read-only text box below the decision table describes what changes were made to that object in the current repository. In addition, the tree panels at the bottom of the dialog show the affected objects for the selected row. Alternatively, you can select an object in one of the tree views to automatically highlight the corresponding row in the decision table.

    The **Modified** option in the **Decision** list displays a suffix that indicates whether the object in question will be added to or deleted from the merged repository. **Modified (A)** indicates that the object will be added, and **Modified (D)** indicates that the object will be deleted.

    The type of conflict is displayed in the Description column of the Conflicts table. The decision choices you can make depend on the type of conflict shown in this column. The following list shows example results for different types of conflicts:

    - **Added to Current:** Choosing **Current** keeps the new object in the merged repository. Choosing **Modified (D)** deletes the new object from the merged repository.

- **Deleted from Current:** Choosing **Current** keeps the repository as it is without adding the object to the merged repository. Choosing **Modified (A)** adds the object back into the merged repository.
- **Changed in both (different):** The object was not added or deleted, but at least one of its properties was modified. Click the plus sign (+) to the left of the row to view the property that was changed, as well as its value in the original, current, and modified versions of the repository. Property values are only shown for small-length strings. Longer-length strings like descriptions, features, and init strings are not shown.

  Click the option for the value you want to retain in the merged version of the repository. For some properties, such as aliases, you can choose the **Merge Choices** option to merge the properties rather than choose one over the other. This option is only available if the properties can be merged.

Note: You typically do not need to make merge decisions regarding objects that have been added to or deleted from the Modified repository. However, you can view change statistics for this merge to see a summary of changes, including objects that have been added to or deleted from Modified. See Table 17–4 for more information about this feature.

After you make a merge decision, the row for that decision in the table changes from red to black. When all rows have a value in the Decision field, the **Finish** button is enabled.

12. In addition to making merge decisions, you can perform other operations in the Define Merge Strategies screen. See Table 17–4 for details.
13. Click **Finish**.

Table 17–4 lists and describes the elements in the Define Merge Strategies screen.

Table 17–4 Elements in the Define Merge Strategies Screen

Element	Description
Conflicts table	The Conflicts table includes the following columns: ■ **Type:** The type of object for which there is a conflict (for example, Presentation Column). ■ **Name:** The name of the object for which there is a conflict. ■ **Description:** The reason for the conflict, such as Added to Current. See the previous step for a description of different conflict types. ■ **Decision:** Select the decision according to what change you want to keep in the merged repository, such as Current, Modified (A), Modified (D), or By Property. See the previous step for a description of the results of different decisions. For objects with properties that are modified in each repository, a sub-table (grid) is displayed with details of the changed properties. The grid includes the following columns: ■ **Property:** The name of the property that has been modified in each repository. ■ **Original:** The value of the property in the original repository. ■ **Modified:** The value of the property in the modified repository. Select this option to keep this value. ■ **Current:** The value of the property in the current repository. Select this option to keep this value. ■ **Merge Choices:** For some properties, like aliases, you can choose this option to merge the property values rather than choose one or the other.
Show qualified names	When selected, shows fully qualified names for objects in the decision table (for example, "Paint"..."Month Year Ago fact"). **Note:** When the **Show qualified names** option is selected, some of the object names can be too long to fit into the cells of the decision table. Use the mouse to hover over the truncated name to see the full name of the object or property. Alternatively, you can manually resize columns, or double click the column separator to expand the column to the width of the object name.
Set Default Decisions	When clicked, allows you to choose how the Administration Tool resolves conflicts. Use this option when you do not want to set each conflict's decision manually. Select **All** if you want the Administration Tool to resolve all unresolved and resolved conflicts, that is conflicts for which you have already specified decisions. If you choose this option, then the Administration Tool checks the conflict decisions that you have already specified and changes them if necessary. Select **Only undecided** if you want the Administration Tool to resolve only the unresolved conflicts. That is, to assign decisions to all conflicts for which you have not specified decisions. When you select this option, the Administration Tool preserves the conflict decisions that you have already specified.
Check consistency of the merged RPD	When selected, runs a consistency check before saving the merged file.

Table 17–4 (Cont.) Elements in the Define Merge Strategies Screen

Element	Description
Hide object views	Select this option to hide the tree panels in the dialog. The tree panels show the affected objects for the row selected in the conflicts decision table. Hiding the tree panels can improve the performance of the Define Merge Strategies screen.
Save Decisions to File 	Saves a file containing interim changes in the Repository subdirectory so that you can stop work on the merge and continue it later. After saving the changes (decisions), close the Merge repositories dialog by clicking **Cancel**. **Note:** If there are a large number of decisions, you can save time by saving the merge decisions to a CSV file, opening the file in Excel or a text editor, and then modifying the merge decisions manually. Then, you can load the updated merge decisions file in the Define Merge Strategies screen, or include the decision file as an argument in `patchrpd`.
Load Decision File 	Loads a saved decisions file from the Repository subdirectory so that you can continue processing a repository merge. This option is especially useful for resolving conflicts after running an automated patch merge using `patchrpd`. See "Using patchrpd to Apply a Patch" for more information.
Find by Name or Type 	Searches by an object Name and Type (such as Initialization Block).
Find Again 	Searches again for the most recent Find value.
View Change Statistics 	Shows statistics for this merge, such as the number of objects deleted from the Modified repository, the number of objects that were changed in both repositories, and so on.
Details 	Shows the text in the read-only text box below the decision table in a separate window.
View Original/Modified/ Current repository 	Shows properties for the affected object in the selected repository.

Performing Full Repository Merges Without a Common Parent

This section explains how to use the Administration Tool to perform a full repository merge without a common parent. Use this method when you want to import objects from one repository (the **modified** repository) into another (the **current** repository).

Note: In the repository you choose to define as **current**, make sure that the Presentation layer references any Physical layer or Business Model and Mapping layer objects that you want to keep. Objects like business models, databases, and connection pools in the current repository that are not referenced by any Presentation layer objects are discarded during the merge. If necessary, you might want to add a placeholder subject area that references the objects before you perform the merge to ensure the desired objects are kept.

See Appendix D, "Merge Rules" for more information about which objects are retained or discarded during the merge process.

To merge two versions of an Oracle BI repository file without a common parent:

1. If you do not already have a blank repository file to serve as the original repository in the merge, create one, as follows:
   a. In the Administration Tool, select **File**, then select **New Repository**. The Create New Repository Wizard appears.
   b. Provide a name for the repository (for example, blank.rpd).
   c. For **Import Metadata**, choose **No**.
   d. Enter and confirm the repository password you want to use for this repository.
   e. Click **Finish**.
2. Close the blank repository.
3. Open the **current** repository in offline mode. This is the repository that contains the objects you want to import.
4. From the Administration Tool menu, choose **File**, then select **Merge**. The Merge Repository Wizard appears.
5. In the Select Input Files screen, for **Merge Type**, select **Full Repository Merge**.
6. Click **Select** next to **Original Master Repository**, then click **Repository**. Then, browse to select your blank repository file as the original repository and click **Open**. Leave the password field blank.
7. Select the destination repository by clicking **Select** next to the **Modified Repository** field. Select **Repository** from the submenu to select a binary repository file in RPD format, or select **XML** to select a set of MDS XML documents.

   For binary repositories, browse to select the modified repository, and then click **Open**. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.

   This is the repository into which you want to import objects.
8. Provide a password for the modified repository in the appropriate **Password** field.
9. Optionally, you can change the default name and location of the saved (merged) repository by clicking **Select** next to the **Save Merged Repository as** field. Select **Repository** from the submenu to save the merged repository as a binary repository file in RPD format, or select **XML** to save the merged repository as a set of MDS XML documents.

   For binary repositories, provide a new name and location, and then click **Save**. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.

10. Click **Next**. If there are any conflicts, the Define Merge Strategy screen of the Merge Repository Wizard appears. If there are no conflicts, the Merge Wizard continues with the merge process and then closes automatically when finished.
11. The Define Merge Strategy screen displays a decision table that shows conflicts for this merge. To make decisions about whether to include or exclude objects from the merged repository, choose **Current** or **Modified** from the **Decision** list. When you select an object in the decision table, the read-only text box below the decision table describes what changes were made to that object in the current repository.

    Refer to Table 17–4 for information about additional options in the Define Merge Strategy screen, such as saving merge decisions to a comma-separated values (.csv) file.

    After you make a merge decision, the row for that decision in the table changes from red to black. When all rows have a value in the Decision field, the **Finish** button is enabled.
12. Click **Finish**.

Performing Patch Merges

You can use the patch merge to generate an XML patch file that contains only the changes made to a repository. This patch can be then applied to the old (original) version of the repository to create the new version.

This merge method is very useful for development-to-production scenarios, and can also be used for Oracle BI Applications customers to upgrade their repository.

This section explains how to generate a patch that contains the differences between two repositories, and then apply the patch to a repository file.

This section contains the following topics:

- About Patch Merges
- Generating a Repository Patch
- Applying a Repository Patch

About Patch Merges

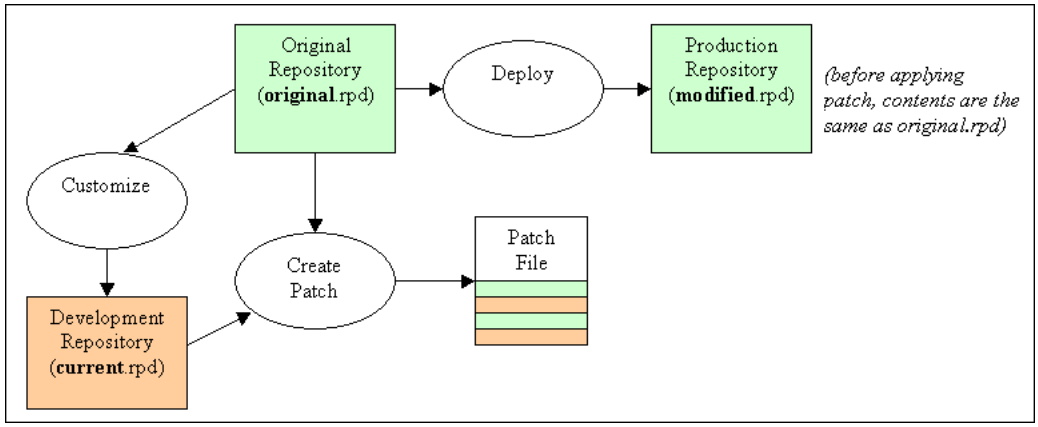
In a patch merge, you create a patch that contains the differences between the **current** repository and the **original** repository. Then, you apply the patch file to the **modified** repository. Differences between the current and original repository must exist in matching existing projects in both repositories.

In a development-to-production scenario, you have an original parent repository, a current repository that contains the latest development changes, and a modified repository that is the deployed copy of the original repository.

To generate a patch, you open the current repository and select the original repository, then create the patch.

Figure 17–6 shows how to create a patch in a development-to-production scenario. The example shown in Figure 17–6 assumes you are creating a patch for binary (RPD) repositories, but you can also create a patch for MDS XML repositories.

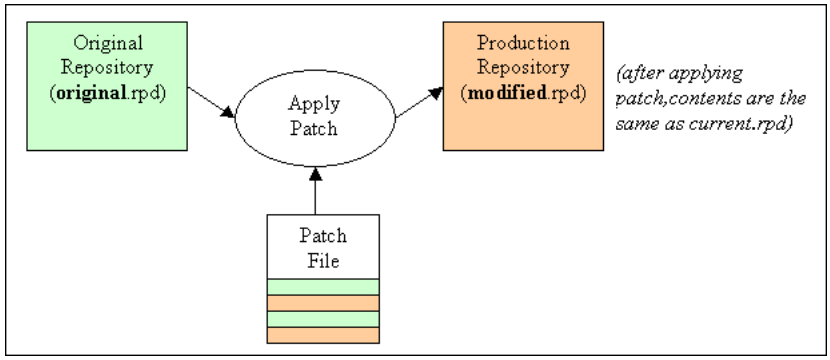
Figure 17–6 Development-to-Production: Creating the Patch



To apply the patch, you open the modified repository and select the original repository, then apply the patch.

Figure 17–7 shows how to apply a patch in a development-to-production scenario. The example shown in Figure 17–7 assumes you are applying a patch to a binary (RPD) repository, but you can also apply a patch to an MDS XML repository.

Figure 17–7 Development-to-Production: Applying the Patch

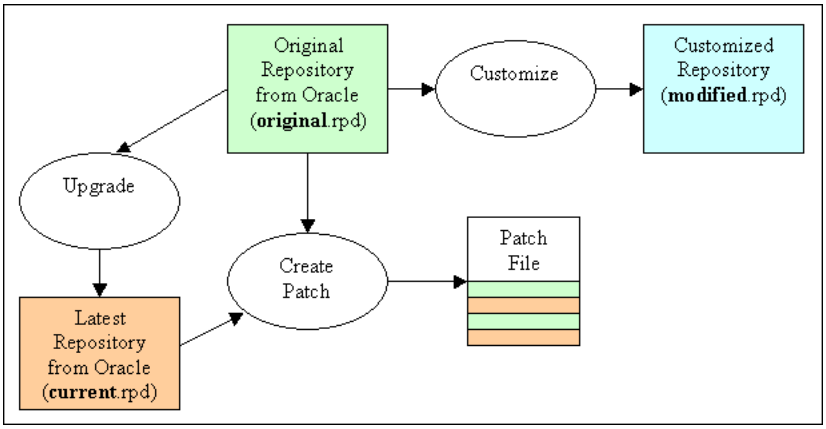


In an Oracle BI Applications repository upgrade scenario, the current repository is the latest version of the repository shipped by Oracle, and the original repository is the original repository shipped by Oracle. The modified repository is the one that contains the customizations you made to the original repository.

To generate a patch, you open the current repository and select the original repository, then create the patch.

Figure 17–8 shows how to create a patch in an Oracle BI Applications repository upgrade scenario. The example shown in Figure 17–8 assumes you are creating a patch for binary (RPD) repositories, but you can also create a patch for MDS XML repositories.

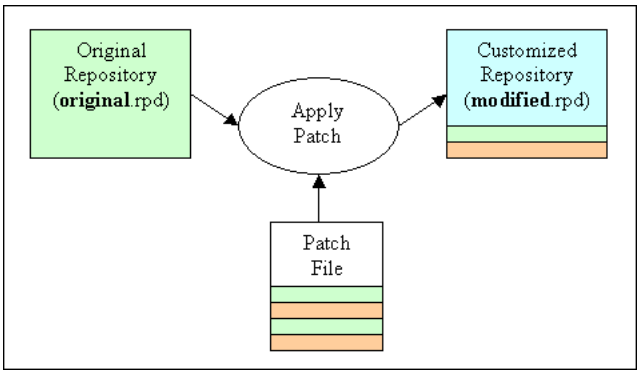
Figure 17–8 Oracle BI Applications Repository Upgrade: Creating the Patch



To apply the patch, you open the modified repository and select the original repository, then apply the patch.

Figure 17–9 shows how to apply a patch in an Oracle BI Applications repository upgrade scenario. The example shown in Figure 17–9 assumes you are applying a patch to a binary (RPD) repository, but you can also apply a patch to an MDS XML repository.

Figure 17–9 Oracle BI Applications Repository Upgrade: Applying the Patch



Generating a Repository Patch

Use the Administration Tool to generate a patch that contains the differences between two repositories.

To generate a patch using the Administration Tool:

1. In the Administration Tool, open the **current** Oracle BI repository in offline mode. In other words, open the updated repository that contains the changes you want to put in the patch.

   Note: In the case that changes are made using projects, metadata changes must be added to an existing project. Changes made to a new project will be lost during the patch merge process.

2. Select **File**, then select **Compare**.

3. Select the **original** Oracle BI repository. Select **Repository** from the submenu to select a binary repository file in RPD format, or select **XML** to select a set of MDS XML documents.
4. In the Open Offline dialog, enter the repository password and click **OK**.
5. It is a good practice to equalize your changes to clean up underlying object IDs before generating a patch. See "Equalizing Objects" for more information.
6. In the Compare repositories dialog, review the changes between the repositories. Then, click **Create Patch**.
7. In the Create Patch dialog, enter a name for the patch file (for example, my_patch.xml) and click **Save**.

Note that if the patch contains passwords, such as connection pool passwords, the patch file is encrypted using the repository password from the current repository. The current repository password effectively becomes the patch file password. You might need to supply this patch file password when applying the patch, if it is different from the repository password for the original repository.

You can also generate a patch at the command line using the `comparerpd` utility. See "Comparing Repositories Using comparerpd" for more information.

Applying a Repository Patch

Use the Administration Tool to apply a patch that contains the differences between two repositories.

Note that you can apply patches from a larger multiuser repository to a smaller subset extract repository. In this case, only the changes in the subset are applied from the patch.

To apply a patch:

1. In the Administration Tool, open the **modified** Oracle BI repository in offline mode. In other words, open the repository on which you want to apply the patch.
2. Select **File**, then select **Merge**. The Merge Repository Wizard appears.
3. For **Merge Type**, select **Patch Repository Merge**. When you select this option, the **Partial Subset Merge** field displays. By default, this field is selected and the patch is applied as a partial subset merge. Clear this option if you want the patch applied to the whole repository.
4. Select the original parent repository by clicking **Select** next to **Original Master Repository**. Select **Repository** from the submenu to select a binary repository file in RPD format, or select **XML** to select a set of MDS XML documents.

   For binary repositories, browse to select the original repository, and then click **Open**. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.

   Note that the original repository cannot be the same as the modified (currently open) repository.
5. Enter the repository password for the original repository.
6. Click **Select** next to **Patch File**. Browse to select the patch file you want to apply, then click **Open**. The patch file must be in XML format.
7. In most cases, you can leave the **Patch Password** field blank. You only need to supply the patch file password when the patch file contains passwords for objects, such as connection pool passwords, and when the patch file password is different

from the original repository password. The patch file password is the same as the password for the current repository.

8. Optionally, you can change the default name and location of the saved (patched) repository by clicking **Select** next to the **Save Merged Repository as** field. Select **Repository** from the submenu to save the merged repository as a binary repository file in RPD format, or select **XML** to save the merged repository as a set of MDS XML documents.

   For binary repositories, provide a new name and location, and then click **Save**. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.

9. Click **Finish**.

Using patchrpd to Apply a Patch You can also apply a patch using the `patchrpd` utility. This feature is especially useful when you want to patch repositories on Linux and UNIX systems where the Administration Tool is not available. The `patchrpd` utility is available on both Windows and UNIX systems. However, `patchrpd` can only be used with binary repositories in RPD format.

Note that unlike the Administration Tool patch feature, `patchrpd` provides an option to apply default decisions for conflicts automatically. It also provides the capability to save all conflicts to a decision file so that you can examine the results and determine whether additional manual changes are needed. See "Using Patchrpd to Automate the Patch Process" for more information.

`Patchrpd` also provides the ability to select the set of merge rules to apply. By default, the merge rules for patches are used, but you can optionally select to use the full (upgrade) merge rules or the multiuser development merge rules.

The location of the `patchrpd` utility is:

ORACLE_HOME/user_projects/domains/bi/bitools/bin

Syntax

The `patchrpd` utility takes the following parameters:

```
patchrpd [-P modified_rpd_password] -C modified_rpd_pathname
[-Q original_rpd_password] -G original_rpd_pathname [-S patch_file_password]
-I patch_file_pathname [-S patch_file_2_password -I patch_file_2_pathname ...]
[-D input_decision_file_pathname] -O output_rpd_pathname
[-V output_decision_file_pathname] [-E] [-M mode] [-R] [-A] [-U] [-N] [-8]
```

Where:

`-P` *`modified_rpd_password`* is the repository password for the modified repository, also called the customer or customized repository.

`-C` *`modified_rpd_pathname`* is the name and location of the modified repository.

`-Q` *`original_rpd_password`* is the repository password for the original repository.

`-G` *`original_rpd_pathname`* is the name and location of the original repository.

`-S` *`patch_file_password`* is the password for the patch file. The patch file password is the same as the password for the current repository. You only need to supply the patch file password when the patch file contains passwords for objects, such as connection pool passwords, and when the patch file password is different from the original repository password.

`-I` *`patch_file_pathname`* is the name and location of the XML patch file you want to apply. You can apply multiple patches by including multiple -I arguments (with paired -S arguments as needed).

`-D` *`input_decision_file_pathname`* is the name and location of a decision file in CSV format that you want to use to affect the behavior of this patch merge. This argument is optional.

`-O` *`output_rpd_pathname`* is the name and location of the RPD output file you want to generate.

`-V` *`output_decision_file_pathname`* is an optional argument that lets you generate a CSV format decision file. The decision file shows all the conflicts from the merge. In other words, the decision file lists the decisions that would have been displayed in the Define Merge Strategy screen of the Merge Wizard in the Administration Tool. The decision file provides a record of all items that could be influenced by user input. This argument must be used in conjunction with the -U argument.

`-E` is an optional argument that enables you to skip the equalize step.

`-M` *`mode`* is an optional argument that enables you to change the mode of the merge. By default, `patchrpd` runs in patch mode, in which as many changes as possible are applied silently. To change this default, then for mode specify "mud" to use merge rules for multiuser development merges, or "upgrade" to use merge rules for full merges. See Appendix D, "Merge Rules" for information about the rules for different types of merges.

By default, the Administration Tool's merge functionality uses full merge. If you want to run the `patchrpd` utility to match the Administration Tool's default merge functionality, then you must specify "upgrade" in the `-M` *`mode`* argument.

`-R` is an optional argument that skips consistency checking for logical columns. This option can speed up the patch process when the patch file does not contain logical to physical column mappings.

`-A` is an optional argument that can be used in multiuser development environments to skip subset patching and instead apply the patch using input repository files.

`-U` is an optional argument that applies default decisions for conflicts automatically so that `patchrpd` can finish successfully. If you do not include this parameter, `patchrpd` displays a warning and exits if a conflict is detected.

`-N` is an optional argument that is used to ignore all non-fatal errors. Examples of non-fatal errors are unresolved objects, duplicated objects, and broken or incorrect expressions.

`-8` specifies UTF-8 encoding.

Note: The arguments for all passwords, including the *`modified_rpd_password`*, *`original_rpd_password`*, and *`patch_file_password`*, are optional. If you do not provide password arguments, you are prompted to enter any required passwords when you run the command. To minimize the risk of security breaches, Oracle recommends that you do not provide password arguments either on the command line or in scripts. Note that the password arguments are supported for backward compatibility only, and will be removed in a future release. For scripting purposes, you can pass the password through standard input.

For example:

```
patchrpd -C customer.rpd -G original.rpd -I patch.xml -O patched.rpd
-V decision.csv -U
Give password for customer repository: my_modified_rpd_password
Give password for original repository: my_original_rpd_password
```

This example applies a patch called patch.xml to the customer.rpd repository, and then generates an output repository called patched.rpd and an output decision file called decision.csv.

Note: Be sure to provide the full pathnames to all files, including the repository files and the XML patch file, if they are located in a different directory.

Querying and Managing Repository Metadata

You can use repository queries to help manage repository metadata in the following ways:

- Examine and update the internal structure of the repository. For example, you can query for objects in the repository based on name, type (such as Logical Column or Presentation Hierarchy), or on a combination of name and type. You can then edit or delete objects that appear in the Results list. You can also create new objects and view parent hierarchies.
- Query a repository and view reports that show such items as all tables mapped to a logical source, all references to a particular physical column, content filters for logical sources, initialization blocks, and security and user permissions.

  For example, you might want to run a report before making any physical changes in a database that might affect the repository. You can save the report to a file in comma-separated value (CSV) or tab-delimited format.
- You can save a query to run again later, or save the query results to an external file. When you save to an external file, the encoding options are ANSI, Unicode, and UTF-8.

This section contains the following topics:

Querying the Repository

You can query for objects in the repository using the Query Repository tool. You can also construct a filter to filter the results, save a query, run a previously saved query, or create new repository objects.

To query a repository:

1. In the Administration Tool, open your repository.
2. Select **Tools**, then select **Query Repository**.
3. In the Query Repository dialog, complete the query information using Table 17–5 as a guide.
4. Click **Query**.

Table 17–5 lists the options available in the Query Repository dialog.

Table 17–5 Query Repository Options

Option	Description
Name	Use this option to search by object name. You can use an asterisk (*) wildcard character to specify any characters. The wildcard character can represent the first or last characters in the search string. Searches are not case sensitive.
Type	Select a type from the list to narrow your search to a particular type of object, or select All Types to query all objects. The list does not contain objects such as aggregate rules, logical source folders, privilege packages, and other objects that are considered internal objects.
Filter	Click **Filter** to create or edit a filter for your query. After you create a filter, the filter criteria appear in the box to the left of the button. See "Constructing a Filter for Query Results" for more information.
Query	Click **Query** when you are ready to submit your query.
Save Query As	Click **Save Query As** to save your query to run again later. Enter the name of the saved query in the **Save query as** field, then click **Save**.
Saved Queries	Click **Saved Queries** to view or run previously saved queries. You can also delete saved queries. To run a previously saved query, select the row that contains the query you want to run and click **Select**, or double-click the row. The **Saved Queries** option is only available if you have previously saved queries.
Edit	After executing a query, select an object from the Results list and click **Edit** to edit an object in the list of query results. Not all repository objects can be edited from the results list (for example, privilege objects and user database sign-on objects). If an object cannot be edited from the results list, **Edit** is not available.
Delete	After executing a query, select one or more objects in the Results list and click **Delete** to delete the objects. After you confirm the deletion, the objects are deleted from your metadata repository.
New	Use this option to create new repository objects. First, select the type of object you want to create from the **Type** list, then click **New**. This option is not available when **All Types** is selected. The dialogs that appear depend on the object type that you select. For more information, refer to the sections that describe how to create that object. Note that if you choose to create a new logical dimension, you must choose whether to create a dimension with a level-based hierarchy, or a parent-child-hierarchy. Similarly, if you choose to create a new Oracle OLAP hierarchy, you must select whether you want to create a level-based or value-based hierarchy.
Show Parent	After executing a query, select an object in the Results list and click **Show Parent** to view the parent hierarchy of an object. If the object does not have a parent, a message appears. You cannot use Show Parent with users or application roles. In the Parent Hierarchy dialog, you can edit or delete objects. Note that if you delete an object from this dialog, any child objects of the selected object are also deleted.
Mark	After executing a query, select one or more objects in the Results list and click **Mark** to mark the selected objects. To unmark the objects, select them and click **Mark** again. Marking objects makes them easier to visually identify as you develop metadata.
Set Icon	After executing a query, select one or more objects in the Results list and click **Set Icon** to select a different icon for the objects. You can set special icons for objects to help visually identify them as having common characteristics. For example, you might want to pick a special icon to identify columns that will be used only by a certain user group. To change the icons back to the original icons, select the objects and click **Set Icon** again. Then, select **Remove associated icon** and click **OK**.

Table 17–5 (Cont.) Query Repository Options

Option	Description
GoTo	After executing a query, select one or more objects in the Results list and click **GoTo** to go to the objects in the Administration Tool view of the repository. The selected objects appear highlighted in the Physical, Business Model and Mapping, or Presentation layer. Note that the Query Repository dialog closes when you choose this option.
Save	After executing a query, click Save to save query results to an external file. Then, in the Save As dialog, provide a name, file type, and encoding value for the file, then click **Save**.
Columns	Click **Columns** to add additional columns of information to the results. Then, select the columns you want from the list and click **OK**. Note that in the Select Columns dialog, you can re-order the columns by selecting a checked column and clicking **Up** or **Down**. Select **Show Upgrade ID in Query Repository** in the General tab of the Options dialog to enable Upgrade ID as a results column. See "Setting Administration Tool Options" for more information.
Show Qualified Name	Use this option to display the fully qualified name of the objects found by the query. For example, if you query for logical columns, the default value in the **Name** column of the Results list is the column name. However, if you select **Show Qualified Names**, the value in the **Name** list changes to `businessmodel.logicaltable.column`.

Constructing a Filter for Query Results

Use the Query Repository Filter dialog to filter the results in the Results list of the Query Repository dialog.

The Query Repository Filter dialog contains five columns: an Item column and its operator or selection column, a Value column and its operator or selection column, and a Delete column that lets you delete the selected filter.

To construct a filter:

1. In the Administration Tool, select **Tools**, then select **Query Repository**.
2. In the Query Repository dialog, select an item in the **Results** list or select an item from the **Type** list, and then click **Filter**.
3. In the Query Repository Filter dialog, click the **Item** field. The **Item** list contains the items by which you can filter.

   Select **Show Upgrade ID in Query Repository** in the General tab of the Options dialog to enable filtering by Upgrade ID. See "Setting Administration Tool Options" for more information.
4. In the **Item** list, select the filter that you want to apply to the Results or Type object you selected in Step 2. Then, adjust or enter information in the **Value** column, as appropriate.

   You can construct multiple filters. When you do, the **Operator** field becomes active. When the **Operator** field is active, you can set `AND` and `OR` conditions.
5. Click **OK** to return to the Query Repository dialog. The filter appears in the box to the left of the **Filter** button.

Note: If you are constructing a complex filter, you might want to click **OK** after adding each constraint to verify that the filter construction is valid for each constraint.

Example 17–1 and Example 17–2 show how to create different kinds of filters.

Example 17–1 Viewing All Databases Referenced In a Business Model

The following example shows how to create a filter that lets you view all databases referenced in a particular business model.

1. In the Query Repository dialog, select **Database** from the **Type** list, and then click **Filter**.
2. In the Query Repository Filter dialog, click the **Item** field, and then select **Related to**.
3. Click the ellipsis button to the right of the **Value** field, and in the list, choose **Select object**.
4. In the Select dialog, select the business model by which you want to filter, and then click **Select**. Your selection appears in the **Value** field.
5. Click **OK** to return to the Query Repository dialog. The filter appears in the box to the left of the **Filter** button.
6. Click **Query**. The Results list shows the databases referenced by the business model you selected.

Example 17–2 Viewing All Presentation Columns Mapped to a Logical Column

The following example shows how to create a filter that lets you view all presentation columns mapped to a particular logical column.

1. In the Query Repository dialog, select **Presentation Column** from the **Type** list, and then click **Filter**.
2. In the Query Repository Filter dialog, click the **Item** field, and then select **Column**.
3. Click the ellipsis button to the right of the **Value** field, and in the list, choose **Select object**.
4. In the Select dialog, select the column by which you want to filter, and then click **Select**. Your selection appears in the **Value** field.
5. Click **OK** to return to the Query Repository dialog. The filter appears in the box to the left of the **Filter** button.
6. Click **Query**. The Results list shows the presentation columns mapped to the logical column you selected.

Example 17–3 Nested Queries

The following example shows nested queries, where the filter itself is another query.

1. In the Query Repository dialog, select **Logical Column** from the **Type** list, and then click **Filter**.
2. In the Query Repository Filter dialog, click the **Item** field, and then select **Related to**.

3. Click the ellipsis button to the right of the **Value** field, and in the list, choose **Set Condition for Physical Column**.
4. In the new Query Repository Filter dialog, click the **Item** field, and then select **Source column**.
5. Click the ellipsis button to the right of the **Value** field, and in the list, choose **Select Object**.
6. In the Browse dialog, select a source physical column (for example, Column A) and click **Select**.
7. Click **OK** in the Query Repository Filter dialog for the subquery condition. This subquery queries all aliases for the source column you selected.
8. In the Query Repository Filter dialog for the main query, click the **Item** field in the next row and then select **Related to**.
9. Click the ellipsis button to the right of the **Value** field, and in the list, choose **Select Object**.
10. In the Browse dialog, select the same source physical column (for example, Column A) and click **Select**.
11. Select OR from the **Operator** list.
12. Click **OK** to return to the Query Repository dialog. The filter appears in the box to the left of the **Filter** button.
13. Click **Query**. The Results list shows a list of logical columns related to either Column A, or aliases of Column A.

Querying Related Objects

The Query Related Objects feature enables you to query objects related to one or more objects that you select from the Physical, Business Model and Mapping, or Presentation layer.

You can only use this feature with objects selected from the same layer. For example, you cannot query objects related to both a Physical layer object and a Business Model and Mapping layer object.

To query objects related to a selected object:

1. In the Administration Tool, open your repository.
2. Select one or more objects from a single layer (for example, a set of logical columns from the Business Model and Mapping layer). The objects you select must all be of the same type.
3. Right-click the objects and select **Query Related Objects**.
4. From the right-click submenu, select an object type to narrow your search to a particular type of object, or select **All Types** to query all objects related to your source objects. If you have previously made queries for this source object type, the three most recent queries are available at the top of the submenu.

   After you select an object type, the Query Related Objects dialog is displayed, showing the objects related to your source objects in the **Name** list.

Table 17–6 lists the options available in the Query Repository dialog.

Table 17–6 Query Related Objects Options

Option	Description
Mark	Select one or more objects in the Name list and click **Mark** to mark the selected objects. To unmark the objects, select them and click **Mark** again. Marking objects makes them easier to visually identify as you develop metadata.
Set Icon	Select one or more objects in the Name list and click **Set Icon** to select a different icon for the objects. You can set special icons for objects to help visually identify them as having common characteristics. For example, you might want to pick a special icon to identify columns that will be used only by a certain user group. To change the icons back to the original icons, select the objects and click **Set Icon** again. Then, select **Remove associated icon** and click **OK**.
Show Qualified Name	Use this option to display the fully qualified name of the objects found by the query. For example, if you query for logical columns, the default value in the **Name** list is the column name. However, if you select **Show Qualified Names**, the value in the **Name** list changes to `businessmodel.logicaltable.column`.
Show Parent	Select an object in the Name list and click **Show Parent** to view the parent hierarchy of an object. If the object does not have a parent, a message appears. You cannot use Show Parent with users or application roles. In the Parent Hierarchy dialog, you can edit or delete objects. Note that if you delete an object from this dialog, any child objects of the selected object are also deleted.
GoTo	Select one or more objects in the Name list and click **GoTo** to go to the objects in the Administration Tool view of the repository. The selected objects appear highlighted in the Physical, Business Model and Mapping, or Presentation layer. Note that the Query Related Objects dialog closes when you choose this option.

Changing the Oracle BI Repository Password

Each Oracle BI repository has a password that is used to encrypt its contents. You create the repository password when you create a new Oracle BI repository file, or when you upgrade a repository from a previous release of Oracle Business Intelligence.

You can change the repository password using the Administration Tool in offline mode, or using the `obieerpdpwdchg` utility. You cannot change the repository password when the repository is open in the Administration Tool in online mode.

The `obieerpdpwdchg` utility is especially useful when you want to change the repository password on Linux and UNIX systems where the Administration Tool is not available.

After you change the repository password using the Administration Tool or `obieerpdpwdchg`, you must also publish the updated repository and specify the new password in Fusion Middleware Control. Specifying the repository password in Fusion Middleware Control enables the password to be stored in an external credential store, so that the Oracle BI Server can retrieve it to load the repository.

Note: If you are using the SampleAppLite.rpd or SampleApp.rpd sample repository, you must change the default password the first time you open it in the Administration Tool, for security reasons. See "About the SampleApp.rpd Demonstration Repository" for more information about the sample repository.

This section contains the following topics:

- Changing the Oracle BI Repository Password Using the Administration Tool
- Changing the Oracle BI Repository Password Using the obieerpdpwdchg Utility

Changing the Oracle BI Repository Password Using the Administration Tool

Follow these steps to change the repository password using the Administration Tool, and then publish the modified repository using the "Upload Repository Command":

1. Open the repository in the Administration Tool in offline mode.
2. Select **File**, then select **Change Password**.
3. Enter the current (old) password.
4. Enter the new password and confirm it. The repository password must be longer than five characters and cannot be empty.
5. Click **OK**.
6. Save and close the repository.
7. Open a Web browser and log in to Fusion Middleware Control from the computer where the updated repository is located.
8. In the navigation tree, expand Business Intelligence and then click **coreapplication** to display the Business Intelligence Overview page.
9. Display the Repository tab of the Deployment page.
10. Click **Lock and Edit Configuration**.
11. Click **Browse** next to Repository File. Then, select the updated repository file and click **Open**.
12. Enter the new (updated) repository password in the **Repository Password** and the **Confirm Password** fields.

    Make sure to specify the password that has been set in the repository. If the passwords do not match, the Oracle BI Server fails to start, and an error is logged in nqserver.log.
13. Click **Apply**, then click **Activate Changes**.
14. Return to the Business Intelligence Overview page and click **Restart**.

Changing the Oracle BI Repository Password Using the obieerpdpwdchg Utility

Follow these steps to change the repository password using the `obieerpdpwdchg` utility, and then publish the modified repository using the "Upload Repository Command":

1. Navigate to the `obieerpdpwdchg` utility, which is located here:

   ORACLE_HOME/user_projects/domains/bi/bitools/bin

2. Type the following arguments for `obieerpdpwdchg`:
   - -I *name_and_path_of_existing_repository*
   - -O *name_and_path_of_new_repository*

   Then, enter the current (old) password and the new password when prompted. The repository password must be longer than five characters and cannot be empty. For example:

   ```
   obieerpdpwdchg  -I my_repos.rpd -O my_changed_repos.rpd
   Please enter the repository password:

   Please enter a new repository password:
   ```

   Note that passwords are masked on the command line unless you include the `-C` option in the command to disable masking.
3. Open a Web browser and log in to Fusion Middleware Control from the computer where the updated repository is located.
4. In the navigation tree, expand Business Intelligence and then click **coreapplication** to display the Business Intelligence Overview page.
5. Display the Repository tab of the Deployment page.
6. Click **Lock and Edit Configuration**.
7. Click **Browse** next to Repository File. Then, select the updated repository file and click **Open**.
8. Enter the new (updated) repository password in the **Repository Password** and the **Confirm Password** fields.

   Make sure to specify the password that has been set in the repository. If the passwords do not match, the Oracle BI Server fails to start, and an error is logged in nqserver.log.
9. Click **Apply**, then click **Activate Changes**.
10. Return to the Business Intelligence Overview page and click **Restart**.

18

Using Expression Builder and Other Utilities

This chapter describes Expression Builder and provides instructions for creating constraints, aggregations, and other definitions within the Oracle BI repository. It also describes the various utilities and wizards contained in the Oracle BI Administration Tool.

This chapter contains the following topics:

Using Expression Builder

You can use the Expression Builder dialogs in the Administration Tool to create constraints, aggregations, and other definitions within a repository. Expression Builder provides automatic color highlighting and other formatting enhancements to make expressions easier to build and to read.

The expressions you create with Expression Builder are similar to expressions created with SQL. Except where noted, you can use all expressions constructed with Expression Builder in SQL queries against the Oracle BI Server.

For information about using SQL with Expression Builder, and for information about the SQL functions supported by the Oracle BI Server, see Appendix C, "Logical SQL Reference."

This section contains the following topics:

About the Expression Builder Dialogs

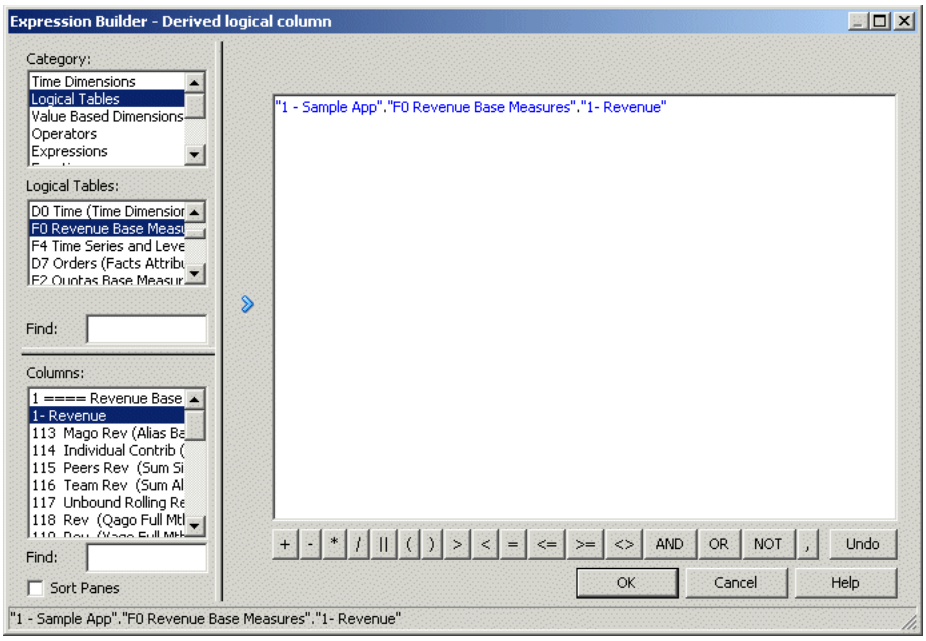
You can access Expression Builder from the following dialogs:

- Logical Table Source—Content tab
- Logical Table Source—Column Mapping tab
- Logical Column—General tab
- Logical Column—Aggregation tab

- Logical Foreign Key
- Physical Foreign Key
- Session Variable
- Repository Variable

Figure 18–1 shows Expression Builder.

Figure 18–1 Example Expression Builder Dialog



The dialog contains the following sections:

- The edit pane on the right hand side of the dialog lets you edit the current expression.
- The toolbar at the bottom of the dialog contains commonly used expression operators.
- In the left section of the dialog:
  - The top pane is the Category pane. It displays categories that are appropriate for the dialog from which you accessed Expression Builder.
  - The middle pane displays a list of available items for the category you selected in the Category pane.

    You can use the **Find** field below the middle pane to display specific values in the middle pane.
  - The lower pane is the Building Blocks pane. It displays the individual building blocks for the item you selected in the middle pane.

    You can use the **Find** field below the lower pane to display specific values in the lower pane.

When creating expressions in Expression Builder, you select a category from the Category pane and values are displayed in the lower panes depending on the value selected in the Category pane. When you type a value into a Find field, it filters out the non-matching strings and displays matching strings only. After typing search criteria

in a Find field, you can move up and down the list using the scroll bar, and use the tab key to move between the Find fields. To return to the full list of results, delete the string from the Find field.

Note that you can only enter text in the Find field that matches the text of one of the available strings. For example, if the available string options begin with A11, A12, and A13, the text you enter in the Find field must begin with A.

When you locate the building block you want to insert into the expression, select it and do one of the following:

- Click the arrow button
- Double click the item
- Press Enter on your keyboard

The building block you selected appears in the expression in the edit pane.

When you first open Expression Builder, the items are not sorted. When selected, the **Sort Panes** option sorts all items in the panes. As soon as you select this option, the panes are automatically redrawn without changing the contents of the panes or your filtering criteria.

About the Expression Builder Toolbar

The toolbar is located at the bottom of Expression Builder.

Table 18–1 describes each button and its function in an expression.

Table 18–1 Expression Builder Toolbar

Operator	Description
+	Plus sign for addition.
-	Minus sign for subtraction.
*	Multiply sign for multiplication.
/	Divide by sign for division.
||	Character string concatenation.
(	Open parenthesis.
)	Close parenthesis.
>	Greater than sign, indicating values higher than the comparison.
<	Less than sign, indicating values lower than the comparison.
=	Equal sign, indicating the same value.
<=	Less than or equal to sign, indicating values the same or lower than the comparison.
>=	Greater than or equal to sign, indicating values the same or higher than the comparison.
<>	Not equal to, indicating values higher or lower, but different.
AND	AND connective, indicating intersection with one or more conditions to form a compound condition.
OR	OR connective, indicating the union with one or more conditions to form a compound condition.
NOT	NOT connective, indicating a condition is not met.

Table 18–1 (Cont.) Expression Builder Toolbar

Operator	Description
,	Comma, used to separate elements in a list.

About the Categories in the Category Pane

The categories that appear in the Category pane vary, depending on the dialog from which you accessed Expression Builder.

Table 18–2 describes the categories that may appear.

Table 18–2 Expression Builder Categories in the Category Pane

Category Name	Description
Aggregate Content	Contains the available aggregate functions. Aggregate sources must use one of the functions listed here to specify the level of their content.
Time Dimensions	Contains the time dimensions configured in the business model. If no time dimensions exist in a business model, or if time dimensions are not pertinent to a particular Expression Builder, the Time Dimensions category is not displayed. When you select the Time Dimensions category, each configured time dimension appears in the middle pane, and each level for the selected dimension appears in the lower pane.
Logical Tables	Contains the logical tables configured in the business model. If logical tables are not pertinent to a particular Expression Builder, the Logical Tables category is not displayed. When you select the Logical Tables category, each logical table in the business model appears in the middle pane, and each column for the selected logical table appears in the lower pane.
Value Based Dimensions	Contains the dimensions with parent-child hierarchies configured in the business model. If no dimensions with parent-child hierarchies exist in a business model, or if dimensions with parent-child hierarchies are not pertinent to a particular Expression Builder, the Value Based Dimensions category is not displayed. When you select the Value Based Dimensions category, the configured dimensions with parent-child hierarchies appear in the middle pane. No lower pane exists for this category.
Logical Levels	Contains the related logical levels. If level-based dimensions are not pertinent to a particular Expression Builder, the Logical Levels category is not displayed. When you select the Logical Levels category, you can then select the appropriate logical dimension (level-based) in the middle pane, and the level itself in the lower pane.
Physical Tables	Contains the related physical tables. If physical tables are not pertinent to a particular Expression Builder, the Physical Tables category is not displayed.
Operators	Contains the available SQL logical operators.
Expressions	Contains the available expressions.
Functions	Contains the available functions. The functions that appear depend on the object you selected.
Constants	Contains the available constants.
Types	Contains the available data types.

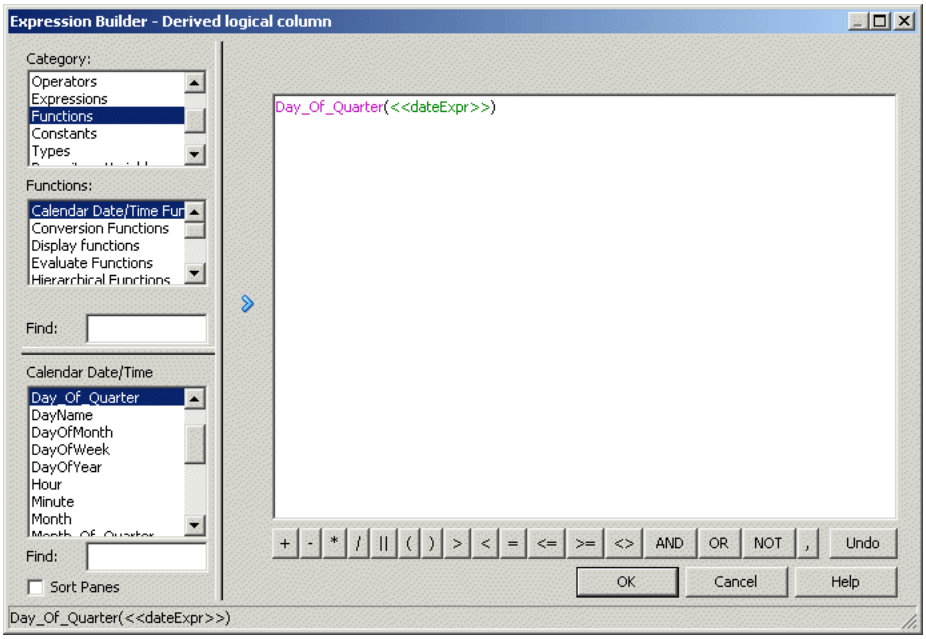
Table 18–2 (Cont.) Expression Builder Categories in the Category Pane

Category Name	Description
Repository Variables	Contains the available repository variables. If no repository variables are defined, this category does not appear.
Session Variables	Contains the available system session and non-system session variables. If no session variables are defined, this category does not appear.

Setting Up an Expression

Figure 18–2 shows the Expression Builder dialog for a derived logical column.

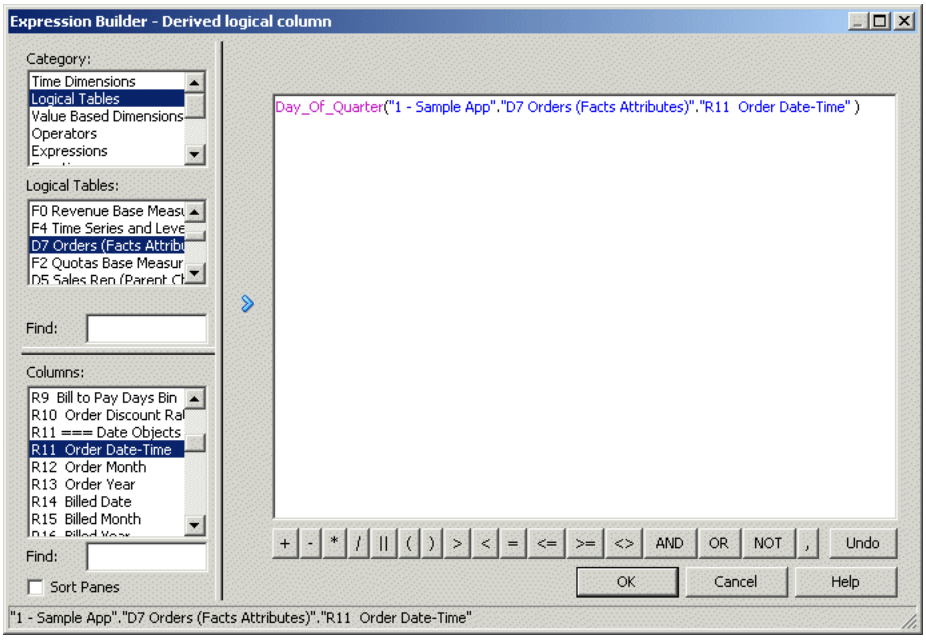
Figure 18–2 Expression Builder for Derived Logical Columns



To set up an expression, select Functions from the Category pane, select a function type from Functions pane, then select a function from the lower pane. Double-click the function you want to use to paste it in the edit pane. Then, in the edit pane, click once between the parentheses of the function to select that area as the insertion point for adding the argument of the function.

To paste a logical column at the insertion point, select Logical Tables from the Category pane, select the table you want to use in the Logical Tables pane, and then double-click the logical column in the lower pane to paste the logical column at the insertion point as the argument of the function in the edit pane. Figure 18–3 shows where the expression appears in the edit pane.

Figure 18–3 Example Logical Column Function in the Editing Pane



Navigating Within Expression Builder

Use the following procedure to navigate within Expression Builder.

To navigate within Expression Builder:

1. In the Category pane, select the appropriate category for the expression you want to build.

   The available expression types for the selected category appear in the middle pane.

2. Select the appropriate item for the expression you want to build.

   The available building blocks for the selected item appear in the lower pane.

3. Double-click a building block to display it in the edit pane.

4. To insert an operator into the expression, click an operator on the Expression Builder toolbar.

Building an Expression

Use this procedure to build an expression in Expression Builder.

To build an expression:

1. Navigate to the individual building blocks you want in the expression.

   The Syntax bar at the bottom of the Expression Builder dialog shows the syntax for the expression.

   For example: `BETWEEN <<Upper Bound>> AND <<Lower Bound>>`

2. Add the building blocks to the edit pane.

3. Edit the building blocks to reflect the expression you want.

4. Use the Expression Builder toolbar to insert operators into the expression.

5. Repeat the preceding steps until the expression is complete, and then click **OK**.

The Administration Tool displays a message for any syntax errors in the expression. When the expression is syntactically correct, the Administration Tool adds the expression to the dialog from which you accessed Expression Builder.

Note that if the parameter `PREVENT_DIVIDE_BY_ZERO` is set to `YES` in NQSConfig.INI, the Oracle BI Server prevents errors in divide-by-zero situations, even for Answers column calculations. The Oracle BI Server creates a divide-by-zero prevention expression using `nullif()` or a similar function when it writes the physical SQL. Because of this, you do not have to use `CASE` statements to avoid divide-by-zero errors, as long as `PREVENT_DIVIDE_BY_ZERO` is set to `YES` (the default value).

See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about configuration settings.

About the INDEXCOL Conversion Function

The `INDEXCOL` function enables you to build a derived logical column. Selecting `INDEXCOL` automatically generates the following function template:

```
IndexCol( <<integer literal>>, <<expr1>> [, <<expr2>>, ?-] )
```

Note: The argument `integer literal` can also be a session variable, an arithmetic expression, or a `CASE WHEN` statement (evaluation must be possible without reference to back-end data).

See "INDEXCOL" for more information.

Using Administration Tool Utilities

The Administration Tool provides several utilities and wizards that perform functions like renaming objects, persisting aggregates, and externalizing strings.

This section contains the following topics:

- Using the Replace Column or Table Wizard
- Using the Oracle BI Event Tables Utility
- Using the Externalize Strings Utility
- Using the Rename Wizard
- Using the Update Physical Layer Wizard
- Generating Documentation of Repository Mappings
- Generating a Metadata Dictionary
- Providing Access to Metadata Dictionary Information
- Removing Unused Physical Objects
- Persisting Aggregates
- Using the Oracle BI Summary Advisor Wizard
- Using the Convert Presentation Folders Utility
- Generating a List of Logical Column Types
- Comparing Logical Column Types
- Fixing Upgrade IDs
- Setting Permissions In Bulk

Using the Replace Column or Table Wizard

The Replace Column or Table Wizard automates the process of replacing physical columns or tables in logical table sources. For example, if you have purchased Oracle BI Applications, you can update your logical table sources to map to a different database type. You can also use this utility to change logical table source mappings from a development table to a production table.

You can use the Replace Column or Table Wizard to replace a single column (within the same table), or to replace an entire table. If you replace a table, you must map all the columns in the table.

To replace a physical column in logical table sources:

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Replace Column or Table in Logical Table Sources** and click **Execute**.
3. In the Select Object screen, select **Replace single column only**.
4. In the left pane, select the physical column that you want to replace. You must select a valid column. For example, you cannot select a column that is used in a logical table source that has more than one table as a source.
5. In the right pane, select the physical column that you want to use as a replacement for the original column. Then, click **Next**.
6. The Select Sources screen shows all logical table sources that map to the physical column you selected. Select the logical table sources in which you want to change the physical column mapping. Select **Show Qualified Names** to see the full context for each source.

   If you select an invalid logical table source, or in other words, one that cannot be used for replacement, a message appears explaining why that source cannot be used, and the check box for that source is disabled.

   Note that invalid logical table sources do not appear in the list when **Hide unusable logical table sources in Replace wizard** has been selected in the General tab of the Options dialog. Instead, the **Info** button is displayed when a logical table source that maps to that column does not appear in the list. Click **Info** to see details on why the physical objects could not be replaced in the logical table source or sources.

   The Select Sources screen only appears if there are multiple logical table sources that map to the physical column you selected.

   Click **Next** after you have selected logical table sources.
7. When the repository is open in online mode, the Checkout screen appears. In online mode, objects need to be checked out before you can make changes to them. Click **Next** to check out the necessary objects.
8. The Finish screen displays a summary of the objects that will be replaced. If you want to make changes, click **Back**, or select a particular step from the navigation panel.
9. Click **Finish**.

To replace a physical table in logical table sources:

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Replace Column or Table in Logical Table Sources** and click **Execute**.
3. In the Select Object screen, select **Replace whole table**.

4. In the left pane, select the physical table that you want to replace.
5. In the right pane, select the physical table that you want to use as a replacement for the original table. Then, click **Next**.
6. The Select Sources screen shows all logical table sources that map to the physical table you selected. Select the logical table sources in which you want to change the physical table mapping. Select **Show Qualified Names** to see the full context for each source.

   If you select an invalid logical table source, or in other words, one that cannot be used for replacement, a message appears explaining why that source cannot be used, and the check box for that source is disabled.

   Note that invalid logical table sources do not appear in the list when **Hide unusable logical table sources in Replace wizard** has been selected in the General tab of the Options dialog. Instead, the **Info** button is displayed when a logical table source that maps to that column does not appear in the list. Click **Info** to see details on why the physical objects could not be replaced in the logical table source or sources.

   The Select Sources screen only appears if there are multiple logical table sources that map to the physical table you selected.

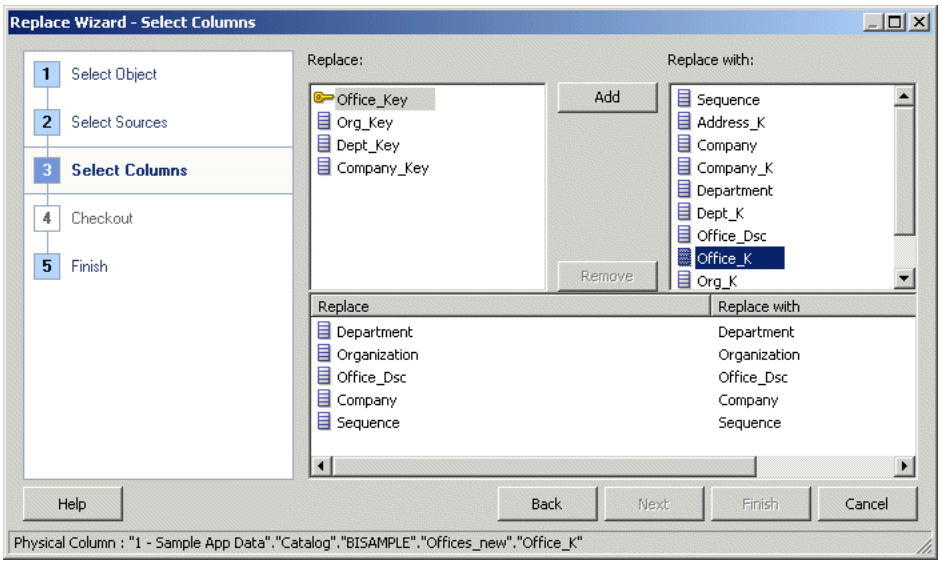
   Click **Next** after you have selected logical table sources.
7. The bottom pane of the Select Columns screen shows individual column mappings between the selected physical tables. If column names in the two selected tables match, default column mappings appear in the bottom pane.

   To add a column mapping to the list of mapped columns, first select a source column in the left pane. Then, select a replacement column in the right pane and click **Add**.

   To remove a column mapping from the list of mapped columns, select a row of mapped columns from the list and click **Remove**.

   Figure 18–4 shows the Select Columns screen.

Figure 18–4 Select Columns Screen of the Replace Column or Table Wizard



8. When you have finished mapping columns between the selected physical tables, click **Next**.
9. When the repository is open in online mode, the Checkout screen appears. In online mode, objects need to be checked out before you can make changes to them. Click **Next** to check out the necessary objects.
10. The Finish screen displays a summary of the objects that will be replaced. If you want to make changes, click **Back**, or select a particular step from the navigation panel.
11. Click **Finish**.

Using the Oracle BI Event Tables Utility

This utility lets you identify a table as an Oracle BI event polling table. An event polling table is a way to notify the Oracle BI Server that one or more physical tables have been updated.

Each row that is added to an event table describes a single update event. The cache system reads rows from, or polls, the event table, extracts the physical table information from the rows, and purges cache entries that reference those physical tables.

For more information about event polling tables, see "Cache Event Processing with an Event Polling Table" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

To start the Oracle BI Event Tables utility:

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Oracle BI Event Tables** and click **Execute**.

Using the Externalize Strings Utility

You can use the Externalize Strings utility to localize the names of Presentation layer subject areas, tables, hierarchies, columns and their descriptions. You can save these text strings to an external file with ANSI, Unicode, and UTF-8 encoding options. You can also choose to save strings to a set of XML files with Unicode encrpytion.

Before you can use the Externalize Strings utility, you must externalize strings in the Presentation layer. Note the following about externalizing strings in the Presentation layer:

- You can right-click any Presentation layer object, such as a subject area, presentation table, or presentation column, and choose **Externalize Display Names > Generate Custom Names** or **Externalize Descriptions > Generate Custom Descriptions** to externalize strings. Note that when you select **Generate Custom Names** and then run the Externalize Strings utility, the translation key will also appear in the Externalize Strings dialog.
- Choosing one of these right-click externalization options automatically selects the **Custom display name** or **Custom description** options in the Properties dialog for the selected object and all of its child objects.

  For example, if you right-click a subject area and choose one of the externalization options, the externalization flag is set on all presentation tables, columns, hierarchies, and levels within that subject area.
- Running the Externalize Strings utility only externalizes those strings that have been selected for externalization in the Presentation layer.

For full information about using the Externalize Strings utility, see "Localizing Metadata Names in the Repository" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

To start the Externalize Strings utility from the Administration Tool:

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Externalize Strings** and click **Execute**.

To run the Externalize Strings command-line utility:

Use the `externalizestrings` command-line utility to localize the names of Presentation layer subject areas, tables, hierarchies, columns and their descriptions.

1. Navigate to the `externalizestrings` utility, which is located here:

   ORACLE_HOME/user_projects/domains/bi/bitools/bin
2. For information about the required syntax, see the information displayed with the `externalizestrings` utility.

Using the Rename Wizard

You can use the Rename Wizard to rename tables and columns in the Presentation layer and Business Model and Mapping layer. It provides a convenient way to transform physical names to user-friendly names.

It is a best practice to rename objects in the Business Model and Mapping layer rather than the Presentation layer, for better maintainability. Giving user-friendly names to logical objects rather than presentation objects ensures that the names can be reused in multiple subject areas. Also, it ensures that the names persist even when you need to delete and re-create subject areas to incorporate changes to your business model.

To use the Rename Wizard:

1. In the Administration Tool, select **Tools**, then select **Utilities**. Then, select **Rename Wizard** and click **Execute**.

   You can also access the Rename Wizard by right-clicking an object or set of objects in the Business Model and Mapping layer or Presentation layer, and then selecting **Rename Wizard**. The wizard starts in the Select Rules screen and only applies to the logical or presentation objects you selected.
2. In the Select Objects screen, select the objects you want to rename. First, select the layer that contains the objects (**Presentation** or **Business Model and Mapping**), then select an object and click **Add**. Click **Add Hierarchy** to add all objects associated with the selected object. Note the following information:

   - You must enable the **Edit presentation names** Administration Tool option before you can select objects from the Presentation layer.
   - You can only select individual presentation columns with the **Use Logical Column Name** property not selected (set to false).

   Click **Next** after you have selected the objects you want to rename.
3. In the Select Types screen, select the object types you want to rename, such as Subject Area, Logical Table, or Logical Column. Note that if you select Presentation Column, then only presentation columns with the **Use Logical Column Name** property not selected (set to false) will be renamed. Then, click **Next**.

4. In the Select Rules screen, select the renaming rules you want to apply and click **Add**. Select **Change specified text** to rename particular words or phrases.

   The renaming rules are applied in the order in which they appear in the list. Select a rule that you have added and click **Up** or **Down** to change the order in which the rules will be applied.

   For example, say you want to rename the logical columns GlobalGROUP, GlobalSales, and GlobalCustomerName to Group, Sales, and Customer Name. To achieve this, you can apply the following rules in the given order:

   ```
   Insert space before each first uppercase letter, unless on the first position
   or there is a space already
   All text lowercase
   First letter of each word capital
   Change each occurance of "Global " to "" (not case sensitive)
   ```

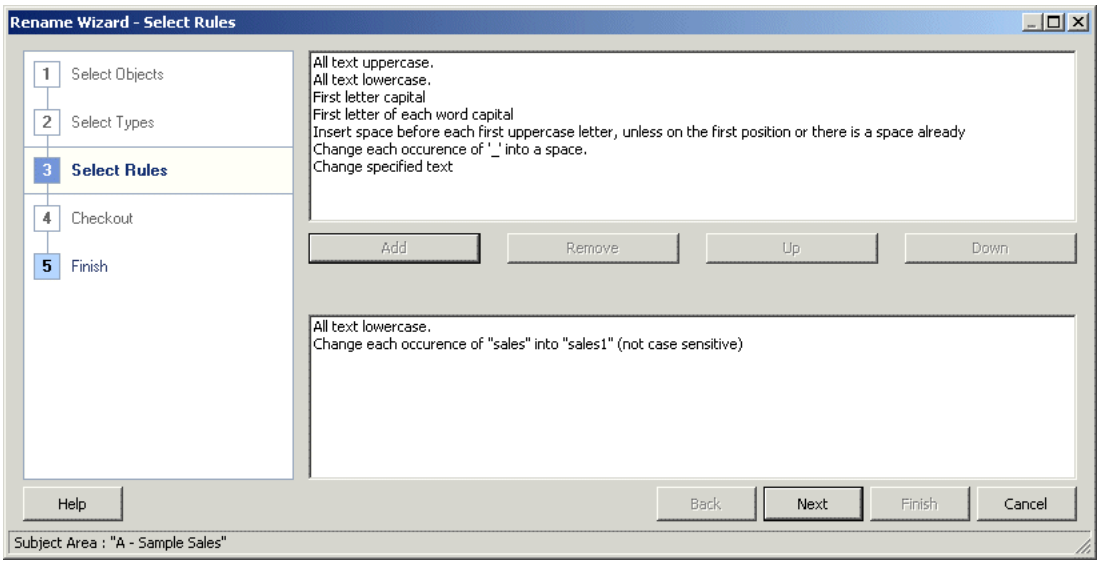
   Click **Next** after you have selected renaming rules.

5. When the repository is open in online mode, the Checkout screen appears. In online mode, objects need to be checked out before you can make changes to them. Click **Next** to check out the necessary objects.

6. The Finish screen displays a summary of the objects that will be renamed. If you want to make changes to the list of renamed objects, click **Back**, or select a particular step from the navigation panel. Click **Finish** to rename the objects.

Figure 18–5 shows the Rename Wizard.

Figure 18–5 Rename Wizard



Using the Update Physical Layer Wizard

You can use the Update Physical Layer Wizard to update database objects in the Physical layer of a repository, based on their current definitions in the back-end database. This wizard is only available for repositories open in online mode.

When the wizard processes the update, the Oracle BI Server connects to each back-end database. The objects in the Physical layer are compared with those in the back-end database. Explanatory text alerts you to differences between objects as defined in the database in the Physical layer and as defined the back-end database, such as data

type-length mismatches and objects that are no longer found in the back-end database. For example, if an object exists in the database in the Physical layer of the repository but not in the back-end database, the following text is displayed:

```
Object does not exist in the database and will be deleted
```

The wizard does not add columns or tables to the repository that exist in the back-end database, but not in the repository. Additionally, the wizard does not update column key assignments. It checks that there is a column in the repository that matches the column in the database, and then, if the values do not match, the wizard updates the type and length of the column in the repository.

The connection pool settings for each database need to match the connection pool settings used when the objects were last imported into the Physical layer from the back-end database. See "Creating or Changing Connection Pools" for more information about connection pool settings.

To update objects in the Physical layer:

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Update Physical Layer** and click **Execute**.

   The databases in the Physical layer of the repository are listed in the left pane of the wizard.
3. In the Select Database screen, select the databases that you want to update in the left pane, and then click **Add**. To remove a database from the update list in the right pane, select it and click **Remove**.
4. Click **Next**.
5. In the Select Connection Pool screen, select the connection pool for each database that you want to update and then click **Next**. You might need to set or confirm values for variables before continuing.
6. In the Update screen, review the information about each update and select the updates you want to perform. You can sort the rows (toggle between ascending and descending order) by clicking the **Name** column heading.
7. If you decide that you do not want the wizard to update a particular object in the Physical layer, click the **Back** button and remove the object.
8. When the repository is open in online mode, the Checkout screen appears. In online mode, objects need to be checked out before you can make changes to them. Click **Next** to check out the necessary objects.
9. Click **Finish**.

   The wizard updates the objects in the Physical layer, and then closes automatically. Objects that do not exist in the database are deleted.
10. From the **File** menu, select **Save** to save the updated objects in the Physical layer.

Generating Documentation of Repository Mappings

The Repository Documentation utility documents the mapping from the presentation columns to the corresponding logical and physical columns. The documentation also includes conditional expressions associated with the columns. The documentation can be saved in comma separated (CSV), XML, or tab delimited (TXT) format.

You can use the Repository Documentation utility to extract Oracle Business Intelligence metadata to a flat file so that it can be loaded into Excel and RDBMS. You

can query the resulting file to answer questions such as "If I delete physical column X, what logical columns will be affected?" or "How many places in the business model refer to the physical table W_SRVREQ_F?" Then, you can establish dependency relationships among elements in the repository.

Excel only allows data sets of 1,000,000 rows. You might exceed this in a large repository. Therefore, you might want to run the Repository Documentation utility on a subset of the repository by extracting relevant business models into a new project. For more information, see Chapter 3.

The Repository Documentation utility creates a comma-separated values file or a tab-separated values file that shows the connections between the Presentation and Physical layers in the current repository. This file can be imported into a repository as a Physical layer. Note that the file does not include information about repository variables and marketing objects.

To run the Repository Documentation utility:

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Repository Documentation** and click **Execute**.
3. In the Save As dialog, choose the directory where you want to save the file.
4. Type a name for the file.
5. Choose a type of file and an Encoding value, and then click **Save**. Current encoding options are ANSI, Unicode, and UTF-8.

Generating a Metadata Dictionary

You can generate a metadata dictionary to help Oracle Business Intelligence users obtain more information about metrics or attributes for repository objects.

For example, users might need to resolve issues caused by confusing metadata object names, or to obtain more details when an attribute is derived in a complicated way.

A metadata dictionary is a static set of XML documents. Each XML document describes a metadata object, such as a column, including its properties and relationships with other metadata objects. These XML documents can be viewed within the Oracle BI Presentation Services user interface, or they can be viewed directly in a browser.

Use the Administration Tool to generate a metadata dictionary for your repository. Because the dictionary does not change dynamically as repository changes are made, you must generate the dictionary periodically to update the content.

The metadata dictionary files need to be hosted on a Web server, such as Oracle HTTP Server or Apache HTTP Server. When you generate the dictionary, you can set the output location to the final location on the Web server, or to a temporary location. If you generate the dictionary in a temporary location, you must then copy the files to the location on the Web server.

Note that some large repositories can contain tens of thousands of objects. Generating a dictionary for a large repository can take a significant period of time.

To generate a metadata dictionary:

1. In the Administration Tool, open your repository in offline mode. You cannot generate a metadata dictionary in online mode.
2. Select **Tools**, then select **Utilities**.
3. Select **Generate Metadata Dictionary** and click **Execute**.

4. In the Choose Directory dialog, click **Browse** to locate and select the location where you want to store the dictionary. You can select a destination for your dictionary in the following ways:

   - Select a local or network location. When the dictionary is generated, a subdirectory with the same name as the repository is created in that location. The dictionary directories and files are created in that subdirectory.

     For example, if you select J:\BI_DataDictionary and your repository name is demo1.rpd, the dictionary files, including the style sheets, will be located in J:\BI_DataDictionary\demo1.

   - If you want to use an IIS virtual directory, you can create or select a virtual directory in IIS before you generate the dictionary. When you generate the dictionary, choose the physical directory associated with the IIS virtual directory.

   Note that you cannot store the dictionary in a directory with multibyte characters. If you receive a system error about creating the necessary directories for the dictionary, then you must choose another directory.

5. Click **OK**.

6. If you did not save the files directly to a location on a Web server, copy the files over to your Web server and ensure they are accessible. Refer to the documentation for your Web server for detailed information.

   The location where the metadata dictionary files can be viewed is dependent on the host name and port number of your Web server, along with the directory location where you store the files.

7. You must edit the instanceconfig.xml configuration file to enable the metadata dictionary feature in the Oracle BI Presentation Services user interface, as well as grant the appropriate privilege to your users, groups, or application roles. See "Providing Access to Metadata Dictionary Information" for more information about these additional configuration steps.

After you generate a metadata dictionary, style sheets and index files are created for that dictionary. The related style sheets (XSL files) are created and stored in a directory named xsl within the repository directory.

A name index and tree index are created and stored in the [drive]:\[path]\[repository name] root directory. The index files are associated with each other so that you can quickly switch views.

For additional information about viewing metadata dictionary information from the Oracle BI Presentation Services user interface, see "Viewing Metadata Information from the Subject Areas Pane" in *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition*.

Providing Access to Metadata Dictionary Information

When creating analyses, content designers might need more information about subject areas, folders, columns, or levels (such as relationships to other metadata objects) to guide them. You can provide content designers with this information by allowing them access to the metadata dictionary for the repository.

The metadata dictionary describes the metrics that are contained within the repository and the attributes of repository objects. The metadata dictionary output is a static set of XML documents.

To provide access to metadata dictionary information:

1. Ensure that the metadata dictionary has been generated and the files have been stored in an appropriate location. For information, see "Generating a Metadata Dictionary."
2. Set the DictionaryURLPrefix element within the ServerInstance element in the instanceconfig.xml file to one of the following values. The value that you specify depends on the web servers in use.
   - The prefix for the name of the directory in which you have stored the XML files. The directory must have been specified as a shared directory for the web server, and the web server must be the same one that is used by Oracle BI EE.

     For example, suppose that you stored the XML files for the metadata dictionary in a directory called demo1 under the metadictionary directory. Suppose that the metadictionary directory is specified as a shared directory for the web server, which is also used by Oracle BI EE. Then you specify the following value for the DictionaryURLPrefix element:

     ```
     <DictionaryURLPrefix>demo1/</DictionaryURLPrefix>
     ```

     See the documentation for the web server for information about sharing directories.
   - The URL that points to the directory in which you have stored the XML files. Use a value such as this when the files for the metadata dictionary are stored in the directory structure for a web server that is not being used by Oracle BI EE. For example:

     ```
     <DictionaryURLPrefix>http://10.10.10.10/metadictionary/demo1</Dicti
     onaryURLPrefix>
     ```

   The following shows an example setting in the instanceconfig.xml file:

   ```
   <WebConfig>
     <ServerInstance>
       <SubjectAreaMetadata>
         <DictionaryURLPrefix>demo1</DictionaryURLPrefix>
       </SubjectAreaMetadata>
     </ServerInstance>
   </WebConfig>
   ```

3. Grant the Access to Metadata Dictionary privilege to the appropriate content designers. For information about privileges, see "Managing Presentation Services Privileges" in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*.

For details on how content designers can view metadata dictionary information, see "Viewing Metadata Information from the Subject Areas Pane" in *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition*.

Removing Unused Physical Objects

Large repositories use more memory on the server and are harder to maintain. Additionally, development activities take longer on a large repository. This utility enables you to remove objects that you no longer need in your repository. You can remove databases, initialization blocks, physical catalogs, and variables.

To remove unused physical objects:

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Remove Unused Physical Objects** and click **Execute**.

3. In the Remove Unused Physical Objects dialog, from the **Type** list, select the type of object.
4. In the list of objects, verify that only the objects that you want to remove are selected.

   Below the list of objects, the number of selected objects and the total number of objects appears.
5. To remove the selected objects, click **Yes**.

Persisting Aggregates

You can use the Aggregate Persistence Wizard to create the SQL file that will be used to create aggregate tables and map them into the metadata. See "Using the Aggregate Persistence Wizard to Generate the Aggregate Specification" for more information.

Using the Oracle BI Summary Advisor Wizard

If running Oracle Business Intelligence on the Oracle Exalytics Machine, you can use the Oracle BI Summary Advisor Wizard to identify which aggregates will increase query performance and to generate a script for creating the recommended aggregates.

See "Using Oracle BI Summary Advisor to Identify Query Candidates for Aggregation" for more information.

The Oracle BI Summary Advisor Wizard is only available in online mode for Oracle Business Intelligence on the Oracle Exalytics Machine.

Using the Convert Presentation Folders Utility

Starting in Release 11.1.1.6.2, you can designate child presentation tables using the Child Presentation Tables tab in the Presentation Table dialog to give the appearance of nested folders in Answers and BI Composer. However, in previous releases, repository developers could achieve one level of nesting in Answers by adding a hyphen at the beginning of a presentation table name, or by adding an arrow (->) at the beginning of a presentation table description. If you used these methods to achieve nesting, it is recommended that you run the Convert Presentation Folders utility to convert your metadata to the new structure.

> **Note:** Achieving nesting by adding hyphens at the beginning of presentation table names or adding arrows at the beginning of presentation table descriptions is deprecated for this release and will be removed in a future release.

To use the Convert Presentation Folders utility:

1. Open your repository in the Administration Tool in offline mode.

   Note: Do not run the Convert Presentation Folders utility in online mode.
2. Select **Tools**, then select **Utilities**.
3. Select **Convert Presentation Folders** and click **Execute**.

   The hyphens and arrows disappear from presentation table names and descriptions, and the affected tables are listed as child tables for the appropriate parent object.

Generating a List of Logical Column Types

You can use the Generate Logical Column Type Document utility to generate a complete list of logical columns and their corresponding types. The output is stored in XML format. You can select ANSI, Unicode, or UTF-8 encoding options.

This utility is often used with the Compare Logical Column Types utility. See "Comparing Logical Column Types" for more information.

To generate a list of logical column types:

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Generate Logical Column Type Document** and click **Execute**.
3. In the Save As dialog, choose the directory where you want to save the file.
4. Type a name for the file. The file must have an XML extension.
5. Choose an **Encoding** value (ANSI, Unicode, or UTF-8) and then click **Save**.

Using the biservergentypexml Utility to Generate a List of Logical Column Types

Similar to the Generate Logical Column Type Document utility in the Administration Tool, you can generate a list of logical columns and their corresponding types using the `biservergentypexml` utility. This utility is available on both Windows and UNIX systems. You can only use `biservergentypexml` with binary repositories in RPD format.

The location of the `biservergentypexml` utility is:

ORACLE_HOME/user_projects/domains/bi/bitools/bin

Syntax

The `biservergentypexml` utility takes the following parameters:

```
biservergentypexml -R repository_name [-P repository_password]
-O output_XML_file_name {-8 | -U | -A}
```

Where:

repository_name is the name and path of the repository from which you want to generate a list of logical column types.

repository_password is the password for the repository from which you want to generate a list of logical column types.

Note that the *repository_password* argument is optional. If you do not provide the password argument, you are prompted to enter the password when you run the command. To minimize the risk of security breaches, Oracle recommends that you do not provide password arguments either on the command line or in scripts. Note that the password argument is supported for backward compatibility only, and will be removed in a future release. For scripting purposes, you can pass the password through standard input.

output_XML_file_name is the name and path of the XML file where you want to store the output generated by the utility.

- 8 specifies UTF-8 encoding for the output file.

- U specifies Unicode encoding for the output file.

- A specifies ANSI encoding for the output file.

Example

The following example creates a UTF-8 encoded output XML file called log_col_types.xml that includes logical column type information from my_repos.rpd.

```
biservergentypexml -R my_repos.rpd -O log_col_types.xml -8
Give password: my_rpd_password
```

> **Note:** Be sure to provide the full pathnames to your repository file and XML output file if they are located in a different directory.

Sample Output for a Logical Column Types Document

Sample output for a logical column types document, generated either with the Generate Logical Column Type Document utility in the Administration Tool or with the `biservergentypexml` utility, appears as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
 <REPOSITORY>
  <BUSINESS_MODEL NAME="SampleApp Lite">
   <LOGICAL_TABLE NAME="D1 Products">
    <COLUMN NAME="Product Number">
     <TYPE>INT</TYPE>
     <NULLABLE>No</NULLABLE>
    </COLUMN>
    <COLUMN NAME="Product">
     <TYPE>VARCHAR</TYPE>
     <LENGTH>25</LENGTH>
     <NULLABLE>No</NULLABLE>
    </COLUMN>
    <COLUMN NAME="Product Type">
     <TYPE>VARCHAR</TYPE>
     <LENGTH>25</LENGTH>
     <NULLABLE>No</NULLABLE>
    </COLUMN>
    <COLUMN NAME="Product Type Key">
     <TYPE>INT</TYPE>
     <NULLABLE>No</NULLABLE>
    </COLUMN>
...
   </LOGICAL_TABLE>
   <LOGICAL_TABLE NAME="D0 Time">
    <COLUMN NAME="Calendar Date">
     <TYPE>DATE</TYPE>
     <NULLABLE>No</NULLABLE>
    </COLUMN>
    <COLUMN NAME="Per Name Week">
     <TYPE>VARCHAR</TYPE>
     <LENGTH>12</LENGTH>
     <NULLABLE>No</NULLABLE>
    </COLUMN>
...
   </LOGICAL_TABLE>
  </BUSINESS_MODEL>
 </REPOSITORY>
```

Comparing Logical Column Types

Sometimes, logical column types can change over the course of MUD development, resulting in unexpected logical column types. When this occurs, you can generate a list

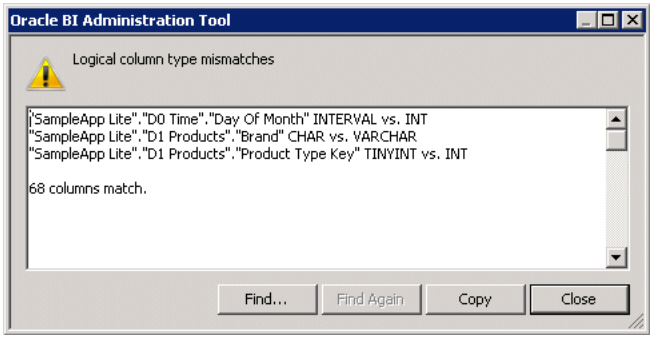
of logical columns and their types using the Generate Logical Column Type Document utility in the Administration Tool or `biservergentypexml`, and then use the Compare Logical Column Types utility for subsequent MUD versions to ensure that the logical column types match as expected. For example, you could generate a logical column type list for repository version 20, and then use the Compare Logical Column Types utility to compare the list against repository version 30.

To use this utility, you must have already generated a list of logical column types with which you want to compare the current repository. Note that the Compare Logical Column Types utility only compares logical columns that exist in both the repository and the XML file - newly created logical columns and deleted columns are ignored.

To compare logical column types:

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Compare Logical Column Types** and click **Execute**.
3. In the Select XML File dialog, select the generated list of logical column types with which you want to compare the column types in the current repository.
4. Click **Open**. A list of logical column type mismatches appears, as shown in the following image:

Figure 18–6 List of Logical Column Type Mismatches



In the dialog showing the logical column type mismatches, you can perform the following actions:

- Use **Find** to find a particular text string
- Use **Find Again** to find subsequent text strings
- Use **Copy** to copy all text shown in the dialog to the clipboard

Fixing Upgrade IDs

If cases where you are comparing or merging repositories, the upgrade IDs sometimes do not function correctly. You can use the Fix Upgrade IDs utility to correct issues with upgrade IDs.

Oracle BI uses upgrade IDs to compare or merge repositories. They identify when two object in two repositories are supposed to be the same object. However, in some cases, the upgrade IDs do not work correctly. For example, when two or more objects have the same upgrade ID, when objects are missing upgrade IDs, and when hidden internal object have upgrade IDs set.

To fix upgrade IDs:

1. In the Administration Tool, select **Tools**, then select **Utilities**.

2. Select **Fix Upgrade IDs** and click **Execute**.

   If you ran the utility on a repository open in read-only mode, then the utility reports how many invalid upgrade IDs are in the repository. To fix the upgrade IDs, you must open the repository in non-ready-only mode and rerun the utility.

   If you ran the utility on a repository that is not read-only, then the utility fixes the invalid upgrade IDs and displays a message stating how many upgrade IDs it fixed.

Setting Permissions In Bulk

You can use the Set Permissions in Bulk utility when you want to assign the same object, data filters, and query limits permissions to several users or roles at the same time.

To set permissions in bulk:

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Set Permissions in Bulk** and click **Execute**.
3. In the Set Permissions in Bulk dialog, specify the items for which you want to set permissions (Object Permissions, Data Filters, and Query Limits).
4. Select the users and roles for which you want to bulk assign permissions and click **Add** to move them to the selected table.
5. Click **OK**. In the dialog, select the tab corresponding to the item to which you want to bulk assign permissions (for example, Data Filters,) and specify the permissions to assign to the users and roles you chose.

   For information about completing the Object Permissions tab, see "Setting Up Object Permissions."

   For information about completing the Data Filters tab, see "Setting Up Row-Level Security (Data Filters) in the Repository."

   For information about completing the Query Limits tab, see "Setting Query Limits."

6. After you have specified the permissions, click **OK**.

Using the Calculation Wizard

You can use the Calculation Wizard to create new calculation columns that compare two existing columns, and also to create metrics in bulk. It has a built-in mechanism to handle divide-by-zero and null cases, as well as other difficult situations. The Calculation Wizard provides an automated way to calculate the sales by quarter, the percentage of revenue, minimum and maximum values, and so on.

To start the Calculation Wizard, right-click any logical fact or dimension column in the Business Model and Mapping layer of data type numeric, and then select the option **Calculation Wizard**. The wizard starts with the column on which you right-clicked as the source column, and then displays the other columns in that table for comparison.

To use the Calculation Wizard:

1. Right-click a measure column in the Business Model and Mapping layer (any logical fact or dimension column of data type numeric), and then select **Calculation Wizard**.

2. The first time you use the Calculation Wizard, the Introduction screen appears. Select **In the future, do not show this introduction screen** if you do not want this screen to display subsequently. If you choose not to display the Introduction screen, you can go to **Tools > Options** to cause it to appear again. See "Setting Administration Tool Options" for more information.

   Click **Next** to display the Select Columns screen.

3. Select the columns that you want to compare with the source column. If the source column is mapped to multiple logical tables, a list of tables appears in the upper left pane. Select a table, then select a column or columns from the upper right pane to add comparison columns to the Selected Columns list.

   You can remove items from the Selected Columns list by selecting a column and clicking **Remove**.

   Click **Next** when you have finished selecting comparison columns.

4. In the New Calculations screen, you can choose which calculations you want to perform for specific columns. The elements of the New Calculation screen are as follows:

   - The upper left pane shows the name of the source column, followed by a list of comparison columns that you selected in the Select Columns screen. Select a particular column to create calculations for that column.
   - The upper right pane shows a list of calculations you can perform. Select a calculation to view the calculation definition and the default calculation name. In the calculation definition, CurrentX refers to the value of the source column, and ComparisonX refers to the value of the comparison column you selected in the upper left pane.

     You can optionally change the calculation name. This name becomes the name of the resulting calculation column.

     The following calculations are available:

     – **Change (CurrentX - ComparisonX).** Subtract the value of the comparison column from the source column.

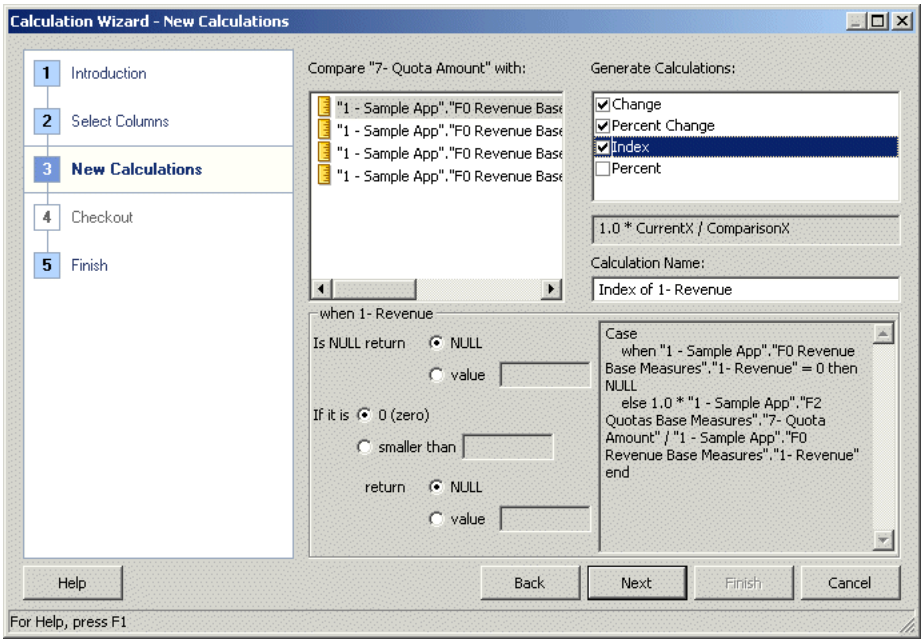
     – **Percent Change (100.0 * (CurrentX - ComparisonX) / ComparisonX).** Subtract the value of the comparison column from the source column and express as a percentage.

     – **Index (1.0 * CurrentX / ComparisonX).** Divide the source column by the comparison column.

     – **Percent (100.0 * (CurrentX / ComparisonX)).** Divide the source column by the comparison column and express as a percentage.
   - The lower left pane shows special cases that are available for the selected calculation. You can keep the default values, or specify how you want the special cases to be handled. For example, for the Change calculation, you can choose whether to return `NULL` or some other value when the comparison column is `NULL`.

     Select a calculation in the upper right pane to view and set special cases for that calculation.
   - The lower right pane shows the resulting SQL for the selected calculation.

Figure 18–7 New Calculations Screen of Calculation Wizard



5. Click **Next** when you have finished creating calculations.
6. When the repository is open in online mode, the Checkout screen appears. In online mode, objects need to be checked out before you can make changes to them. Click **Next** to check out the necessary objects.
7. The Finish screen displays a summary of the calculations that will be created. If you want to make changes, click **Back**, or select a particular step from the navigation panel.
8. Click **Finish**. The new calculation columns are created.

19

Using Variables in the Oracle BI Repository

This chapter describes how to use variables in the Oracle BI repository to streamline administrative tasks and dynamically modify metadata content to adjust to a changing data environment. There are two classes of variables: repository variables and session variables.

- A repository variable has a single value at any point in time. There are two types of repository variables: static and dynamic.
- Session variables are created and assigned a value when each user logs on. There are two types of session variables: system and nonsystem.

Initialization blocks are used to initialize dynamic repository variables, system session variables, and nonsystem session variables.

You can use the Variable Manager in the Administration Tool to define variables. The Variable Manager dialog has two panes. The left pane displays a tree that shows variables and initialization blocks, and the right pane displays details of the item you select in the left pane. Repository variables and system and nonsystem session variables are represented by a question mark icon. The icon for an initialization block is a cube labeled with a question mark and a clock.

> **Caution:** Values in repository and session variables are not secure, because object permissions do not apply to variables. Anybody who knows or can guess the name of the variable can use it in an expression in Answers or in a Logical SQL query. Because of this, do not put sensitive data like passwords in session or repository variables.

This chapter contains the following topics:

Working with Repository Variables

This section provides information about working with repository variables, and contains the following topics:

About Repository Variables

A repository variable has a single value at any point in time. Repository variables can be used instead of literals or constants in Expression Builder in the Administration Tool. The Oracle BI Server substitutes the value of the repository variable for the variable itself in the metadata.

This section contains the following topics:

About Static Repository Variables

The value of a static repository variable is initialized in the Variable dialog. This value persists, and does not change until an administrator decides to change it.

For example, suppose you want to create an expression to group times of day into different day segments. If Prime Time were one of those segments and corresponded to the hours between 5:00 PM and 10:00 PM, you could create a `CASE` statement like the following:

```
CASE WHEN "Hour" >= 17 AND "Hour" < 23 THEN 'Prime Time' WHEN... ELSE...END
```

where Hour is a logical column, perhaps mapped to a timestamp physical column using the date-and-time `Hour(<<timeExpr>>)` function.

Rather than entering the numbers 17 and 23 into this expression as constants, you could use the Variable tab of the Variable dialog to set up a static repository variable named `prime_begin` and initialize it to a value of `17`, and create another variable named `prime_end` and initialize it to a value of `23`.

Static repository variables must have default initializers that are either numeric or character values. In addition, you can use Expression Builder to insert a constant as the default initializer, such as Date, Time, and TimeStamp. You cannot use any other value or expression as the default initializer for a static repository variable.

In previous releases, the Administration Tool did not limit the values of default initializers for static repository variables. Because of this, if your repository has been upgraded from a previous release, you may see warnings in the Consistency Checker similar to the following:

```
The variable, 'Current Month' does not have a constant default initializer.
```

If you see warnings similar to this, update the relevant static repository variables so that the default initializers have constant values.

About Dynamic Repository Variables

You initialize dynamic repository variables in the same way as static variables, but the values are refreshed by data returned from queries. When defining a dynamic repository variable, you create an initialization block or use a preexisting one that

contains a SQL query. You also set up a schedule that the Oracle BI Server will follow to execute the query and periodically refresh the value of the variable.

When the value of a dynamic repository variable changes, all cache entries associated with a business model that reference the value of that variable are purged automatically.

Each query can refresh several variables: one variable for each column in the query. You schedule these queries to be executed by the Oracle BI Server.

Dynamic repository variables are useful for defining the content of logical table sources. For example, suppose you have two sources for information about orders. One source contains recent orders and the other source contains historical data.

You need to describe the content of these sources on the Content tab of the Logical Table Source dialog. Without using dynamic repository variables, you would describe the content of the source containing recent data with an expression such as:

```
Orders.OrderDates."Order Date" >= TIMESTAMP '2001-06-02 00:00:00'
```

This content statement becomes invalid as new data is added to the recent source and older data is moved to the historical source. To accurately reflect the new content of the recent source, you would have to modify the fragmentation content description manually. Dynamic repository values can be set up to do it automatically.

Another suggested use for dynamic repository values is in `WHERE` clause filters of logical table sources, defined on the Content tab of the Logical Table Source dialog.

A common use of these variables is to set filters for use in Oracle BI Presentation Services. For example, to filter a column on the value of the dynamic repository variable CurrentMonth, set the filter to the variable CurrentMonth.

Creating Repository Variables

This section explains how to create repository variables.

To create a repository variable:

1. In the Administration Tool, select **Manage**, then select **Variables**.
2. In the Variable Manager dialog, select **Action > New > Repository > Variable**.
3. In the Variable dialog, type a name for the variable.

   Names for all variables should be unique. The names of system session variables are reserved and cannot be used for other types of variables.
4. Select the type of variable: **Static** or **Dynamic**.
5. If you selected **Dynamic**, use the **Initialization Block** list to select an existing initialization block that will be used to refresh the value on a continuing basis. If you are creating a repository variable to be used to override a hierarchy column's selection steps, then you must choose an initialization block with its initialization string written in JSON syntax. See "Initialization Strings Used in Variables to Override Selection Steps" for more information.

   To create a new initialization block, click **New**. See "Creating Initialization Blocks" for more information.
6. To add a **Default initializer** value, type the value in the **Default initializer** box, or click the **Expression Builder** button to use Expression Builder.

   For static repository variables, the value you specify in the **Default initializer** window persists. It will not change unless you change it. If you initialize a variable

using a character string, enclose the string in single quotes ('). Static repository variables must have default initializers that are constant values.

7. Click **OK**.

Using Repository Variables in Expression Builder

After they are created, variables are available for use in Expression Builder. In Expression Builder, click the **Repository Variables** folder in the left pane to display all repository variables (both static and dynamic) in the middle pane by name.

To use a repository variable in an expression, select it and double-click. Expression Builder pastes it into the expression at the active cursor insertion point.

Variables should be used as arguments of the function `VALUEOF()`. This happens automatically when you double-click the variables to paste them into the expression.

For example, the following `CASE` statement is identical to the one explained in the preceding example, except that variables have been substituted for the constants:

```
CASE WHEN "Hour" >= VALUEOF("prime_begin")AND "Hour" < VALUEOF("prime_end") THEN
'Prime Time' WHEN ... ELSE...END
```

Note: You cannot use variables to represent columns or other repository objects.

Working with Session Variables

This section provides information about working with session variables, and contains the following topics:

- About Session Variables
- Creating Session Variables

About Session Variables

Session variables obtain their values from initialization blocks. Unlike dynamic repository variables, however, the initialization of session variables is not scheduled. When a user begins a session, the Oracle BI Server creates new instances of session variables and initializes them.

Unlike a repository variable, there are as many instances of a session variable as there are active sessions on the Oracle BI Server. Each instance of a session variable could be initialized to a different value.

Session variables are primarily used when authenticating users against external sources such as database tables or LDAP servers. If a user is authenticated successfully, session variables can be used to set filters and permissions for that session. For information about using session variables when setting up security, see "Managing Session Variables" in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*.

This section contains the following topics:

- About System Session Variables
- About Nonsystem Session Variables

About System Session Variables

System session variables are session variables that the Oracle BI Server and Oracle BI Presentation Services use for specific purposes. System session variables have reserved names that cannot be used for other kinds of variables (such as static or dynamic repository variables and nonsystem session variables).

When you use these variables for Oracle BI Presentation Services, preface their names with `NQ_SESSION`. For example, to filter a column on the value of the variable `LOGLEVEL`, set the filter to the variable `NQ_SESSION.LOGLEVEL`.

Table 19–1 describes the available system session variables.

Table 19–1 System Session Variables

Variable	Description
USER	Holds the value the user enters as his or her logon name. This variable is typically populated from the LDAP profile of the user.
USERGUID	Contains the global unique identifier (GUID) of the user, typically populated from the LDAP profile of the user.
GROUP	Contains the groups to which the user belongs. Exists only for compatibility with previous releases. Legacy groups are mapped to application roles automatically. When a user belongs to multiple groups, include the group names in the same column, separated by semicolons (for example, GroupA;GroupB;GroupC). If a semicolon must be included as part of a group name, precede the semicolon with a backslash character (\).
ROLES	Contains the application roles to which the user belongs. When a user belongs to multiple roles, include the role names in the same column, separated by semicolons (for example, RoleA;RoleB;RoleC). If a semicolon must be included as part of a role name, precede the semicolon with a backslash character (\).
ROLEGUIDS	Contains the global unique identifiers (GUIDs) for the application roles to which the user belongs. GUIDs for application roles are the same as the application role names.
PERMISSIONS	Contains the permissions held by the user, such as oracle.bi.server.manageRepositories.
PROXY	Holds the name of the proxy user. A proxy user is a user that has been authorized to act for another user. See *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition* for more information about the PROXY system session variable.
WEBGROUPS	Specifies the Catalog groups (Presentation Services groups) to which the user belongs, if any. Note that the recommended practice is to use application roles rather than Catalog groups. When a user belongs to multiple Catalog groups, include the Catalog group names in the same column, separated by semicolons (for example, WebgroupA;WebgroupB;WebgroupC). If a semicolon must be included as part of a Catalog group name, precede the semicolon with a backslash character (\).
DISPLAYNAME	Used for Oracle BI Presentation Services. It contains the name that is displayed to the user in the greeting in the Oracle BI Presentation Services user interface. It is also saved as the author field for catalog objects. This variable is typically populated from the LDAP profile of the user.

Table 19–1 (Cont.) System Session Variables

Variable	Description
LOGLEVEL	The value of LOGLEVEL (a number between 0 and 5) determines the logging level that the Oracle BI Server uses for user queries. This system session variable overrides a variable defined in the Users object in the Administration Tool. If the administrator user (defined upon install) has a Logging level defined as 4 and the session variable LOGLEVEL defined in the repository has a value of 0 (zero), the value of 0 applies.
DESCRIPTION	Contains a description of the user, typically populated from the LDAP profile of the user.
USERLOCALE	Contains the locale of the user, typically populated from the LDAP profile of the user.
DISABLE_CACHE_HIT	Used to enable or disable Oracle BI Server result cache hits. This variable has a possible value of 0 or 1.
DISABLE_CACHE_SEED	Used to enable or disable Oracle BI Server result cache seeding. This variable has a possible value of 0 or 1.
DISABLE_SUBREQUEST_CACHE	Used to enable or disable Oracle BI Server subrequest cache hits and seeding. This variable has a possible value of 0 or 1.
SELECT_PHYSICAL	Identifies the query as a SELECT_PHYSICAL query. See "Syntax and Usage Notes for SELECT_PHYSICAL" for more information.
DISABLE_PLAN_CACHE_HIT	Used to enable or disable Oracle BI Server plan cache hits. This variable has a possible value of 0 or 1.
DISABLE_PLAN_CACHE_SEED	Used to enable or disable Oracle BI Server plan cache seeding. This variable has a possible value of 0 or 1.
TIMEZONE	Contains the time zone of the user, typically populated from the LDAP profile of the user.
WEBLANGUAGE	Used for Oracle BI Presentation Services. Holds the Oracle BI Presentation Services user interface display language. Users can select a language on the sign-in page for Oracle BI EE, or they can change the language setting on the Preferences tab of the My Account dialog after signing in.
AUTHINITBLOCKONLY	Determines if the initialization blocks required for authentication are executed. This variable has a value of Yes. The value is case-insensitive.
PORTALPATH	Used for Oracle BI Presentation Services. It identifies the default dashboard the user sees when logging in (the user can override this preference after logged on).
REQUESTKEY	Used for Oracle BI Presentation Services. Any users with the same nonblank request key share the same Oracle BI Presentation Services cache entries. This tells Oracle BI Presentation Services that these users have identical content filters and security in the Oracle BI Server. Sharing Oracle BI Presentation Services cache entries is a way to minimize unnecessary communication with the Oracle BI Server.
SKIN	Determines certain elements of the look and feel of the Oracle BI Presentation Services user interface. The user can alter some elements of the user interface by picking a style when logged on to Oracle BI Presentation Services. The SKIN variable points to an Oracle BI Presentation Services folder that contains the nonalterable elements (for example, figures such as GIF files). Such directories begin with sk_ . For example, if a folder were called sk_companyx, the SKIN variable would be set to companyx.

About Nonsystem Session Variables

You use the same procedure to define nonsystem session variables as for system session variables.

A common use for nonsystem session variables is setting user filters. For example, you could define a nonsystem variable called SalesRegion that would be initialized to the name of the sales region of the user.

You could then set a security filter for all members of a group that would allow them to view only data pertinent to their region.

When you use these variables for Oracle BI Presentation Services, preface their names with `NQ_SESSION`. For example, to filter a column on the value of the variable `SalesRegion`, set the filter to the variable `NQ_SESSION.SalesRegion`.

Creating Session Variables

This section explains how to create session variables.

To create a session variable:

1. In the Administration Tool, select **Manage**, then select **Variables**.
2. In the Variable Manager dialog, select **Action > New > Session > Variable**.
3. In the Session Variable dialog, type a variable name.

   Names for all variables should be unique. The names of system session variables are reserved and cannot be used for other types of variables.
4. For session variables, you can select the following options:
   - **Enable any user to set the value.** Select this option to set session variables after the initialization block has populated the value (at user login) by calling the ODBC stored procedure `NQSSetSessionValue()`. For example, this option lets non-administrators to set this variable for sampling.

     Note that the `NQSSetSessionValues()` stored procedure is not supported for use through the Issue SQL page in Oracle BI Presentation Services Administration.

     If this option is not selected, then the variable cannot be set.
   - **Security Sensitive.** Select this option to identify the variable as sensitive to security when using a row-level database security strategy, such as a Virtual Private Database (VPD). When filtering cache table matches, the Oracle BI Server looks at the parent database object of each column or table that is referenced in the logical request projection list. If the database object has the **Virtual Private Database** option selected, the Oracle BI Server matches a list of security-sensitive variables to each prospective cache hit. Cache hits would only occur on cache entries that included and matched all security-sensitive variables.
5. Use the **Initialization Block** list to select an initialization block that will be used to refresh the value on a continuing basis. If you are creating a session variable to be used to override a hierarchy column's selection steps, then you must choose an initialization block with its initialization string written in JSON syntax. See "Initialization Strings Used in Variables to Override Selection Steps" for more information.

   To create a new initialization block, click **New**. See "Creating Initialization Blocks" for more information.

6. To add a **Default Initializer** value, type the value in the **Default Initializer** box, or click the **Expression Builder** button to use Expression Builder.
7. Click **OK**.

Working with Initialization Blocks

Initialization blocks are used to initialize dynamic repository variables, system session variables, and nonsystem session variables. For example, the `NQ_SYSTEM` initialization block is used to refresh system session variables.

This section contains the following topics:

About Using Initialization Blocks with Variables

An initialization block contains the SQL statement that will be executed to initialize or refresh the variables associated with that block.

The SQL statement must reference physical tables that can be accessed using the connection pool specified in the **Connection Pool** field in the Initialization Block dialog.

If you want the query for an initialization block to have database-specific SQL, you can select a database type for that query. If a SQL initialization string for that database type has been defined when the initialization block is instantiated, this string is used. Otherwise, a default initialization SQL string is used.

Caution: By default, when you open the Initialization Block dialog for editing in online mode, the initialization block object is automatically checked out. While the initialization block is checked out, the Oracle BI Server may continue to refresh the value of dynamic variables refreshed by this initialization block, depending on the refresh intervals that are set. When you check in the initialization block, the value of the dynamic variables is reset to the values shown in the Default initializer. If you do not want this to occur, use the **Undo Check Out** option.

This section contains the following topics:

Initializing Dynamic Repository Variables

The values of dynamic repository variables are set by queries defined in the **Default initialization string** field of the Initialization Block dialog. You also set up a schedule

that the Oracle BI Server will follow to execute the query and periodically refresh the value of the variable. If you stop and restart the Oracle BI Server, the server automatically executes the SQL statements in repository variable initialization blocks, reinitializing the repository variables.

The Oracle BI Server logs all SQL queries issued to retrieve repository variable information in nqquery.log when the logging level for the administrator account (set upon installation) is set to 2 or higher. You should set the logging level to 2 for the administrator to provide the most useful level of information. You can find the nqquery.log file in:

BIDOMAIN\servers\obis*n*\logs

For more information about user-level logging, see "Managing the Query Log" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

Initializing Session Variables

As with dynamic repository variables, session variables obtain their values from initialization blocks. Unlike dynamic repository variables, session variables are not updated at scheduled time intervals. Instead, the Oracle BI Server creates new instances of those variables whenever a user begins a new session. The values remain unchanged for the duration of the session.

Execution of session variable initialization blocks during session logon can be deferred until their associated session variables are actually accessed within the session. See "Creating Initialization Blocks" for more information.

The Oracle BI Server logs all SQL queries issued to retrieve session variable information if the logging level is set to 2 or higher in the Identity Manager User object, or the LOGLEVEL system session variable is set to 2 or higher in the Variable Manager.

The default location for the nqquery.log file is:

BIDOMAIN\servers\obis*n*\logs

For more information about user-level logging, see "Managing the Query Log" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

About Row-Wise Initialization

You can use the row-wise initialization option to create session variables dynamically and set their values when a session begins. The names and values of the session variables reside in an external database that you access through a connection pool. The variables receive their values from the initialization string that you type in the Initialization Block dialog.

For example, suppose you want to create session variables using values contained in a table named RW_SESSION_VARS. The table contains three columns:

- USERID, containing values that represent the unique identifiers of the users
- NAME, containing values that represent session variable names
- VALUE, containing values that represent session variable values

Table 19–2 shows the table in this example.

Table 19–2 Sample Session Variables Database Table

USERID	NAME	VALUE
JOHN	LEVEL	4
JOHN	STATUS	FULL-TIME
JANE	LEVEL	8
JANE	STATUS	FULL-TIME
JANE	GRADE	AAA

To use row-wise initialization, create an initialization block and select the **Row-wise initialization** option (refer to "Creating Initialization Blocks"). For this example, you would provide the following SQL statement for the initialization string:

```
SELECT NAME, VALUE
FROM RW_SESSION_VARS
WHERE USERID='VALUEOF(NQ_SESSION.USERID)'
```

Note that `NQ_SESSION.USERID` has already been initialized using another initialization block.

The following session variables would be created:

- When John connects to the Oracle BI Server, his session contains two session variables from row-wise initialization: `LEVEL`, containing the value `4`, and `STATUS`, containing the value `FULL_TIME`.
- When Jane connects to the Oracle BI Server, her session contains three session variables from row-wise initialization: `LEVEL`, containing the value `8`; `STATUS`, containing the value `FULL-TIME`; and `GRADE`, containing the value `AAA`.

Initializing a Variable with a List of Values You can use the row-wise initialization option to initialize a variable with a list of values. You can then use the SQL `IN` operator to test for values in a specified list.

For example, using the table values in the previous example, you would type the following SQL statement for the initialization string:

```
SELECT 'LIST_OF_USERS', USERID
FROM RW_SESSION_VARS
WHERE NAME='STATUS' AND VALUE='FULL-TIME'
```

This SQL statement populates the variable `LIST_OF_USERS` with a list, separated by colons, of the values `JOHN` and `JANE` (for example, `JOHN:JANE`). You can then use this variable in a filter, as shown in the following `WHERE` clause:

```
WHERE TABLE.USER_NAME = valueof(NQ_SESSION.LIST_OF_USERS)
```

The variable `LIST_OF_USERS` contains a list of values, that is, one or more values. This logical `WHERE` clause expands into a physical `IN` clause, as shown in the following statement:

```
WHERE TABLE.USER_NAME IN ('JOHN', 'JANE')
```

Note that the above information and example pertain to Logical SQL. If you are using Physical SQL to initialize a variable with a list of values, then you must use the `VALUELISTOF` function. For example, to get the customers assigned to the user names in the variable `LIST_OF_USERS`, use the following statement:

```
Select 'LIST_OF_CUSTOMERS', Customer_Name from RW_CUSTOMERS where
```

```
RW.CUSTOMERS.USER_NAME in (VALUELISTOF(NQ_SESSION.LIST_OF_USERS))
```

To filter by only specific values in the list, then use `ValueNameof` as show in the below example. Note that the first value is 0, not 1.

```
Select 'LIST_OF_CUSTOMERS', Customer_Name from RW_CUSTOMERS where
RW.CUSTOMERS.USER_NAME in '(ValueNameOf(0,NQ_SESSION.LIST_OF_USERS))
```

Creating Initialization Blocks

See "About Using Initialization Blocks with Variables" for more information about initialization blocks.

To create initialization blocks, perform the steps in the following sections:

- Assigning a Name and Schedule to Initialization Blocks
- Selecting and Testing the Data Source and Connection Pool

Assigning a Name and Schedule to Initialization Blocks

For repository variables, you can specify the day, date, and time for the start date, as well as a refresh interval.

To assign a name and schedule to initialization blocks:

1. In the Administration Tool, select **Manage**, then select **Variables**.
2. In the Variable Manager dialog, from the **Action** menu, choose **New > Repository** (or **Session**) **> Initialization Block**.
3. In the [Repository | Session] Variable Initialization Block dialog, type a name for the block. (The `NQ_SYSTEM` initialization block name is reserved.)
4. (Repository initialization blocks only) In the **Schedule** area, select a start date and time and the refresh interval.
5. (Session init blocks only) Select the following options when appropriate:
   - **Disabled**. If you select this option, the initialization block is disabled.

     You can also right-click an existing initialization block in the Variable Manager and choose **Disable** or **Enable**. This option enables you to change this property without opening the initialization block dialog.
   - **Allow deferred execution**. If you select this option, execution of the initialization block is deferred until an associated session variable is accessed for the first time during the session.

     This option prevents execution of all session variable initialization blocks during the session logon stage, giving a shorter logon time. Session variables that are not needed during the session do not have their initialization blocks executed. This saves the resources which would have been used to execute these unnecessary initialization blocks.

     The deferred execution of an initialization block also triggers the execution of all unexecuted predecessor initialization blocks. All associated variables of the initialization block and its unexecuted predecessors are updated with the values returned from the deferred execution.

     Note: The **Allow deferred execution** option is unavailable in some circumstances. See "When Execution of Session Variable Initialization Blocks Cannot Be Deferred" for more information.

– **Required for authentication**. If you select this option, this initialization block must succeed for users to log in. In other words, users are denied access to Oracle Business Intelligence if the initialization block fails to execute. Failure to execute can occur if the wrong credentials have been defined in the initialization block, or if there is an error in the default initialization string.

  Note that this requirement is waived for internal processes (like Delivers) that use impersonation, if a single user session variable has been associated with the initialization block. In this case, the trusted internal process can connect regardless of whether the initialization block succeeds or fails.

The next step is to select the data source and connection pool.

Selecting and Testing the Data Source and Connection Pool

If you select **Database** as the data source type for an initialization block, the values returned by the database for the columns in your SQL statement are assigned to variables that you associate with the initialization block. For session variable initialization blocks, you can also select **LDAP Server** or **Custom Authenticator**.

It is recommended that you create a dedicated connection pool for initialization blocks where you select **Database** as the data source type. In addition, if an initialization block fails for a particular connection pool during Oracle BI Server start-up, no more initialization blocks using that connection pool are processed. Instead, the connection pool is blacklisted and subsequent initialization blocks for that connection pool are skipped. See "About Connection Pools for Initialization Blocks" for more information.

If you select **Database** as the data source type:

- If you select **Database** as the data source type, and do not select the **Use OBI EE Server** option

  The SQL statement used to refresh the variable must reference physical tables that can be accessed through the connection pool specified in the **Connection Pool** field. The tables do not have to be included in the Physical layer of the metadata. At run time, if an initialization string for the database type has been defined, this string is used. Otherwise, the default initialization SQL for the database type is used. You can overtype this string.

  When you create SQL and submit it directly to the database (for example, when using database-specific SQL in initialization blocks), the SQL statement bypasses the Oracle BI Server. The order of the columns in the SQL statement and the order of the variables associated with the initialization block determine which columns are assigned to each variable.

  You should test this SQL using the **Test** button in the [Repository | Session] Variable Initialization Block Data Source dialog. If the SQL statement contains an error, the database returns an error message. See "Testing Initialization Blocks" for more information.

- If you select **Database** as the data source type, and select the **Use OBI EE Server** option

  The SQL statement you use to refresh the variable might be written for a specific database. However, it will still work with other data sources because the SQL statement is processed by the Oracle BI Server. The Oracle BI Server can also provide functions (such as `PI`) that might not be available in the data source, and the SQL statement will work with other data sources supported by the Oracle BI Server (for example, ADF, SQL Server, Oracle, and XML files). When you select the **Use OBI EE Server** option, there is no need for a connection pool, because the SQL

statement is sent to the Oracle BI Server and not directly to the underlying database.

You can only test this SQL statement using the **Test** button in the [Repository | Session] Variable Initialization Block Data Source dialog when in online mode. If the SQL statement contains an error, the database returns an error message. See "Testing Initialization Blocks" for more information.

To select a data source and connection pool for initialization blocks:

1. In the Administration Tool, select **Manage**, then select **Variables**.
2. In the Variable Manager dialog, double-click the initialization block you want to edit. You can edit Repository initialization blocks, or Session initialization blocks.
3. Click **Edit Data Source** next to the **Connection Pool** field.
4. From the **Data Source Type** list, select one of the following types.
   - **Database:** For repository and session variables.
   - **LDAP Server:** For session variables.
   - **Custom Authenticator:** For session variables. See *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition* for more information.
5. If you selected **Database** for your data source type, perform one of the following steps:
   - Select **Default initialization string** or **Use database specific SQL**, and then perform the following steps:
     a. Click **Browse** next to the **Connection Pool** field to select the connection pool associated with the database where the target information is located. If you do not select a connection pool before typing the initialization string, you receive a message prompting you to select the connection pool.
     b. In the Select Connection Pool dialog, select the connection pool and click **Select**. You must select a connection pool before typing an initialization string.

        By default, the first connection pool under the database object in the Physical layer is not available for selection. This behavior ensures that you cannot use the same connection pool for initialization blocks that you use for queries. See "About Connection Pools for Initialization Blocks" for more information

        You can change this behavior so that the first connection pool is available for selection by selecting **Allow first Connection Pool for Init Blocks** in the Options dialog, although this is not recommended. See "Setting Administration Tool Options" for more information.
     c. If you selected **Use database specific SQL**, then in the **Database** pane, expand and select the database. Then, enter its associated string.

        Otherwise, in the **Default initialization string** box, type the SQL initialization string needed to populate the variables. See "Examples of Initialization Strings" for examples.

        If you are editing an initialization block to be used by a variable to override a hierarchy column's selection steps, then in the **Default initialization string** box, type the JSON initialization string. See "Initialization Strings Used in Variables to Override Selection Steps" for more informa-

tion.

d. (Optional) Click **Test** to test the data source connectivity for the SQL statement.

e. Click **OK** to return to the Initialization Block dialog.

- Select **Use OBI EE Server**, and then perform the following steps:

a. In the box, enter the SQL initialization string needed to populate the variables.

The string you enter here is processed by the Oracle BI Server, and therefore as long as it is supported by the Oracle BI Server, the string will work with different data sources.

For example, an initialization block might use the function `pi()`, which is specific to SQL Server. However, if you select **Use OBI EE Server**, the query is rewritten by the Oracle BI Server for the appropriate database. In other words, if you change the SQL Server back-end database to Oracle, the query will still work.

See "Examples of Initialization Strings" for additional examples.

b. Click **OK** to return to the Initialization Block dialog.

6. If you selected **LDAP Server** for your data source type, perform the following steps:

a. Click **Browse** to select an existing LDAP Server, or click **New** to open the General tab of the LDAP Server dialog and create an LDAP Server.

b. Click **OK** to return to the Initialization Block dialog.

The LDAP server name and the associated domain identifier appear in the **Name** and **Domain identifier** columns.

7. If you selected **Custom Authenticator** for your data source type, perform the following steps:

a. Click **Browse** to select an existing custom authenticator, or click **New** to create one.

b. Click **OK** to return to the Initialization Block dialog.

8. Click **OK**.

Initialization Strings Used in Variables to Override Selection Steps For analyses that contain hierarchical columns, selection steps can be overridden with session variables or repository variables.

Session and repository variables intended for this purpose must contain valid JSON syntax, rather than SQL syntax, in their initialization strings.

Using JSON, you must define type, column, and members with the following syntax.

```
{
  "type": "Hierarchy",
  "column": {
    "subject_area":"your_subject_area",
    "hier_id":"your_hier_id",
    "dim_id":"your_dim_id",
    "table_name":"your_table_name"
  },
  "members": [
    {
```

```
      "level_id":"your_level_id",
      "values": [
        your_value,
        your_value
      ]
    },
    {
      "level_id":"your_level_id",
      "values": [
        your_value
      ]
    }
  ]
}
```

Where:

`"type"` indicates hierarchy type.

`"column"` indicates the hierarchy column's information such as subject area and table name.

`"dim_id"` is the logical dimension name.

`"members"` indicates which hierarchy level and which member ID.

`"level_id"` is the presentation level name.

Example of Standard Hierarchy Syntax

```
{
  "type": "Hierarchy",
  "column": {
    "subject_area": "A - Sample Sales",
    "hier_id": "H2 Offices",
    "dim_id": "H3 Offices",
    "table_name": "Offices"
  },
  "members": [
    {
      "level_id": "Company",
      "values": [
        10001,
        10002
      ]
    },
    {
      "level_id": "Organization",
      "values": [
        1005
      ]
    }
  ]
}
```

Example of Parent-Child Hierarchy Syntax

```
{
  "type":"Hierarchy",
  "column":{
    "subject_area":"A - Sample Sales",
    "hier_id":"Sales Rep Hierarchy",
```

```
    "dim_id":"H5 Sales Rep",
    "table_name":"Sales Person"
  },
  "members":[
    {
      "level_id":"Grand Total",
      "values":[
        27,
        24,
        18,
        16
      ]
    }
  ]
}
```

Examples of Initialization Strings This section contains the following initialization string examples:

- Example 19–1, "A SQL Statement When Site Uses Delivers"
- Example 19–2, "A SQL Statement When Site Does Not Use Delivers"
- Example 19–3, "A SQL Statement Joining Tables From Multiple Data Sources - When Using the 'OBI EE Server' Setting"

Example 19–1 A SQL Statement When Site Uses Delivers

```
SELECT username, groupname, dbname, schemaname FROM users
WHERE username=':USER'
NQS_PASSWORD_CLAUSE(and pwd=':PASSWORD')NQS_PASSWORD_CLAUSE
```

This SQL contains two constraints in the `WHERE` clause:

`':USER'` (note the colon and single quotes) is the ID the user types when logging in.

`':PASSWORD'` (note the colon and single quotes) is the password the user enters. This is another system variable whose presence is always assumed when the `USER` system session variable is used. You do not need to set up the `PASSWORD` variable, and you can use this variable in a database connection pool to allow passthrough login using the user ID and password of the user. You can also use this variable in a SQL statement.

When using external table authentication with Delivers, the portion of the SQL statement that makes up the `:PASSWORD` constraint must be embedded between `NQS_PASSWORD_CLAUSE` clauses.

The query returns data only if the user ID and password match values found in the specified table. You should test the SQL statement outside of the Oracle BI Server, substituting valid values for the `USER` and `PASSWORD` variables and removing the `NQS_PASSWORD_CLAUSE` clause.

For more information, see *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

Example 19–2 A SQL Statement When Site Does Not Use Delivers

```
SELECT username, groupname, dbname, schemaname FROM users
WHERE username=':USER'
AND pwd=':PASSWORD'
```

This SQL statement contains two constraints in the `WHERE` clause:

':USER' (note the colon and the single quotes) is the ID the user types when logging in.

':PASSWORD' (note the colon and the single quotes) is the password the user enters. This is another system variable whose presence is always assumed when the USER system session variable is used. You do not need to set up the PASSWORD variable, and you can use this variable in a database connection pool to allow passthrough login using the user ID and password of the user. You can also use this variable in a SQL statement.

The query returns data only if the user ID and password match values found in the specified table. You should test the SQL statement outside of the Oracle BI Server, substituting valid values for the USER and PASSWORD variables.

Example 19–3 A SQL Statement Joining Tables From Multiple Data Sources - When Using the 'OBI EE Server' Setting

```
select WUSER.name, wuser_detail.email
from "db-11g/orcl"."NAME"."WUSER',
"sqlexpress"."master"."dbo"."wuser_detail"
where username=:USER:
```

The above query example in the initialization block uses a join query with multiple tables from different data sources (for example, SQLServer, Oracle and XML Files). The query works because when you select the **Use OBI EE Server** option, the query is rewritten by the BI Server for the specified data sources.

Testing Initialization Blocks You should test the SQL statement using the **Test** button or a SQL tool such as the Oracle BI Client utility. If you use a SQL tool, be sure to use the same DSN or one set up identically to the DSN in the specified connection pool.

In online mode, Initialization Block tests do not work with connection pools set to use :USER and :PASSWORD as the user name and password. In offline mode, the Set values for variables dialog is displayed so that you can populate :USER and :PASSWORD.

To test initialization blocks (optional):

1. In the Administration Tool, select **Manage**, then select **Variables**.
2. In the Variable Manager dialog, double-click the initialization block.
3. In the [Repository | Session] Variable Initialization Block dialog, click **Edit Data Source**.
4. In the [Repository | Session] Variable Initialization Block Data Source dialog, click **Test**.

   Note: The **Test** button is disabled when the **Use OBI EE Server** option is selected in offline mode.
5. In the Set value for the variables dialog, verify the information is correct, and then click **OK**.
6. In the View Data from Table dialog, type the number of rows and the starting row for your query, and then click **Query**.

   The Results dialog lists the variables and their values.

The next step is to associate variables with the initialization block.

Associating Variables with Initialization Blocks

The SQL `SELECT` statement in the Default initializer list can contain multiple columns. The order of the columns in the SQL statement and order of the variables associated with the initialization block determine the column value that is assigned to each variable. Therefore, when you associate variables with an initialization block, the value returned in the first column is assigned to the first variable in the list.

For repository variable initialization blocks, when you open a repository in online mode, the value shown in the **Default initialization string** field of the Initialization Block dialog is the current value of that variable as known to the Oracle BI Server. The number of associated variables can be different from the number of columns being retrieved. If there are fewer variables than columns, extra column values are ignored. If there are more variables than columns, the additional variables are not refreshed (they retain their original values, whatever they may be). Any legal SQL can be executed using an initialization block, including SQL that writes to the database or alters database structures, assuming the database permits the user ID associated with the connection pool to perform these actions.

If you stop and restart the Oracle BI Server, the server automatically executes the SQL statement in the repository variable initialization blocks, re-initializing the repository variables.

For session variable initialization blocks, you can select Row-wise initialization. The **Use caching** option is automatically selected when you select the **Row-wise initialization** option. Selecting the **Use caching** option directs the Oracle BI Server to store the results of the query in a main memory cache. See "About Row-Wise Initialization" for more information.

The Oracle BI Server uses the cached results for subsequent sessions. This can reduce session startup time. However, the cached results might not contain the most current session variable values. If every new session needs the most current set of session variables and their corresponding values, you should clear this option.

See "About Using Initialization Blocks with Variables" for more information.

To associate variables with initialization blocks:

1. In the Administration Tool, select **Manage**, then select **Variables**.
2. In the Variable Manager dialog, double-click the initialization block you want to edit. You can edit repository initialization blocks, or session initialization blocks.
3. Click **Edit Data Target**.
4. In the [Repository | Session] Variable Initialization Block Variable Target dialog, perform one of the following steps:
   - Associate variables with the initialization block by doing one of the following:
     - Click **New**, and in the Variable dialog, create a new variable. See "Creating Repository Variables" or "Creating Session Variables" for information about creating variables.
     - Click **Link** to associate an existing variable with an initialization block. Then, in the **Browse** dialog, select the variable to be refreshed by this initialization block and click **OK**.

Note: For the **Custom Authenticator** data source type (Session variables only), the variable `USER` is required.

- Select **Row-wise initialization**. This option is for session variable initialization blocks only. See "About Row-Wise Initialization" for more information. If you select **Row-wise initialization**, the **Use caching** option becomes available.

5. To reorder variables, select a variable and click **Up** or **Down**.
6. To remove a variable from association with this block, select the variable and click **Remove**.
7. Click **OK**.

The next step is to establish execution precedence.

Establishing Execution Precedence

When a repository has multiple initialization blocks, you can set the order (establish the precedence) in which the blocks will be initialized.

First, you open the block that you want to be executed last and then add the initialization blocks that you want to be executed before the block you have open. For example, suppose a repository has two initialization blocks, A and B. You open initialization block B, and then specify that block A will execute before block B. This causes block A to execute according to block B's schedule, in addition to its own.

To establish execution precedence:

1. In the Administration Tool, select **Manage**, then select **Variables**.
2. In the Variable Manager dialog, double-click the last initialization block that you want to be initialized.
3. In the [Repository | Session] Variable Initialization Block dialog, click **Edit Execution Precedence**.
4. In the [Repository | Session] Variable Initialization Block Execution Precedence dialog, click **Add**.

   Add is only available if there are initialization blocks that have not yet been selected.
5. In the Browse dialog, select the blocks that should be initialized before the block that you have open, and then click **OK**.
6. To remove a block, in the [Repository | Session] Variable Initialization Block Execution Precedence dialog, select the block you want to remove and click **Remove**.
7. Click **OK**.
8. If you want the initialization block to be required, in the [Repository | Session] Variable Initialization Block dialog, select the **Required for authentication** option.
9. Click **OK**.

Note: When you select the **Use OBI EE Server** option for an initialization block:

- Execution precedence does not apply, because during user login, an initialization block with the **Use OBI EE Server** option selected is executed after initialization blocks with the **Use OBI EE Server** option not selected.
- The **Required for authentication** option is dimmed, because this type of initialization block is executed after authentication.

When Execution of Session Variable Initialization Blocks Cannot Be Deferred

Execution of session variable initialization blocks cannot be deferred in some circumstances. When the execution of session variable initialization blocks cannot be deferred, a message is displayed that explains why.

See "Assigning a Name and Schedule to Initialization Blocks" for more information.

The following list summarizes the scenarios in which execution of session variable initialization blocks cannot be deferred:

- The **Row-wise initialization** option is selected in the Session Variable Initialization Block Variable Target dialog and the variables have not been declared explicitly with default values.

  Example message: "The execution of init block 'A_blk' cannot be deferred as it is using row-wise initialization."

- The **Required for authentication** option is selected in the Session Variable Initialization Block dialog.

  Example message: "The execution of init block 'A_blk' cannot be deferred as it is required for authentication."

- The **Data Source Type** is not Database.

  Example message: "The execution of init block 'A_blk' cannot be deferred as it does not have a connection pool."

- The initialization block is used by session variables named `PROXY` or `USER`.

  Example message: "The execution of init block 'A_blk' cannot be deferred as it is used by session variable 'PROXY'."

- The initialization block is used by session variables where the **Security Sensitive** option is selected in the Session Variable dialog.

  Example message: "The execution of init block 'A_blk' cannot be deferred as it is used by session variable 'A' which is security sensitive."

- The initialization block is a predecessor to another initialization block which does not have the **Allow deferred execution** option selected.

  Example message: "One of the successors for init block 'A_blk' does not have "Allow deferred execution" flag set. Init block 'B_blk' does not have "Allowed deferred execution" flag set.

Enabling and Disabling Initialization Blocks

You can use the Variable Manager in the Administration Tool to enable and disable initialization blocks.

To enable or disable an initialization block:

1. In the Administration Tool, select **Manage**, then select **Variables**. The Variable Manager appears.
2. In the left pane, select **Initialization Blocks** under **Repository** or **Session**, depending on whether you want to enable or disable repository initialization blocks or session initialization blocks.
3. In the right pane, right-click the initialization block you want to enable or disable.
4. Choose **Enable** or **Disable** from the right-click menu.
5. Close the Variable Manager and save the repository.

Working with Multi-Source Session Variables

Oracle Business Intelligence supports session variables that can be populated from multiple data sources. These multi-source session variables can be used in logical queries or in repository data filters, and contain the union of values from the different data sources.

There is no restriction on the number of values that the multi-source session variable can hold. To create a multi-source session variable, you first create row-wise initialization blocks for each source.

Then, you explicitly define session variables for each source. The format for the session variable names must be:

- `<ms_variable_name>____<source>`

where the separator must be exactly four underscore characters.

This automatically creates a single multi-source session variable, named:

- `<ms_variable_name>`

The component session variable names (<*ms_variable_name*>____<*source*>) appear separately in the Variable Manager in the Administration Tool, but the Expression Builder displays only the single multi-source session variable name (<*ms_variable_name*>).

Note: While the main focus of this section is on the definition and usage of *multi-source* session variables, you may also select the VALUEOF the component session variables in logical queries and data filters.

If any of the row-wise initialization blocks returns null results, this is logged in the Oracle BI Server log, nqserver.log. Values can still be added to the multi-source session variable from other component initialization blocks that succeed in returning values. The multi-source session variable will fail only if all of the component initialization blocks return null values.

You can set execution precedence and deferred execution with multi-source session variables, similar to regular session variables.

Example to Illustrate the Creation and Usage of Multi-Source Session Variables

The following example illustrates how to create and use a multi-source session variable:

1. In the Variable Manager in the Administration Tool, select **Action > New > Session > Initialization Block**.
2. Create a row-wise initialization block called `mvcountry_sebl_init` with the following SQL for **Default initialization string**:

   ```
   select distinct 'MVCOUNTRY____SEBL', country from siebel_table
   ```

3. Create a second row-wise initialization block called `mvcountry_orcl_init` with the following SQL for **Default initialization string**:

   ```
   select distinct 'MVCOUNTRY____ORCL', country from oracle_table
   ```

4. Still in the Variable Manager, select **Action > New > Session > Variable**.
5. Create a session variable called MVCOUNTRY____SEBL, making sure to include four underscores between the variable name and the source name. For Initialization Block, select `mvcountry_sebl_init`.
6. Create a second session variable called MVCOUNTRY____ORCL, making sure to include four underscores between the variable name and the source name. For Initialization Block, select `mvcountry_orcl_init`.

While the component session variables appear in the Variable Manager, the multi-source session variable that has been created, MVCOUNTRY, will appear in Expression Builder.

Using the Multi-Source Session Variable in a Logical Query

You can now use the multi-source session variable MVCOUNTRY in a logical query.

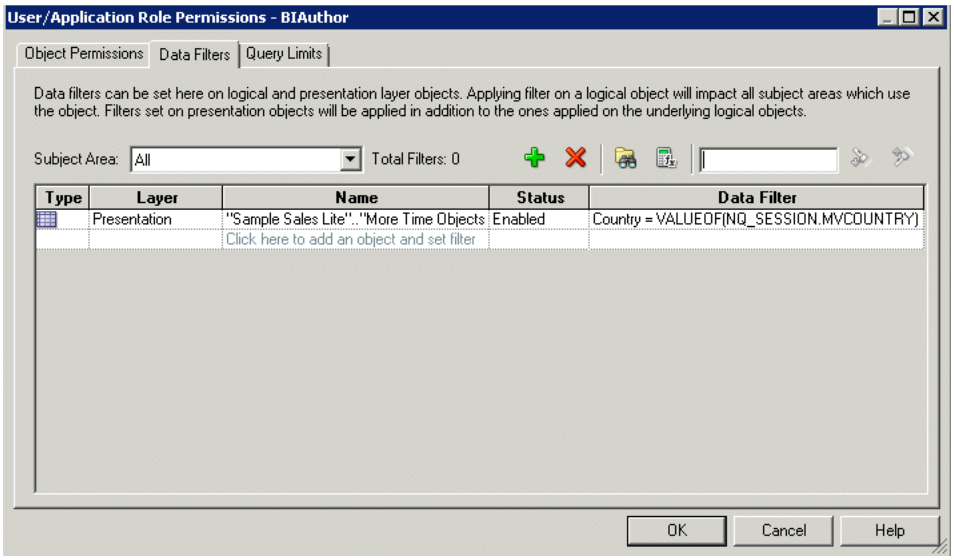
For example:

```
select lastName, firstName, country from employee
where country=VALUEOF(NQ_SESSION.MVCOUNTRY)
```

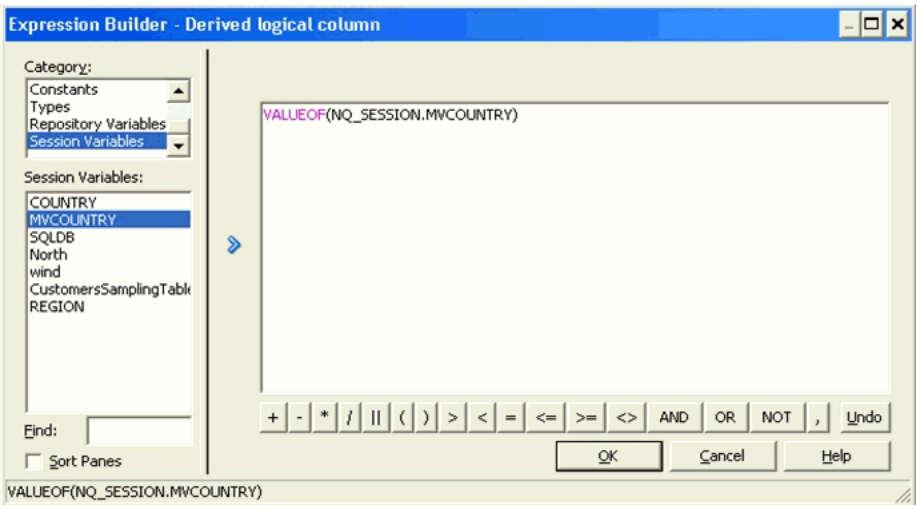
Using the Multi-Source Session Variable in a Data Filter

To use the multi-source session variable MVCOUNTRY in a data filter, perform the following steps:

1. In the Administration Tool, select **Manage**, then select **Identity**.
2. In the Identity Manager dialog, in the tree pane, select **BI Repository**.
3. In the right pane, select the Application Roles tab, then double-click the application role for which you want to set data filters.
4. In the Application Role dialog, click **Permissions**.
5. In the User/Application Role Permissions dialog, click the Data Filters tab.
6. In the Data Filters tab, create the data filter expression:
   - Country=VALUEOF(NQ_SESSION.MVCOUNTRY)



Note that the Expression Builder, as shown in the image that follows, displays only the multi-source session variable MVCOUNTRY, and not the regular session variables that were used during the creation of the multi-source session variable.



List Repository Variables Command

Use the list connection pool command `listrpdvariable` to create a list of repository variables in JSON format for a specific service instance. Use this and the `updaterpdvariables` utility when you need to update more than one variable.

You execute the utility through a launcher script, data-model-cmd.sh on UNIX and data-model-cmd.cmd on Windows. You can find the launcher script at the following location:

Oracle_Home`/user_projects/domains/bi/bitools/bin`

See "What You Need to Know Before Using the Command" for more information.

Syntax

The `listrpdvariables` command takes the following parameters:

```
listrpdvariables -SI <service_instance> -U <cred_username> [-P <cred_
password>] [-S <hostname>] [-N <port_number>] [-V <comma or new line
separated FILE containing selected variables names>] [-O
<outputFile.json>] [-SSL] [-H]
```

Where

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for Oracle BI EE authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you do not supply the password, then you will be prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you are using automated scripting to run the command.

`S` specifies the Oracle BI EE host name. Only include this option when you are running the command from a client installation.

`N` specifies the Oracle BI EE port number. Only include this option when you are running the command from a client installation.

`V` is an optional argument that specifies the repository variable names that you want to list. You must separate the variable names with commas. If you do not pass the `V` argument or pass the `V` argument without listing any variable names, then by default all repository variables are returned.

`O` specifies the output file name with the .json suffix.

`SSL` specifies to use SSL to connect to the WebLogic Server to run the command. Only include this option when you are running the command from a client installation.

`H` displays the usage information and exits the command.

Example

```
data-model-cmd.sh listrpdvariables -SI ssi -U weblogic -P password
-slc01.example.com -N 7777 -V selectedvar.csv -O listrpdvar.json
```

Sample JSON List Repository Variable Output

```
{
    "Title":"List Rpd Variables",
    "Rpd-Variables":[
        {
            "uid":"80000000-3335-155c-991a-0af2537d0000",
            "variable":"RPD_ST_VARIABLE",
            "value":"'rpdStatic Variable'"
        },
        {
            "uid":"c0000000-33c0-155c-991a-0af2537d0000",
            "variable":"DYNAMIC_REPO_VAR",
            "value":"'dynamic repo var'"
        }
    ]
}
```

Sample JSON Output

Note that if there is no match, meaning none of the variable names included in the `V` argument matched the repository variables in the repository, then the JSON output is an empty array list.

```
{
    "Title":"List Rpd Variables",
    "Rpd-Variables":[
    ]
}
```

Update Repository Variables Command

Use the update variable command `updaterpdvariables` to upload a JSON input file or a modified JSON file containing variable information to a specific server instance. Use this and the `listrpdvariable` utility when you need to update more than one variable.

You can create and upload a JSON input file that contains new repository variables (names and values). See "Creating a JSON Input File" and "JSON Input Repository Variable File Example" for more information.

You can also upload an updated JSON file containing modified repository variables (names or values). Use the `listrpdvariable` command to create a JSON file containing a list of repository variables for a specific service instance. Modify the variable information in this file and then upload it to the service instance using the `updaterpdvariables` command. Note that you must not modify the `uid` values for variables in the file. See "List Repository Variables Command" for more information.

You execute the utility through a launcher script, data-model-cmd.sh on UNIX and data-model-cmd.cmd on Windows. You can find the launcher script at the following location:

Oracle_Home`/user_projects/domains/bi/bitools/bin`

See "What You Need to Know Before Using the Command" for more information.

Syntax

```
updaterpdvariables -C <rpdVariablesList.json> -SI <service_instance> -U
<cred_username> [-P <cred_password>] [-S <hostname>] [-N <port_number>]
[-SSL] [-H]
```

Where

`C` specifies the name of the JSON file that you want to upload. Note this file must not contain modified `uid` values for variables. See "Creating a JSON Input File" and "JSON Input Repository Variable File Example" information about the correct syntax for the update repository variables input file.

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for Oracle BI EE authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you do not supply the password, then you will be prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you are using automated scripting to run the command.

`S` specifies the Oracle BI EE host name. Only include this option when you are running the command from a client installation.

`N` specifies the Oracle BI EE port number. Only include this option when you are running the command from a client installation.

`SSL` specifies to use SSL to connect to the WebLogic Server to run the command. Only include this option when you are running the command from a client installation.

`H` displays the usage information and exits the command.

Example

```
data-model-cmd.sh updaterpdvariables -SI ssi -U weblogic -P password -S
slc01.example.com -N 7777 -C listrpdvar.json
```

Creating a JSON Input File

Use the JSON file that was generated when you ran the `listrpdvariable` command as a model for a JSON input file. Using the outputted JSON file as a model ensures that the new file's syntax is valid. See "List Repository Variables Command" for more information.

When writing the input file, note the following information:

uid – This element can be any text.

variable – This element is the new variable's name.

value – This element is the new variable's value. Use singular quotes inside of double quotes. For example "'VALUE'".

JSON Input Repository Variable File Example

```
{
    "Title":"List Rpd Variables",
    "Rpd-Variables":[
      {
        "uid":"80000000-3335-155c-991a-0af2537d0000",
        "variable":"RPD_ST_VARIABLE",
        "value":"'rpdStatic Variable My value'"
      },
      {
        "uid":"c0000000-33c0-155c-991a-0af2537d0000",
        "variable":"DYNAMIC_REPO_VAR_NEW_NAME",
        "value":"'dynamic repo var'"
      },
      {
        "uid":"New1",
        "variable":"NEW_VAR_NAME",
        "value":"'new value for new variable'"
      }
    ]
}
```

A

Managing the Repository Lifecycle in a Multiuser Development Environment

This appendix provides best practice information for managing the lifecycle of the Oracle BI repository when you are using a multiuser development environment.

Building your Oracle BI repository using the multiuser development environment enables you to do the following:

- Build large, interrelated semantic models
- Independently build multiple, independent semantic models to run in the same Oracle BI Server and Presentation Services server
- Develop several branches on different schedules, in parallel, while fixing urgent bugs or enhancement requests on the production version
- Incrementally design and test at the individual and team levels
- Enable individual developers to design and test manageable subsets without impacting each other, yet share their changes with other developers in a controlled, incremental fashion
- Migrate changes to test and production systems in bulk, or incrementally

This appendix covers the development lifecycle of the Oracle BI repository. It does not cover the development lifecycle of the Oracle BI Presentation Catalog used by Presentation Services. This appendix also does not cover how to use the multiuser development environment for Independent Software Vendor (ISV) organizations building portable BI applications for sale as products.

See also Appendix B, "MUD Case Study: Eden Corporation" for detailed examples of how the multiuser development environment is used in a typical business scenario.

This appendix contains the following topics:

Planning Your Multiuser Development Deployment

This section describes tasks you need to perform as part of the planning phase before beginning multiuser development.

This section contains the following topics:

About Business Organization and Governance Process Best Practices

You need to provide a strong, effective governance process to make decisions about shared resources and to resolve conflicts among the many stake-holders. As in any business process, you must have a strong business sponsor, and the steering committee must be staffed with strong business people who can negotiate effectively and make good decisions that will not change over time. Having an effective governance process has proven to be the single most important factor in achieving successful multiuser development with Oracle Business Intelligence.

Before you begin your multiuser development project, you must first lay out the business value, priorities, roadmap, and requirements, as well as lower level details of the design, as described in Table A–1.

Table A–1 Tasks to Accomplish During the Planning Phase

For...	You must...
Strategic requirements	■ Determine which business processes to measure ■ Determine which data sources and subject areas to access
Business requirements for repository objects	■ Select and define metrics, dimensions, and hierarchies ■ Identify objects that will be shared between development teams ■ Resolve conflicts between teams ■ Define Presentation layer subject areas
Security requirements	■ Define Application Roles and corresponding privileges for your user base ■ Define which repository developers can access which metadata and data
Development	■ Determine the styles of multiuser development to use ■ Define areas to break down into MUD projects ■ Determine the owners for metadata objects
Project management	■ Set initiatives - purpose, goals, requirements, scope, schedule, budget ■ Define phases - scope, schedule ■ Allocate resources - hardware, software, databases, developers ■ Decide on a strategy for development branching ■ Prioritize and schedule production updates from different development teams
Operations	■ Negotiate service level agreements ■ Coordinate schedules for updates and downtime

About Technical Team Roles and Responsibilities

This section describes the hands-on roles involved in repository development and its lifecycle. Depending on the size of your company and team, some of these roles might be served by one person.

Repository development roles include:

- MUD administrator (one for each development team, plus backup)
  - Assigns repository password
  - Sets up and maintains MUD projects
  - Manages the master repository shared directories
  - Manages branches and branch merges
  - Manages repository migrations
  - Manages test and production connection pools
  - Manages independent semantic models (has metadata read/write privileges for all models)
- Repository developer (many per development team)
  - Knows the repository password
  - Owns, operates, and maintains a personal development sandbox that includes all necessary Oracle Business Intelligence components
  - Manages user and application role provisioning on their sandbox stack
  - Creates functional and data authorization content in the repository
  - Performs unit testing
  - Performs check-outs, merges, and publishing, as required
- Production Operations staff
  - Knows the repository password (for managing connection pools and applying patches)
  - Applies updated repositories, and applies XML patch updates to the running BI Server's repository
  - Can log in to production computers and read/write the Oracle Business Intelligence directories or run programs
  - Manages the production file system, including the repository directory, logs, and configuration files
  - Manages the production servers (Administration Server, Managed Servers with Java components, and Oracle Business Intelligence system components like Oracle BI Server and Presentation Services)
  - Manages production security, including provisioning users, groups, and application roles
  - Manages and migrates application roles in production
  - Manages production connection pools (in the case where the MUD administrator does not have security privileges for production connection information)

People in other roles outside the repository development team are also involved. These include people administering the test environment and running the tests, and also the Oracle BI Presentation Catalog developers.

Multiuser Development Architecture

Before you can understand best practices for MUD and repository lifecycle management, you need to understand the architecture of the development environment.

This section contains the following topics:

About Multiuser Development Concepts

This section explains fundamental concepts related to developing and deploying systems for multiuser development.

Oracle BI Repository

The Oracle BI repository is the fundamental artifact under development. It defines all the metadata used by the Oracle BI Server for interpreting user requests, applying role-based security, generating queries to data sources, and post-processing the results. Repositories used in multiuser development environments must be in binary (RPD) format, not MDS XML format.

Application Roles and the Policy Store

A secondary artifact under development is the set of application roles. User object permissions, data access filters, and query limits (governors) are defined against these application roles in the repository logic. Presentation Services also uses application roles for assigning its privileges and permissions.

You can use the default policy store embedded in Oracle WebLogic Server, or you can use a separate external policy store. If you are using the embedded policy store, you define application roles in Fusion Middleware Control, which persists them in the Policy Store in Oracle WebLogic Server. You can then use the Administration Tool in online mode to add application roles from your policy store to your repository at design time. At run time, the Oracle BI Server uses the application roles provisioned to each user to apply the correct security privileges to user requests.

Sandboxes, Projects, and Branches

An instance of the repository is usually edited by only one repository developer at a time. Multiple developers work in parallel on subset instances of the repository, called "projects." They work in separate sandbox environments, and merge their changes into a master repository instance frequently to distribute changes and pick up changes made by others in the team. This approach enables the creation of very large enterprise applications. It also enables independent semantic models to be developed by separate teams and merged into the master repository for production hosting in a single Oracle BI Server cluster. Finally, it enables branching and merging so that teams can work on major projects in parallel, and can even make emergency fixes to the main code line in production without disrupting ongoing development projects.

You typically use the Simple install type when installing a development sandbox.

Single, Shared Repository

Presentation Services connects to just one repository that has been uploaded to the Oracle BI Server. The metadata for all semantic models must reside in this single repository, even if the semantic models share no objects. See "About Multiuser Development Styles" for more information about semantic models in a repository.

Repository Password

The repository file is protected by the repository password. The Oracle BI Server needs this password to open and load the repository at startup. It stores the repository password in the secure Credential Store. You must also enter this password when you open the repository in the Administration Tool or other utilities and line commands. Note that user logon credentials are stored in the identity store, not the credential store.

Oracle BI Presentation Catalog

The Oracle BI Presentation Catalog is an important BI application artifact that contains the metadata that defines reports, dashboards, KPIs, scorecards, and other reporting layer objects. The Oracle BI Presentation Catalog is outside the scope of this document. See "Managing Objects in the Oracle BI Presentation Catalog" in *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition* for more information about Oracle BI Presentation Catalog development.

Migration

The completed repository is migrated to test and production systems using Fusion Middleware Control. No downtime is necessary, because you can refresh clustered production Oracle BI Servers with a rolling restart.

Deployment Parameters During Migration

Some repository parameters must change when migrating a repository between development, test, and production systems, such as connection pool settings. These parameters must change because they are based on the deployment, not the application logic. You can automate these updates using the Oracle BI Server XML API (`biserverxmlexec -B`). During multiuser development, developers merging in content are automatically prevented from overwriting the master repository test connection pool and database parameters with their local unit test parameters.

Application Role (Policy Store) Migration

There are several options for migrating application roles between development, test, and production systems. For simplicity, this document assumes you will re-key a small number of application role names by hand. For full information about migrating application roles, and other migration considerations, see "Moving Oracle Business Intelligence to a Target Environment" in *Oracle Fusion Middleware Administrator's Guide*.

Users and the Identity Store

As a best practice, users are not represented by metadata objects in the repository at design time. Also, the repository does not manage or store their credentials. Instead, users must always be provisioned to application roles in the run-time environment to receive privileges. Their credentials, as well as their mapping to application roles through groups, are managed in an external Identity Store. See *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition* for more information.

About Multiuser Development Styles

Choose your style of development based on the size of your team, the number of teams and parallel initiatives, and your requirements for security and availability. Table A–2 shows the multiuser development styles.

Table A–2 Styles of Multiuser Development

Style	Description
Serial Development (Figure A–1)	You can use this method if you have a small number of developers and low concurrency. Development users share a repository file through e-mail, a shared directory, or on a shared development system, and only one of them makes changes at a time. They must coordinate with each other on the development schedule.
Serial Development with Patch Files (Figure A–1)	As a variation on serial development, you can share a base binary repository, and ship changes only between users using patch files.
Shared Online Development (Figure A–2)	The best practice is for only one developer at a time to develop metadata in online mode against a single Oracle BI Server and its repository. However, multiple online users are an option for development situations where communication among the team members is frequent, a higher risk of conflicts is acceptable, and minimum administrative overhead is a goal.
MUD (Figure A–3)	The Multiuser Development feature enables over one hundred development users to work in parallel on a shared, enterprise repository. Each user can develop and unit test in a separate sandbox environment, using only manageable-sized subsets of the metadata. When a unit of work is complete, they can automatically merge and publish it into the branch, where other users can pick up those changes and integrate them with their own metadata. When a project phase is ready for promotion, the MUD administrator migrates it to the test environment, and eventually, production. The MUD administrator manages branches and sub-branches to enable parallel development of independent initiatives or fixes, and merges them into the main branch to incrementally migrate them to test and production environments. The MUD administrator also manages fine-grained "projects," which are the manageable-sized repository subsets individual developers check out to their local sandbox environments. See "Understanding the Multiuser Development Environment" for additional information.
MUD with Multiple, Independent Semantic Models (Figure A–3 and Figure A–4)	There might be cases where you need two or more independent semantic models, rather than a single, integrated, enterprise-wide model. This is common due to security requirements, or when unrelated divisions of a business share a common Oracle Business Intelligence infrastructure. The MUD administrator creates a branch for each model, which enables parallel development and integrated testing for each team's semantic model. When an independent semantic model's branch is ready for promotion to production, the MUD administrator simply merges the branch into main. The MUD administrator can set security on the branches so that each developer can only see the semantic model to which they are assigned, and so that only the MUD administrator and selected production operations staff can access the integrated main model.

Table A–2 (Cont.) Styles of Multiuser Development

Style	Description
MUD with Delegated Administration (Figure A–3 and Figure A–4)	When the independent semantic models are developed by different organizations on different schedules, a centralized MUD administrator might not provide the desired level of local control. In this case, you can provide a dedicated MUD administrator for each independent semantic model's branch. The branch administrator operates in the same way as an ordinary MUD administrator. In this scenario, the MUD super-administrator defines a branch for each organization, checks out the subset repository, and provides it to the branch administrator. When the model is ready for promotion to production, the branch administrator passes the repository back to the super-administrator, who merges it into the main branch for promotion, and then migrates the combined repository to production.

Figure A–1 the serial development style of multiuser development.

Figure A–1 Serial Development

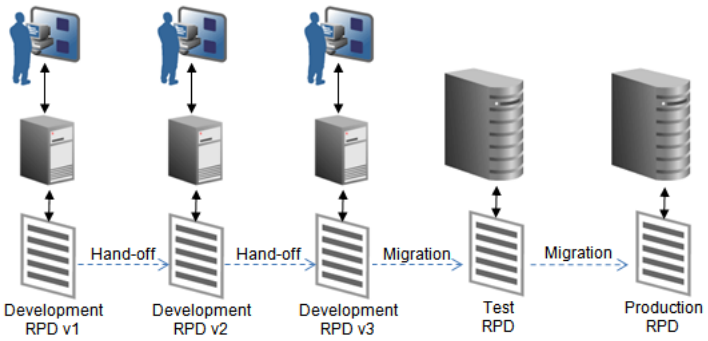


Figure A–2 shows the shared online development style of multiuser development.

Figure A–2 Shared Online Development

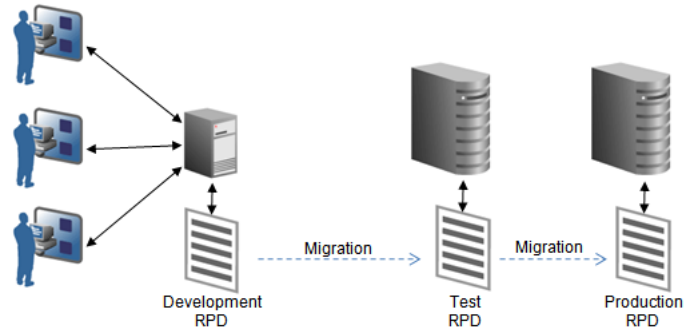


Figure A–3 shows true multiuser development with branching.

Figure A–3 Multiuser Development

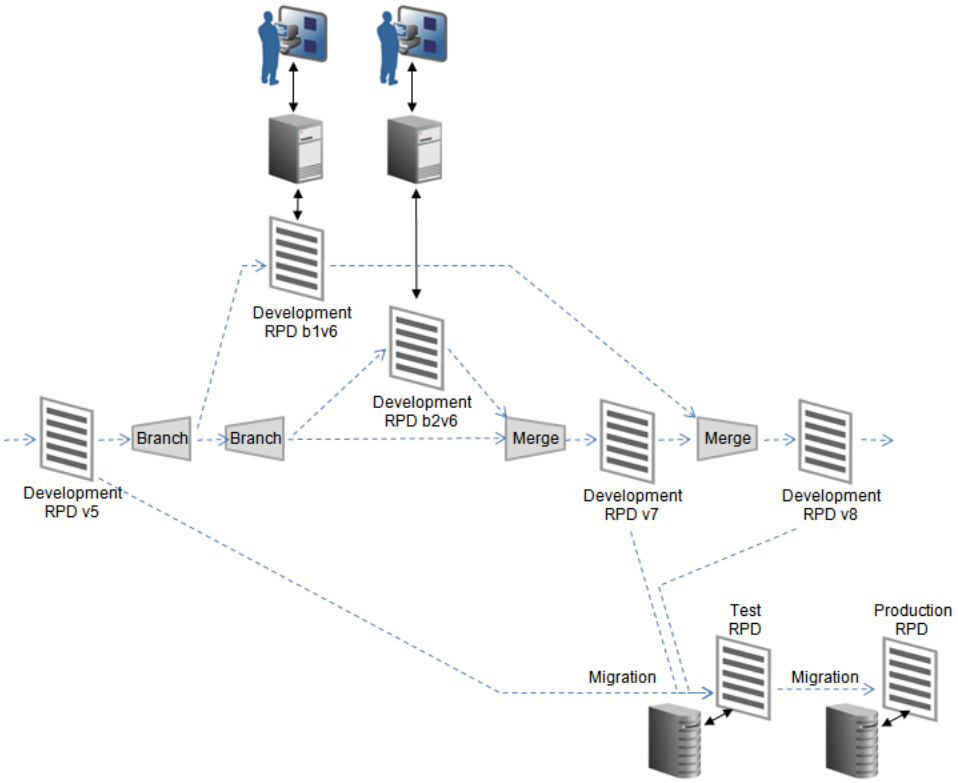


Figure A–4 shows the architecture for a repository with multiple, independent semantic models.

Figure A–4 Repository with Multiple, Independent Semantic Models

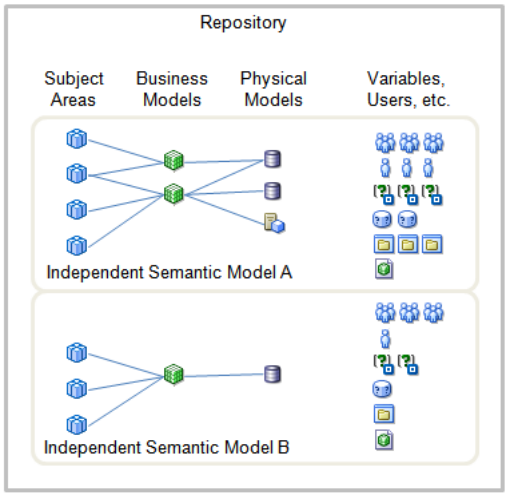


Table A–3 shows which multiuser development styles meet various requirements for security and availability.

Table A–3 Requirements Met by Multiuser Development Styles

Requirement	Serial	Shared Online	MUD with Single Semantic Model	MUD with Multiple Semantic Models	MUD with Delegated Administration
No administrator	Yes	No	No	No	No
Up to five concurrent developers	No	Yes	Yes	Yes	Yes
More than five concurrent developers	No	No	Yes	Yes	Yes
Work on manageable subsets of a large repository, such as Oracle BI Applications	No	No	Yes	Yes	Yes
Built-in checkout, merge, and rollback	No	No	Yes	Yes	Yes
Host independent semantic models in single repository	Yes	Yes	No	Yes	Yes
Incremental migration of units of work to production	No	No	Yes	Yes	Yes
Developers of independent semantic models cannot see each others' metadata	No	No	No	Yes[1]	Yes[1]
Each independent semantic model has its own MUD administrator	No	No	No	No	Yes

[1] Requires secure MUD Directory. An overall MUD administrator must still have access to all metadata from all teams.

Multiuser Development Sandbox Architecture

When using MUD, each developer works on their own, fully dedicated sandbox Oracle Business Intelligence system. You should set up your sandbox to contain all the components you need for development and unit testing.

First, you need to decide whether to use a UNIX or Windows server for your Oracle Business Intelligence stack. Follow these guidelines:

- If you choose the Windows-only option, make sure your system has enough memory. Note that you will need additional resources if you choose to host your database on the same hardware. See "System Requirements and Certification" for information about minimum hardware requirements.
- If you choose the UNIX option, you still need a Windows system to run the Administration Tool. Use the Oracle Business Intelligence Simple install type on the UNIX system, and use the Client install type on the Windows system to install the Administration Tool.

  In online mode, the Oracle BI Server loads the repository from its local repository directory on the UNIX system in:

  `ORACLE_INSTANCE/bifoundation/OracleBIServerComponent/coreapplication_obisn/repository`

  Note that the Administration Tool on Windows also points to a local /repository directory by default, but you can use any directory for offline development.

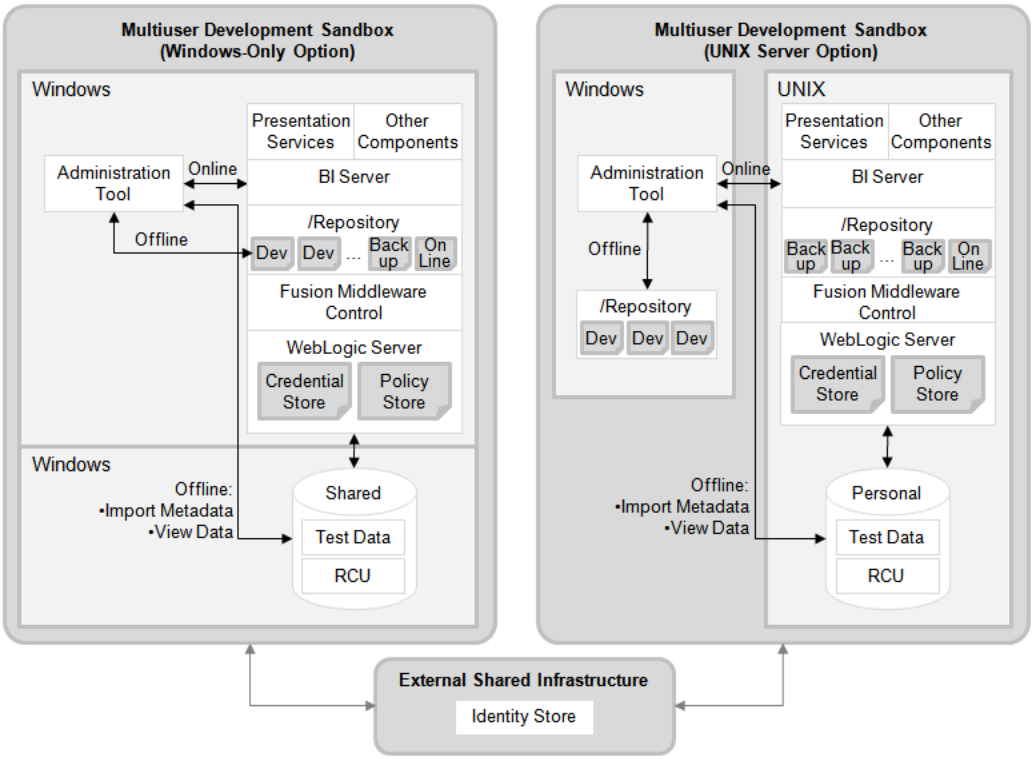
You also need to install a development database. This database can be a dedicated, personal database, or it can be shared among multiple repository developers. Note the following considerations about the development database

- **Platform:** You can choose to host your development database on your sandbox computer if you provide enough memory, or you can host it on a centralized, shared server. Both scenarios are shown in Figure A–5.
- **RCU:** The database must contain the schemas required by Oracle Business Intelligence. You load these schemas using the Repository Creation Utility (RCU). These schemas enable support for Oracle BI Scheduler and annotations for Oracle Scorecard and Strategy Management, provide sample tables for Usage Tracking, and enable many other features. The Oracle WebLogic Server Managed Servers for Oracle Business Intelligence, and all the services that depend on it, require access to a running database with the required RCU schemas in order to operate.
- **Data Source Schemas:** You also need data source schemas for the metadata under development. You can optionally include some data source schemas in your RCU database, or they can be in other databases. Note the following additional information about data source schemas:
  - **Test Data:** The data source schemas should be loaded with test data. If users are testing read-only metadata, the schemas can be shared among multiple development sandboxes. They can be located on the development sandbox computer if enough memory is available.
  - **Multiple Sources:** Optionally, your environment might include multiple data sources needed by your initiative, such as other relational sources, Essbase, Hyperion Financial Management, Microsoft Analysis Services, SAP B/W, and others. These sources can be shared or dedicated, local or remote.
  - **Connectivity:** You must set up connectivity from your Administration Tool and Oracle Business Intelligence stack to each data source. This configuration can include installing the required drivers or clients, setting up ODBC DSNs, setting up native connectivity, and other steps. See Chapter 5, "Importing Metadata and Working with Data Sources" and Chapter 16, "Setting Up Data Sources on Linux and UNIX" for full information.

    Note that for Oracle Database connectivity, Oracle Business Intelligence requires an instance of TNSnames.ora in *ORACLE_HOME*/network/admin.

Figure A–5 shows the architecture of the multiuser development sandbox.

Figure A–5 Multiuser Development Sandbox Architecture



Note: Most developers prefer to disable caching in the development sandbox. This makes it easier to validate and debug physical queries using the log. When the cache is enabled, the physical SQL might not appear in the log, because the request might get fulfilled by the cache. In this release, you must disable caching using Fusion Middleware Control. See "Using Fusion Middleware Control to Enable and Disable Query Caching" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

Multiuser Development and Lifecycle Management Architecture

The overall MUD architecture contains the developer sandbox systems described in the previous section, as well as any test and production systems. In addition, there are several additional major components, as follows:

- **Windows MUD administration system.** This system is maintained by the MUD administrator. Note the following about this system:
  - It provides one shared network MUD directory for the main branch, and additional shared network MUD directories for each side branch. The Windows permissions on each shared directory only allow access to the developers for that branch. Each shared directory stores the master repository for that branch, as well as various control and history files for MUD functions.
  - It has a client installation of Oracle Business Intelligence. The Administration Tool and Oracle BI Server utilities are used for creating and managing MUD projects, performing merges, creating patches, and other MUD administrator tasks. Other Oracle Business Intelligence processes like the Oracle BI Server, as

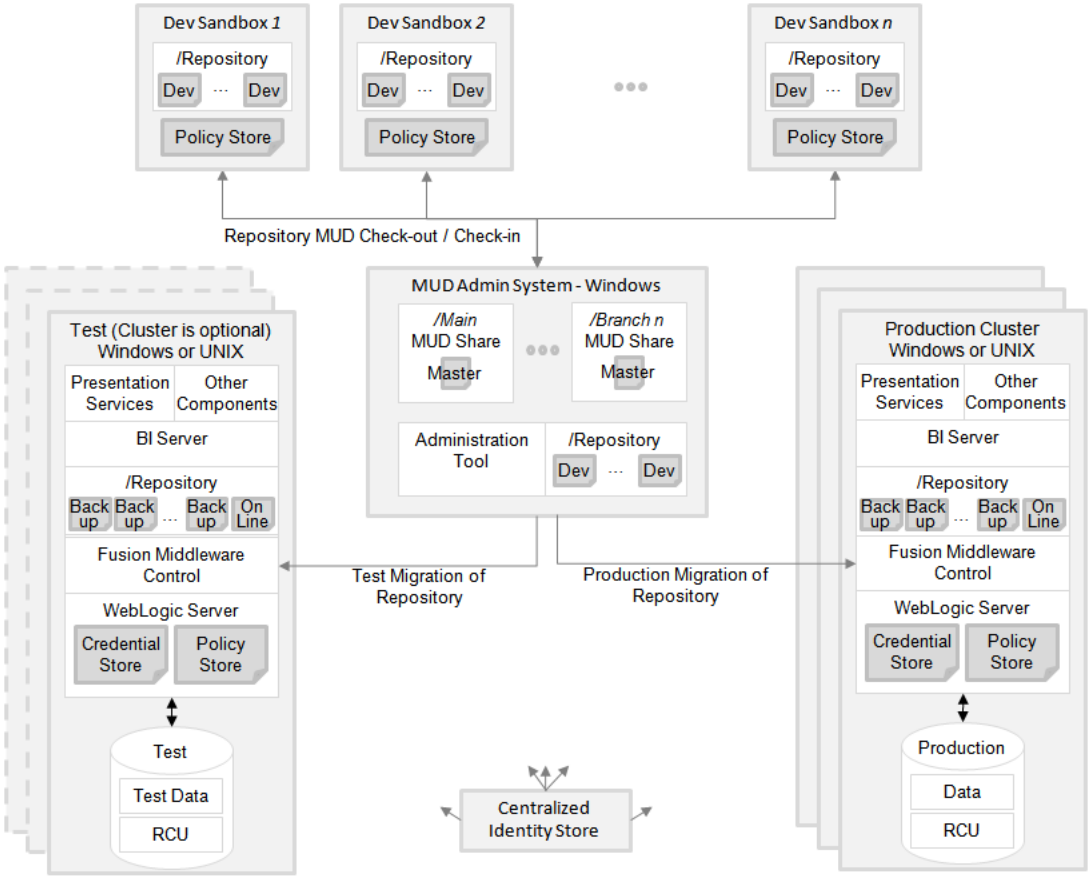
well as the policy store and credential store, are typically not used on this platform.

  - A 32 bit or 64 bit system can be used, because none of the Oracle Business Intelligence Java components, system components, or other infrastructure are used on this computer.

- **One or more test systems.** These systems are used for running integrated tests of merged content. They run the full Oracle Business Intelligence stack, and can be either UNIX- or Windows-based. These systems are frequently clustered.
- **Oracle BI Presentation Catalog system.** Optionally, you might have a system with a full Oracle Business Intelligence stack for developing Oracle BI Presentation Catalog content.
- **Clustered production system.** Eventually, you will have a clustered production system on one of the supported Oracle Business Intelligence platforms.
- **External identity store.** This appendix assumes you are using an external identity store like Oracle Internet Directory.

Figure A–6 shows a sample deployment architecture for the repository lifecycle using the multiuser development environment.

Figure A–6 Example Multiuser Development Deployment Architecture



Understanding the Multiuser Development Environment

MUD is a set of features that enables teams of developers to work in parallel on projects of any size, despite the complex interrelationships and dependencies in the repository model.

With MUD, you can:

- Divide the repository file into subsets
  - Enables users to work with manageable subsets when the repository is very large
  - Enables independent testing for each subset by each developer or team
  - Makes it easier to manage merges later after checking out a branch subset
  - Enables you to separate independent semantic models into secure branches for development
- Incrementally develop, test, and migrate
- Merge subsets and branches, handling conflicts between user changes
- Apply Oracle updates to a packaged BI Application you have modified
- Merge separately developed applications into a single repository
- Access history logging and audit information
- Roll back to historical repository states

The multiuser development feature also provides the following other useful capabilities:

- Coordinates merging into the master, including tracking original repository files
- Provides locking for reliable updates
- Logs changes
- Automatically backs up repositories before each potentially destructive operation

This section contains the following topics:

About Multiuser Development Environment Task Flow

The basic flow of working with multiple users is as follows:

1. A developer defines the "starter" Physical layer, as well as basic facts and subject areas. This provides some basic objects to anchor the MUD projects.
2. The MUD administrator defines projects and puts the RPD into the main branch MUD directory. Note that the MUD directory where the master repository is stored cannot be the same as the Oracle BI Server local repository directory.
3. A developer can now check out one or more projects, do development work, and then merge the changes into the master by publishing to the MUD directory.
4. Meanwhile, other developers check out and do development on the same or other projects (note that projects are for subsetting, not for enforcing security). Because

the publishing step uses a three-way merge, users can check out, develop, and publish their changes in any order. Even property changes to a single object from multiple users are merged. If conflicts do occur between users, the three-way merge feature provides a way for the developer to choose which objects to keep. Communication between users is a key to avoiding and resolving conflicts, and you should have your governance process assign ownership of major objects in order to avoid such conflicts.

5. When a development phase is complete, the MUD administrator can migrate the content to a test system. There might be several iterations back through check out, bug fix, publish, and retest. When the repository passes the testing phase, the MUD administrator can migrate it to the production environment.
6. The MUD administrator can create and manage multiple development branches as large MUD projects. A branch can be secured to ensure that only one development team can work on it. A branch can even be treated recursively as a main, with its own, delegated MUD administrator.

About Multiuser Development Projects

The multiuser development feature is built around a metadata object called a *project*. The project is the unit of check-out from the master repository, and the subsequent merge and publish.

When a master repository becomes very large, a project is a manageable-sized subset that a developer can check out to work on. It is also designed to be self-consistent, so that you can run the consistency checker (analogous to compile-time code checking) and then test it on the Oracle BI Server with a client such as Answers at run time. When you are satisfied with the results, you can merge it back into the master repository so that it becomes part of the larger application. Meanwhile, history is logged and repository backups are automatically created at key points.

MUD features in the Administration Tool streamline this flow for fine-grained developer projects. Similarly, superset projects streamline the management and merging of branches.

The project subset contains a set of metadata objects. You define a project to include a minimum set of objects explicitly, but many others are included implicitly. Having objects implicitly added to projects simplifies your project management task.

The following objects are explicitly specified by the MUD administrator as members of a project:

- Logical fact tables
- Presentation layer subject areas
- Application roles
- Users (although the best practice is to only use application roles in RPD logic)
- Initialization blocks
- Variables

All other objects are implicitly included in a project and are found by the Administration Tool during the check-out process. For example:

- Descendents of the explicitly defined objects. For example, when a logical fact table is included explicitly, all its logical columns are included in the project implicitly.

- Logical dimension tables that join to the selected logical fact tables, and the join objects themselves.
- Logical table sources for the included logical fact and dimension tables.
- Physical tables that map to the logical tables included in the project, and the joins between them.
- Marketing target levels and list catalogs.

Note that objects that are in the list of explicitly defined objects are sometimes included implicitly. For example, if a logical column contains an expression that includes a variable, the variable is implicitly included in the project, even if the MUD administrator does not explicitly add it.

It is normal for projects to overlap. An object that appears in one project can also appear in another project. For example, some customers prefer to create an overall project for each independent semantic model, as well as smaller projects within each independent model for checking out individual units of development work. You can also check out multiple projects simultaneously to work on a larger set of metadata.

See also "Setting Up Projects" for additional information.

How to Create Branches

This section explains how to create main branches, side branches, and delegated administration branches.

This section contains the following topics:

How to Create a Main Branch

The ultimate master repository is usually source-controlled in the main branch, out of which all branches and ultimately all development projects check out. The main branch usually stages the repository in production. That is, to migrate content to production, you merge it into the main branch, and then migrate the main repository to the production system.

Similarly, to fix a production bug, a developer typically checks out of the main branch. The developer then fixes the bug, and then merges it back into the main branch for migration to test and production. Meanwhile, parallel development in side branches is not affected.

To create the main branch as the MUD administrator, you must first create a shared directory and copy the master repository file to it. The directory can be on either Windows or UNIX, but the UNIX share must be accessible by Windows users.

Set the security on the share to only allow access by the appropriate developers. Depending on your requirements, you might only allow developers to access the side branch master repositories, not the main branch master repository.

If this is a new project, you typically have a developer seed the repository with initial content that can be easily split into branch projects.

How to Create a Side Branch

The best practice for branching is to start with a superset MUD project, and then use the MUD check-out, merge, and publish features. Then, individual users or sub-branches use finer-grained projects and check out of the branch master. Using MUD for this functionality provides automatic back-ups at the check-out points, tracks original repositories to ensure correct merges, uses more optimistic merge assumptions that require less user intervention, and provides history and roll-backs.

To create a side branch as the MUD administrator:

1. In the main master repository, create a project that extracts all content required for the branch. Follow these guidelines to create the project:
   - If little or no metadata has been designed in the repository, it is a best practice to seed it with content that can anchor the project. This makes it easier to ensure the project extracts the physical content you need to support the logical fact tables. Usually, this means one or more logical fact tables are created, with at least some representative columns. The columns should be mapped to the physical tables and joins needed to support the fact tables. Finally, create the project and define the objects that belong to it.
   - If content already exists, create the project and define the objects needed in that branch. The branch can overlap with other projects, if necessary.
   - It is also possible to create an empty project for check-out. However, the developer who checks it out must ensure that all the physical objects that need to be implicitly added to the project are mapped to the logical fact table before changes are published. Similarly, the developer must ensure dimensions are joined before changes are published to ensure their inclusion, and must explicitly add any subject areas, variables, initialization blocks, application roles, and users. This method is more prone to errors than seeding the project before defining it.
   - Typically, connection pools for environments such as production must be secured. Ensure that the connection pool settings in the master repository are acceptable for the developers to access. Note that developers typically change the settings to match their local test databases. When changes are published, connection pool and database settings are not merged, to prevent overwriting the settings in the master repository.

     Use the Oracle BI Server XML API to automate connection pool changes required during migrations to production and other environments. See "Moving from Test to Production Environments" in *Oracle Fusion Middleware XML Schema Reference for Oracle Business Intelligence Enterprise Edition* for more information.
2. Create a shared MUD directory for the branch. Every branch should have its own MUD directory. Set the permissions so that only the developers working on that branch have access to it.

   Note that you can use branch permissions combined with project subsetting to prevent developers from seeing metadata that belongs to other teams. Design the projects carefully so that they only extract metadata related to one team. This goal is easiest to achieve if the teams use different business models, subject areas, physical models, variables, initialization blocks, and application roles.

   It is also a best practice to use a consistent system of naming and numbering your branches.

3. Check out the branch project using **File > Multiuser > Checkout**. You can check out into your local repository directory, or another directory.
4. Copy the repository to the branch MUD directory, where it serves as the master repository.
5. Define fine-grained MUD projects for developers to check out from the branch. Inform the developers that the branch is ready for development.
6. Based on your project plan, your developers perform a final merge and publish of their changes when they have completed development and unit tests.
7. When all changes of planned content are published for the phase, the branch project is ready to undergo integrated testing. To accomplish this, migrate the branch master repository file to the test environment. When a bug is found, the assigned developer checks out the appropriate projects, fixes the bug, and tests the metadata. After the changes are published, migrate the branch repository to the test environment again. Note that this branch project can be tested without impacting, or being impacted by, development work in other branches.
8. When integrated testing is complete, the branch is ready to promote to production. Remove the branch master repository from the branch shared directory so that users cannot change it. Copy it back into your local repository directory, and merge it into the main using the Administration Tool. The main repository is now ready for migration to integrated test and production.
9. Typically, the MUD administrator checks out the branch again and places the branch repository in the shared MUD directory for the next phase of development. Note that during the check-out, any changes from other branches, or bug fixes from the main branch, are picked up by the branch repository.

How to Create a Delegated Administration Branch

You can use a branch to delegate local control of a metadata subset to the organization that is developing and maintaining it. To do this, you assign a branch MUD administrator to the branch, who performs the same roles as the main MUD administrator. This approach works best with an independent semantic model, so that you can ensure that there is no metadata overlapping with other groups.

The delegated branch MUD administrator performs the same tasks as the main branch administrator, including defining projects for further branches and creating fine-grained projects for developers.

To create a delegated administration branch as the main MUD administrator:

1. Set permissions on the main MUD directory so that only the main MUD administrator (and the main MUD administrator backup) have access.
2. Create a branch MUD project, branch MUD directory, and checked-out branch master repository as described in the previous section.
3. Set security on the branch MUD directory so that the main MUD administrator and the delegated branch MUD administrator have access.
4. The branch administrator defines projects for further branches, as well as fine-grained projects for developers. If required, the branch administrator deploys additional branches off the delegated branch for development initiatives, with permissions set to allow developers to check out of these repositories.
5. Developers fix production bugs by checking out of the delegated branch MUD directories, because individual developers are not allowed access to the main branch.

6. When developers publish all their changes, the branch administrator checks their branches into the delegated branch for integrated testing.
7. To promote a delegated branch to production after integrated testing is complete, the main MUD administrator performs the following two steps:
   a. Removes the branch master repository from the delegated branch repository shared directory and checks it back into the main branch using the Administration Tool.
   b. Migrates the main branch master repository to production.
8. Typically, the main MUD administrator checks out the branch again and places the branch repository in the delegated branch shared MUD directory for the next phase of development. The branch administrator then checks out next-level branches and places their repositories into the branch shared MUD directories, so that developers can check out their fine-grained projects and begin their work.

Which Merge Utility Should I Use?

There are several different merge tools that are optimized for various situations and environments. When deciding which merge approach and utility to use, you should consider whether you need to perform the task on Windows or UNIX systems. You should also consider your other requirements, such as whether you need to merge changes you made to a semantic model, or whether you need to combine two semantic models from different development efforts.

Table A–4 shows which merge approaches and tools meet various requirements.

Table A–4 Requirements Met by Different Merge Approaches

Requirement	Merge Approach	Tools Used	Platform
▪ Merge a checked-out MUD project back into master repository ▪ Merge a checked-out MUD branch project back into the main branch master repository	Three-way merge	▪ MUD merge	Windows
▪ Combine non-MUD branches and changes back into the main branch	Three-way merge	▪ Merge Repository Wizard (Full Merge selected)	Windows
▪ Apply an Oracle update XML patch to customized, deployed BI Application ▪ Apply an update XML patch you created from development to a deployed repository	Three-way merge	1. Merge Repository Wizard (Patch Merge selected) 2. `Patchrpd` utility	1. Windows 2. All
▪ Combine disjoint logical content with potential ID conflicts	Two-way merge	▪ Merge Repository Wizard (with blank original)	Windows
▪ Combine disjoint content guaranteed in advance by the developer to have no conflicts (all platforms)	Insert-Update-Delete	▪ `biserverxmlexec -B` ▪ `biserverxmlcli` (online) ▪ Copy/Paste XML	All
▪ Combine disjoint content guaranteed in advance by the developer to have no conflicts (Windows only)	Insert-Update-Delete	▪ Copy/Paste Administration Tool tool objects ▪ Administration Tool Import from Repository (deprecated)	Windows

See "Merging Repositories" for more information about merging.

MUD Tips and Best Practices

This section provides tips and best practices for working in a multiuser development environment.

This section contains the following topics:

Best Practices for Branching

Follow these guidelines for creating side branches:

- The MUD directory where the master repository is stored cannot be the same as the Oracle BI Server local repository directory.
- A branch should be a checked-out MUD project. This automates and streamlines many of the tasks of merging the branch back into the main branch, such as using the correct original repository.
- Always put the checked-out branch master repository into its own MUD directory. Then, let developers check out their fine-grained projects from the branch master repository. When branch development, publishing, and testing are complete, remove the master from the branch repository directory and publish it back to the main branch master repository using the Administration Tool. Then, check it out again and place the new version in the branch MUD directory for development of the next phase.
- Use Windows permissions on the branch MUD directory to control which developers have access to it.
- Set multiuser development options by creating an .opt file in the branch MUD directory. As a best practice, define specific administrators, and set Mandatory Consistency Check and Equalize During Merge to Yes. See "Setting Multiuser Development Options" for more information.
- Plan your branches based on the increments of functionality you want to deliver to production. Each branch should contain an increment you want to migrate as a unit.
- If you accidentally merge branches in the wrong order, you can roll them back using the MUD history. See "Viewing and Deleting History for Multiuser Development" for more information.

Best Practices for Setting Up Projects

Follow these guidelines for setting up projects:

- Break your RPD down into fine-grained projects, as small as possible while still being useful. Doing so improves performance and ease of management.
- Break your logical fact tables down into smaller partitions to enable smaller, separate projects.
- For each side branch, overlay a larger project that will extract the branch's contents. This enables the project to manage the checkout and merge of the branch, including tracking of the original repository. Individual developers can check out their development projects from the checked-out branch project. Be sure that all development projects are published back to the side branch before merging it back into the main branch.
- When you add new content to a repository, be sure it is part of your project before you check it in. If you create and publish objects that are not part of a project, they will not be in your extract the next time you check the project out. You or the MUD Administrator must then edit the entire repository, or at least several other projects that do happen to include your new content, and then add the objects to the project at that time.
- Sometimes, you might need to extract several projects at the same time to get all the content you need.

Tip: Presentation layer objects, including subject areas, presentation tables, and presentation hierarchies, are now objects that you explicitly include in the project. Unlike in previous releases, the security settings of the Administration Tool user have no impact on which subject areas, presentation tables, or presentation columns are included in a project when checking it out. Instead, the set of Presentation layer objects determines the scope of the project.

See also "Setting Up Projects" for more information.

Best Practices for Three-Way Merges

Follow these guidelines when performing three-way merges:

- Ensure that you have the original repository from which both the modified and current repositories were built.

  Note that this step is done for you automatically in a MUD merge.
- Typically, you should open the development repository as current, then use the main repository as modified, and the starting point of the branch as original.
- Unit test before merging.
- As a best practice, select **Equalize during merge** and **Check consistency of the merged RPD** in the Merge Repository Wizard. See "Equalizing Objects" for full information about the importance of equalizing objects.

Best Practices for MUD Merges

Follow these guidelines when performing MUD merges:

- Unit test before merging.
- Unit test after merging, but before publishing. Keep in mind that you are holding the lock on the master repository, so keep it brief.

- Be sure your full name is correct in the Tools > Options > Multiuser tab. Doing so assists in logging and in checking who holds the locks.
- When publishing changes, be sure to write useful comments in the Lock Information screen. You or other administrators can use the comments later to help identify historical repositories when you need to perform rollbacks or other tasks.
- When the MUD administrator is editing the master RPD, it must be inaccessible to checkout users. To accomplish this, you can temporarily remove it from the shared directory and place it in another directory, or you can rename it before editing. Make sure to restore it when the edits are complete.

  You can also open the repository in offline mode so that other users are locked out by the Windows file system. Note that you should only use this method when you are sure you will finish all your work in one atomic session.
- Merge frequently. The list of conflicts and decisions needed in a small merge is easy to understand. When the merge is too large, the number of changes make it much harder to understand, and it is much harder to avoid human errors. If you need to roll back, the number of changes discarded is also much bigger. Performance is also better for small merges.
- If performance of merges is a problem, consider breaking the project down into several, finer-grained projects. Also, be sure to merge more frequently, so the number of changes in the merge is smaller and therefore faster.
- Because local connection pool changes are overridden by the connection pool settings in the master repository each time changes are published, the local test settings must be reapplied at each checkout if they are different from the master. It is best to automate application of your local connection pool settings using the Oracle BI Server XML API. See "Moving from Test to Production Environments" in *Oracle Fusion Middleware XML Schema Reference for Oracle Business Intelligence Enterprise Edition* for more information.
- The most successful large teams have formal process requirements and expectations for the communications and tasks between the repository developers and the MUD administrator. For example, they might have a form for a request for the MUD administrator to create a project. Many teams also have service level agreements and lead times, such as 24 hours to create a project.
- Set the option to force a consistency check during MUD merges. A clean consistency check ensures that the Oracle BI Server can load the model correctly, similar to the way a compiler checks to see if code can be generated correctly. Even if the merge seems to succeed, an inconsistent repository may have trouble with starting the Oracle BI Server, online repository editing check-ins, and subsequent merges. See "Setting Multiuser Development Options" for information about how to enable this feature.
- Set the option to force an equalize before merge. This reduces the number of duplicate objects, since it is common for developers to import the same physical tables, for example. See "Setting Multiuser Development Options" for information about how to enable this feature.

  See also "Equalizing Objects" for full information about the importance of equalizing objects.
- Do not delete or change content needed by others, unless you are the owner and have coordinated with the other developers. If you delete a column you do not need in your project, that action usually causes it to be deleted from the master when you merge, even if other users depend on it.

Tip: Presentation object aliases receive special treatment in merges. Their purpose is to hold historical names of objects, so that when names change, old reports do not break. If you changed any names during development, new aliases were added. During merge, you have the option whether to keep any new aliases you have created, or not. You also have the option to keep any or all past aliases, because the historical reports might still exist.

See also "About the Multiuser Development Merge Process" for more information.

Best Practices for Two-Way Merges

Use two-way merge when you need to combine two repositories that were developed separately into a single repository. This situation usually occurs when you need to host two semantic models in a single repository.

Follow these guidelines when performing two-way merges:

- Make sure that the top-level objects in each repository have different names, so there are no unintentional renames or object merges. Check the following objects:
  - Business models
  - Subject areas
  - Physical databases
  - Variables
  - Initialization block
  - Application roles
  - Users
  - Marketing objects
- Equalize before merging. Doing so honors the fully qualified names over which you have control, and assigns upgrade IDs to ensure there will be no conflicts between the two repositories. See also "Equalizing Objects" for full information about the importance of equalizing objects.
- In the Administration Tool, perform a full merge with a blank repository as the original file.

  To create a blank repository, open a new repository, and save it without importing a source or creating any objects. Although this repository contains default security objects, these do not impact your merges.

Caution: Do not use features like Import from Repository or copy/paste in the Administration Tool to move metadata incrementally. These approaches do not correctly merge changes.

Using these features might produce the results you expect most of the time, but this is just good luck. The rest of the time, values of the upgrade IDs in the metadata objects will clash, effectively causing overwrites of random objects. However, you might not notice the problem until much later when you do regression testing. Because upgrade IDs are assigned sequentially, and are only unique within one repository, clashes are very likely.

You should also use caution when using the `biserverxmlcli` and `biserverxmlexec -B` utilities. Be sure to fully understand the information about managing IDs described in "About Using the Oracle BI Server XML API to Merge and Append Objects" in *Oracle Fusion Middleware XML Schema Reference for Oracle Business Intelligence Enterprise Edition*.

See also "Performing Full Repository Merges Without a Common Parent" for more information about two-way merges.

Best Practices for Production Migration

Follow these guidelines when moving from test to production:

- When updating metadata on the production cluster, perform a rolling restart to restart one Oracle BI Server at a time, so that users do not experience down time while changes are being loaded. You can use the BI Systems Management API to programmatically start and stop Oracle BI Servers, or you can restart each Oracle BI Server manually in Fusion Middleware Control.

  For more information, see "Starting and Stopping Oracle Business Intelligence" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*.

- It is not recommended to alter metadata in online mode in production using the Administration Tool.

- It is not recommended to update metadata in online mode in production using the `biserverxmlcli` utility.

Best Practices for Application Roles and Users

Follow these guidelines when working with application roles and users:

- Do not build data access security around user objects in the repository. Instead, define repository permissions, filters and governors based on application roles.

- The set of application roles should be defined by the governance committee. The business team knows what the business roles are, who is allowed to see which data, and which roles are the same from one application to the next. Therefore, the governance committee is in a position to define and name the application roles and decide which roles can be shared across applications.

- When you create a new Application Role, be sure to add it to a project so that you can check it out again after you merge. Also, if you create a placeholder

application role in the Administration Tool in offline mode, make sure to add it to the policy store later.

- You can find whether the application roles used by a repository are provisioned in the system by opening your repository in the Administration Tool in online mode and running the consistency checker. It is recommended that you perform this check each time you migrate the repository and application roles to a new system.
- If you only need to migrate a small number of application roles between environments, you can enter them manually in Fusion Middleware Control on the target system if you are using the embedded policy store in Oracle WebLogic Server.

Troubleshooting Multiuser Development

This section describes common problems and how to resolve them.

Orphan Lock Held on Master RPD

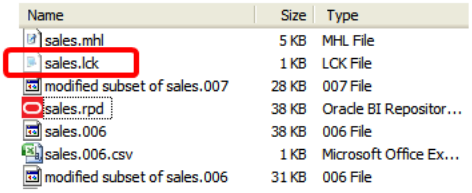
If a user sets a lock by issuing the command to publish changes to the network, it is not cleared until publishing is complete. If the user forgets and leaves for a two-week vacation, the MUD administrator can release the lock.

The lock is stored in a hidden system file in the master directory. If you cannot see the lock file, in Windows Explorer, select **Tools**, then select **Folder Options**. In the View menu, ensure that the option **Show hidden files and folders** is selected.

The lock file has the same name of the master RPD with a .lck extension. Delete the lock file to release the lock on the repository.

Figure A–7 shows a repository lock file.

Figure A–7 Repository Lock File



Object Deleted By Other User

If another MUD developer deletes an object that you need, you can choose one of the following options:

- Roll back to an earlier version, and reapply all the changes since then. The easiest way to roll back is generally to replay history in the history log. To do this, choose **File > Multiuser > History**, and then select an entry and use **Actions > View**.

  See "Viewing and Deleting History for Multiuser Development" for more information.
- Re-create the deleted objects, and equalize so that future merges treat it as the same object.

Project Missing Needed Physical Tables and Joins After Checkout

Physical objects do not explicitly belong to a project. Instead, the physical objects mapped to the logical fact tables in your project are extracted at the time of check out.

To get needed physical objects into your local extract, check out an additional project that does have mappings to the physical objects you need. If there is no such project, then the entire repository must be edited to create mappings to a logical fact table in your project. The MUD administrator can take the repository off line to make that change. Then, your next check out should include the physical objects.

Objects Added in the Last Session Missing from Checked Out Repository

If recently added objects are missing from your checked out repository, you might have forgotten to add the objects to your project before you merged and published. Only objects in your project, or inferred from your project (like dimensions and physical objects), are included in your extracted repository.

To resolve this issue, ask the MUD administrator to add the objects to your project in the master repository, and then check out again.

Object Renamed by Appending #1

This situation occurs when two objects are merged with the same fully qualified name, but with different internal upgrade IDs. The merge logic in this situation determines that the objects are semantically different objects, and changes the name of the object to preserve uniqueness.

To resolve this issue, run the `equalizerpds` utility, which reassigns upgrade IDs so that objects with the same fully qualified names in the two repositories have the same Upgrade IDs. Then, try the merge again. The two objects should be merged instead of causing a rename.

See "Equalizing Objects" for more information.

Rolling Back to Previous Versions

The multiuser development environment stores backup copies of RPDs at appropriate checkpoints. Each time a potentially destructive operation is performed, a new backup is stored in the master directory. It has the name of the RPD, and the extension is a three-digit incrementing integer. Individual developers can also make copies of their RPD files at any time during development.

In the developer's sandbox, the original version of a checked-out project is stored with the name original*rpd_name*.rpd. This version is automatically used if the developer discards changes.

You can also view and roll back to an older version by following these steps:

1. Open the Administration Tool, but not a repository.
2. Select **File > Multiuser > History**.
3. Select the version of interest, and then choose **Actions > View > Repository**.
4. Select **File**, then select **Copy As** to save that version to a new name.
5. Use the older version to replace the latest version, or replace the master repository with the older version.

Example A–1 Replacing the Latest Version

This example explains how to copy an older version to replace the latest version. Assume you are at version 1000 and want to roll back to version 900. In this situation, you have three files: repository.900, repository.1000, and repository.rpd, the current version. To perform the roll back, make a copy of repository.900 and rename it to repository.1001. (This lets you keep repository.1000 in your version history.) Then, copy repository.900 to repository.rpd.

Manually Updating the Master MUD Repository

During the course of Oracle BI repository development in a Multiuser Development (MUD) environment, it might be necessary to make manual changes to the master repository. Because of the highly controlled nature of the MUD process, you need to be careful when performing any manual steps because there is accounting information stored in the MUD history log (.mhl) file. To manually work on the master repository, you must work on the repository in a separate directory from your MUD directory. Then, you must replace both the master RPD and the latest versioned repository in the MUD directory.

For example, follow these steps to manually update a repository named master.rpd:

1. Copy the master repository (master.rpd) out of the MUD directory into a local directory.
2. Use the Administration Tool to make the changes necessary to the local copy of the master repository (master.rpd).
3. When manual edits are complete, copy master.rpd to the MUD directory as master.rpd. For example:

   ```
   copy c:\local\master.rpd c:\mud\master.rpd
   ```

4. In the MUD directory, identify the latest repository with a version number. For example, master.7011.
5. Copy master.rpd to the MUD directory and overwrite the latest versioned repository. For example:

   ```
   copy c:\local\master.rpd c:\mud\master.7011
   ```

B

MUD Case Study: Eden Corporation

This appendix describes a fictional case study that shows how the Oracle Business Intelligence multiuser development environment might be used for a particular business case.

This appendix contains the following topics:

About the Eden Corporation Fictional Case Study

Eden Corporation (a fictional company) recently purchased Oracle Business Intelligence. They have two divisions that are licensed and plan to use the product.

Because of this, the company has two separate initiatives:

- **Initiative S:** The Sales Division wants to use Oracle Business Intelligence for dashboarding and analysis of revenue versus plan. They want to deploy an initial phase to production quickly to meet an immediate need. Then, they want to roll out more functionality in Phases II and III. Initiative S is large enough that they will have two developers working on it.
- **Initiative H:** The Human Resources Division (HR) needs to do dashboarding and analysis of HR data. Initiative H is a smaller initiative, so it will have only one developer. They plan to deliver their application to production between Initiative S Phases II and III.

Note that the Sales developers and the HR developers are not allowed to see each others' data or metadata. The metadata administrator is the only person who has security privileges for all the metadata.

As in all organizations, there will also be a steady stream of urgent requests and occasional bugs from production. The developers will need to deliver fixes for these within days, even though the longer-term initiatives S and H are in development at the same time.

About the Technical Team Roles and Responsibilities

Eden Corporation has staffed the team as follows:

- Adam Straight - MUD Administrator
- Sally Andre - Developer for Sales Division, Revenue project

- Scott Baker - Developer for Sales Division, Quota project
- Helen Rowe - Developer for HR Division

About the Eden Corporation Development Phases

Eden Corporation plans to deploy RPDs to production based on the following timeline:

1. January - Sales Phase I (projects Revenue and Quota)
2. February - Sales Phase II (add project Target, extend projects Revenue and Quota)
3. March - HR (one project used)
4. April - Sales Phase III (extend all three projects)

About the Eden Corporation Topology

Eden Corporation plans to use the following systems for their multiuser development environment:

- MUD Administrator - NT computer with a share
- Sally Andre - NT computer for Administration Tool client, and Linux computer to run the Oracle Business Intelligence stack
- Scott Baker - high-powered NT computer
- Helen Rowe - either of the above
- Test - Linux computer
- Production - Clustered Linux computers

About the Repository Architecture

Because of Eden Corporation's business structure and initiatives, they need to have two independent semantic models in their repository: one for Sales and one for HR. Each of these models can have multiple projects.

Planning the Repository Structure

Eden Corporation knows that it is important to plan the structure of their repository file so that it will be able to support the multiuser development needs of their organization. They assigned owners to major objects, so the developers know who to go to when conflicts arise, and which objects they should not modify on their own.

Tip: When hosting multiple independent semantic models, be sure to itemize the names of top-level objects to prevent duplicate names.

Table B–1 and Table B–2 show the high-level repository objects in main.rpd for both Initiative S and Initiative H, mapped to projects and owners. Note that Adam is the overall owner of both Initiative S and Initiative H.

Table B–1 Initiative S Repository Objects Mapped to Projects and Owners

Object Type	Object	Owner	ProjRevenue	ProjQuota	ProjTarget
physical database	Sample App Data	Sally	Yes	Yes	Yes
business model	Sales	Sally	n/a	n/a	n/a
logical fact table 1	F10 Billed Rev	Sally	Yes	Yes	No
logical fact table 2	F30 Facts Targets	Scott	No	No	Yes
logical fact table 3	F50 Facts Quotas	Scott	No	Yes	No

Table B–1 (Cont.) Initiative S Repository Objects Mapped to Projects and Owners

Object Type	Object	Owner	ProjRevenue	ProjQuota	ProjTarget
logical dimension	(various)	Sally	Yes	Yes	Yes
subject area (1)	Sales Quota	Scott	No	Yes	No
subject area (2)	Sales Revenue	Sally	Yes	No	No
subject area (3)	Sales Target	Scott	No	No	Yes
variable	S_Last_Load	Sally	Yes	Yes	Yes
initialization block	S_Last_Load	Sally	Yes	Yes	Yes
application role (1)	Sales Management	Sally	Yes	Yes	Yes
application role (2)	Sales Rep	Sally	Yes	Yes	Yes

Table B–2 Initiative H Repository Objects Mapped to Projects and Owners

Object Type	Object	Owner	ProjHR
physical database	Human Resources Data	Helen	Yes
business model	HR	Helen	n/a
logical fact table (1)	Payroll Facts	Helen	Yes
logical fact table (2)	Medical Ins Facts	Helen	Yes
logical dimension	(various)	Helen	Yes
subject area (1)	HR Payroll	Helen	Yes
subject area (2)	HR Medical	Helen	Yes
variable	H_Last_Load	Helen	Yes
initialization block	H_Last_Load	Helen	Yes
application role (1)	HR Management	Helen	Yes
application role (2)	HR Rep	Helen	Yes

Phase I - Initiating Multiuser Development (MUD)

In the first phase, both Sally Andre and Scott Baker will develop in parallel. Sally will create the starter content, which Adam Straight will divide into projects. He will then create the MUD directory so that Sally and Scott can check out and perform their development. After unit testing, they merge and publish their changes, and then Adam migrates the repository to the test environment. After a bug fix cycle, Adam promotes the repository to production.

The following sections describe Phase I development:

- Starting Initiative S
- Setting Up MUD Projects
- First Developer Checks Out
- Second Developer Checks Out
- First Developer Publishes Changes to the Master MUD Repository
- Second Developer Publishes Changes to the Master MUD Repository
- MUD Administrator Test Migration Activities

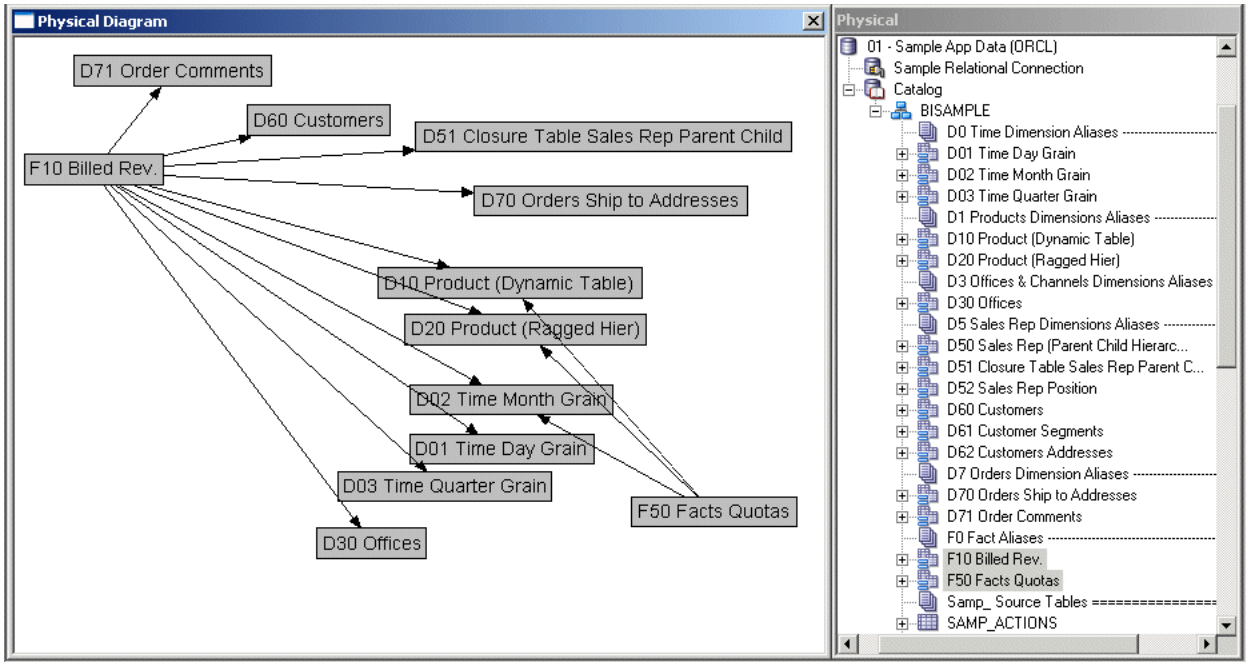
Starting Initiative S

Sally Andre starts off Initiative S from an empty RPD. Because it is easier to divide the repository into MUD projects if you define some logical stars and subject areas first, she begins by developing the physical model needed for Phase I. She includes connection pool details for her own local test data sources.

Tip: The physical model should include the physical tables, the best practice of aliasing all the physical tables to give them meaningful names, and joins.

Figure B–1 shows the physical model for Initiative S.

Figure B–1 Initiative S Physical Model



Sally drags the Physical layer to the Business Model and Mapping layer to create some starter content. She removes unneeded tables, and ensures that the star joins are correct. She also ensures that all the physical tables that will be needed during development have mappings from the starter logical tables, so that they will be included in the projects when they are checked out. For Sally, these steps create two logical fact tables, F10 Revenue and F50 Quotas, that can act as the basis for the projects.

Sally also needs to have some subject areas to map to the projects in the business model. She could drag the entire business model, but a convenient way to accomplish this is to instead right-click the business model and select **Create Subject Areas for Logical Stars and Snowflakes**. This feature creates a subject area from each logical fact table.

Sally does not need to be concerned about the contents of the subject areas yet. All that matters is that each subject area maps to the logical fact table for the same project.

However, she does name the subject areas based on the plan agreed to in the governance meeting: Sales Quota and Sales Revenue.

Sally now has enough content for the MUD administrator to create the first two projects based on the Revenue and Quota fact tables. To review, Sally has made sure that she meets the following criteria at a minimum:

1. At least one logical fact table according to the governance plan, to anchor the projects. The columns of the logical fact tables need not be complete or even properly named, but they do need to be complete enough to map all the physical content.
2. Enough logical dimensions so that the repository will pass the consistency check.
3. Physical content that maps to one or more logical fact tables, so they will be included in projects.
4. The subject areas needed according to the governance plan.

Setting Up MUD Projects

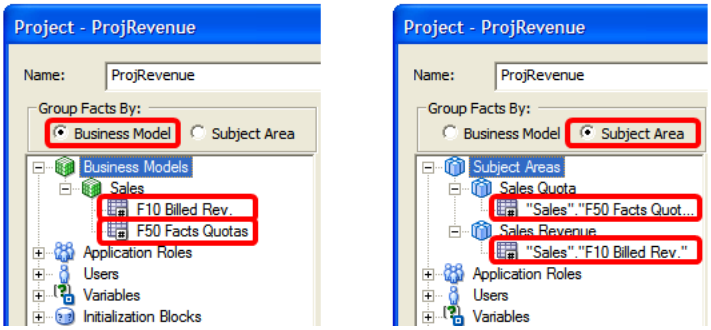
The MUD administrator for Eden Corporation, Adam Straight, now handles the next few steps to create the projects and get them ready for checkout.

First, he creates the MUD directory, RPD_main, where the master RPD will be stored. This master RPD contains the superset of content for the developers. The users will check their projects out of the master, and merge them back in when they want to share their changes. Sally copies her started RPD to the master folder so that Adam can create the first two projects, ProjRevenue and ProjQuota.

First, Adam opens the master RPD in the Administration Tool and selects **Manage > Projects**. Then, in the Project Manager, he selects **Action > New Project**. Adam names the project "ProjRevenue" and proceeds to pick the logical fact tables at the center of the project. The top object in the list expands to show the logical fact tables, but he has a choice of seeing them grouped by the Business Model to which they belong, or by Subject Area.

Figure B–2 shows the different ways Adam can view the logical fact tables.

Figure B–2 Project Dialog with Facts Grouped by Business Model and Subject Area



Adam decides to group facts by Business Model for convenience, although he could have used the Subject Area grouping to select the same fact table. He adds the fact table, plus the default application roles and subject areas specified for this project. Because there are no custom-defined application roles, users, variables, or initialization blocks yet, he cannot yet add them to the project. Adam repeats this process for ProjQuota, the second project.

Tip: Note that some of the explicit objects are the same in both projects, because both projects share application roles. Similarly, many of the implicit project objects are shared, particularly dimension tables in both the logical and physical models. Keep in mind that projects are a convenience for creating small subsets that are easy to work with; they are not for security. It is critical in your governance process that the owner of each top-level object is assigned and documented for the whole team, because this enables developers to avoid conflicts.

Adam included the logical fact table F10 Bill Rev in the project, even though it is owned by Sally Andre, not by Scott Baker, the owner of this project. He did this because Scott needs to create a measure that derives from measures in both fact tables (Sales percent of quota). Again, the point is to provide the user with the subset of content they need to implement their requirements, not just the objects they own.

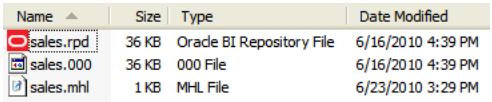
Adam saves the master RPD to the shared drive, RPD_Main, as "sales.rpd". It is now ready for users to check out projects and begin working in parallel.

First Developer Checks Out

Now, the two developers will set up their Administration Tool clients for the master repository, check out their projects, and begin working. Sally starts by setting up her Administration Tool client to use the master repository. To do this, she selects **Tools > Options**, and then selects the Multiuser tab. There, she sets up the pointer to the master repository directory. She also enters her full name, which will be useful in logs and locks. Next, she checks out her project and begins working on it.

Meanwhile, in the Master Repository directory, two new files have been created: sales.000 and sales.mhl. Figure B–3 shows the new files.

Figure B–3 Two New Files in the Master Repository Directory



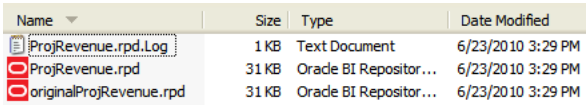
The sales.000 file is an automatic backup created for sales.rpd when Sally checked it out. This file can be used to roll back if problems occur. The sales.mhl file tracks her checkout status and parameters, including project, computer, and user.

Meanwhile, three files have been created in Sally's local repository directory:

- **originalProjRevenue.rpd:** This file is the project subset RPD at the time of checkout. It will be used later as the original in the three-way merge process, and also if Sally discards her changes.
- **ProjRevenue.rpd:** This file contains only the self-consistent subset project (ProjRevenue). This is the file that is open for editing.
- **ProjRevenue.rpd.Log:** This file is the log file for this editing session in the Administration Tool. You can view its contents in the Administration Tool using **File > Multiuser > History**.

  Figure B–4 shows the three files in the local repository directory.

Figure B–4 Three New Files in the Local Repository Directory



Now, Sally begins to work on the model for her application in offline mode. She does not need to change her connection pool settings because she used her own test data source connection pool details when she created the starter content.

Sally starts by opening her fact table and deleting the unused keys based on the modeling best practice. Then, she adds `SUM` aggregation rules to three measures, Discnt_Value, Revenue, and Units. She also changes the name of Discnt_Value to "Discount Amount," Units to "Units Sold," and Revenue to "Sales Revenue."

Sally also needs to add a new column to the D10 Product table, an upper-case version of the Prod_Dsc column called "PRODUCT DESCRIPTION." It uses the following expression: Upper("Sales"."D10 Product (Dynamic Table)"."Prod_Dsc"). She also adds dimension hierarchies, creates a variable called "Constant One", and initializes it to the value 1. She uses it to create a new measure, Constant One. Finally, she saves her work.

Sally starts her sandbox Oracle Business Intelligence stack so that she can add application roles, and then test her repository using Answers. She follows these steps to start her components in the right order and to configure her system environment:

1. Start the database containing the RCU schema, using its standard controls. This database is the local sandbox developer database.
2. Start the sandbox Oracle WebLogic Server Administration Server. For example, on Windows, select **Start > Programs > Oracle WebLogic > User Projects > bifoundation_domain > Start Admin Server for WebLogic Server Domain** and enter the user and password created during installation when prompted.

   Note that if you used an Enterprise or Software-Only install type, you must also start the Oracle WebLogic Server Managed Server using the Oracle WebLogic Server Administration Console. Typically, you use the Simple install type when installing development sandboxes.
3. Log in to the local sandbox Fusion Middleware Control and upload the repository file, making sure to enter the correct repository password. You upload the local subset repository (in Sally's case, the MUD checked-out repository, ProjRevenue.rpd), not the master repository.
4. Also in Fusion Middleware Control, turn off Oracle BI Server caching, so that interpreting the query log is simpler.
5. Still in Fusion Middleware Control, start the system components from the Business Intelligence Overview page.

   Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition provides more information about steps 2 - 5.

Because Sally's Oracle BI Server is on a Linux system, she must set up ODBC connectivity on her Windows computer so that her Administration Tool client can access the BI Server there.

Sally manually adds an Oracle BI Server ODBC DSN pointing to the Oracle BI Server on the Linux computer. See "Integrating Other Clients with Oracle Business Intelligence" in *Oracle Fusion Middleware Integrator's Guide for Oracle Business Intelligence Enterprise Edition* for information about how to create an ODBC DSN for the Oracle BI Server.

Sally is using the Oracle WebLogic Server embedded policy store and needs to add two application roles, "Sales Management" and "Sales Rep." To add the roles, she opens a Web browser on her Windows computer and logs in to Fusion Middleware Control (pointing to her Oracle Business Intelligence stack on Linux). She uses Fusion Middleware Control to create the new roles, maps it to the appropriate users, groups, or other roles, and grants the appropriate permissions to the role.

Tip: See "Creating an Application Role" in *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition* for more information.

Next, Sally needs to add the new application roles to her repository, and then use them for object permissions and data access filters. To accomplish this, Sally does the following:

1. Sally opens the Administration Tool and selects **File > Open > Online**. She picks the local Windows ODBC DSN that connects to her local Oracle Business Intelligence stack, enters her repository password, and also enters the default user name and password for administering her stack that she created upon install.
2. Next, Sally selects **Manage**, and then selects **Identity** to open the Identity Manager. She clicks **BI Repository** in the navigation tree and then clicks the Application Roles tab. She sees the five default application roles, as well as the new ones she just created.
3. Sally double-clicks the Sales Rep application role, and then clicks **Permissions**. On the Data Filters tab, she adds a data filter with an expression that only allows users who belong to this role to see sales that they themselves have made. On the Object Permissions tab, she sets Read, Read/Write, or No Access permissions that allow Sales Rep users to see revenue, but not quota or cost information. On the Query Limits tab, she keeps the defaults for Max Rows and Max Time, and does not set any time restrictions. She clicks **OK** to return to the Identity Manager.
4. Next, Sally double-clicks the Sales Management application role and sets up Data Filters, Object Permissions, and Query Limits appropriate for this role, based on the decisions of the governance committee.
5. Finally, Sally exits the Identity Manager.
6. Sally commits the changes she made in online mode by using the **Check In Changes** menu option. Note that this action propagates the online mode changes to her local subset repository (ProjRevenue.rpd), but does not commit them to the master MUD repository. Sally will publish her changes to the master repository in a later step.

For the new variable and application roles to be in Sally's project the next time she checks it out, she must add them to the project before she checks in her changes. To do this, she performs the same steps that Adam did when he created the projects: She selects **Manage > Projects**, selects her project, selects the new objects in the left pane, and clicks **Add**.

Second Developer Checks Out

While Sally Andre is working on the ProjRevenue project, Scott Baker is getting started on ProjQuota. He set up his Administration Tool options for MUD, checked out his project, and started working.

Scott prefers to work in online mode. Doing this tightens the development/unit test loop, because he is modifying the repository while it is running in the Oracle BI Server. Every time he clicks **Check In Changes** in the Administration Tool toolbar, his changes are applied to the running server. He can then immediately move to Answers and test the changes there. Note that when he adds, deletes, renames, or reorganizes Presentation layer objects, he must reload metadata in the Answers criteria tab to refresh the tree visible there.

First, Scott starts his local Oracle Business Intelligence stack, and uploads his checked-out subset repository. He restarts the Oracle BI Server, opens the Administration Tool, and opens his repository in online mode.

Scott must change the connection pool settings to point to his local test database, because the master repository contains Sally's settings. Note that in the merge process, these connection pool changes will be overridden by the connection pools already in the master repository. Therefore, the next time Scott checks out, he will need to apply his local test connection pool changes again.

Tip: Use the Oracle BI Server XML API to automate connection pool changes required during migrations to production and other environments. See "Moving from Test to Production Environments" in *Oracle Fusion Middleware XML Schema Reference for Oracle Business Intelligence Enterprise Edition* for more information.

Scott's next task is to clean up his logical fact table by removing keys. He also gives a measure a `SUM` aggregation rule and a business-friendly name (Quota Amount).

Scott does not change anything in the F10 logical table because it is owned by Sally. After she merges and publishes her changes to the master RPD, he will do the same. Then he will check out again, picking up her changes.

Next, Scott adds a new measure called "Sales percent of quota" to the F50 table. It derives from both fact tables with the following expression:

```
"Sales"."F10 Billed Rev."."Revenue" / "Sales"."F50 Facts Quotas"."Quota Amount"
```

Note that even if Sally changes the name of Revenue in her project, the merge will identify it as the same object and use the new name in Scott's expression. The merge logic can identify the name change because the upgrade ID of the object is still the same as the original.

Finally, Scott forgets what he learned in the Governance Committee meeting, that all the dimensions are owned by Sally. He has a requirement for an all-capitals version of the D10 Product.Prod_Dsc column called PRODUCT DESCRIPTION. He creates a column identical to the one Sally created. This mistake will be detected and resolved through the merge process during the publishing step in a few moments.

Scott does not need to upload his repository and restart his system because he is working in online mode. Instead, he unit tests his work immediately after committing his changes using **Check In Changes**. Meanwhile, Sally has finished testing her changes.

First Developer Publishes Changes to the Master MUD Repository

Sally has finished creating and unit testing her first batch of changes, so she saves her work and prepares to merge it into the master repository. She chooses **File > Multiuser > Publish to Network**. If she forgot to add any new objects to a project, a detailed warning is displayed so that she can add the objects to her project and try the merge again. Otherwise, the objects are not extracted the next time she checks out the project.

Next, the Administration Tool locks the master repository so that Sally can merge her changes without any chance of corruption from other users' merges.

Tip: For logging purposes, it is a best practice to use the comment field to provide a description of the changes you are publishing. Publishing frequently, or performing a subset refresh, also makes it easier to keep track of changes, and easier to audit the history later. Finally, it is a best practice in Administration Tool modeling to work incrementally, which simplifies testing and reduces the complexity of each task.

Sally's changes cause no conflicts, so they do not appear in the Define Merge Strategy step that is displayed next. However, aliases for presentation objects are a special case where you can choose to keep either the modified (your local version) or current (the master), or merge the two. The aliases were automatically created when Sally changed

the column names, so that reports written to the old names would not break when she put the new names into production. Because Eden Corporation has no reports yet, Sally keeps the aliases empty by selecting Current. She does this for "Sales Revenue," "Units Sold," and "Discount Amount."

Tip: Sometimes, there can be a series of aliases if names change more than once. Because there might be a set of reports using the older names, you can select **Merge Choices** in the Define Merge Strategy screen to keep any aliases already in Current as well as the new ones in Modified.

When the merge step is complete, the master sales.rpd is overwritten with the changes from Sally. A merge log is also stored.

Second Developer Publishes Changes to the Master MUD Repository

Now that Scott has completed his development work for this phase, he selects **File > Multiuser > Refresh Subset** to perform a subset refresh to merge his changes with the latest version of the master repository. The Define Merge Strategy screen asks whether to keep the alias created on the presentation column "Quota Amount." Like Sally, Scott chooses to keep the current repository value, which does not use the alias.

After the subset refresh, Scott unit tests again briefly. Upon inspection, he also notices his mistake of creating the same PRODUCT DESCRIPTION column that Sally did. Because Scott's column was created separately, its internal upgrade ID is different than the one in Sally's. Therefore, even though the name is the same, the merge logic knows it is a different column, and renames it rather than overwriting it by appending #1 (PRODUCT DESCRIPTION#1).

Scott deletes the extra column, connects his logic to Sally's PRODUCT DESCRIPTION column, tests again briefly, and publishes his changes to the network master repository.

Note that if Scott had deleted or modified a different user's object, the error might have been more difficult to resolve. It might have required re-creating and equalizing the object, or rolling back to a backup version of the repository and re-creating his own changes.

MUD Administrator Test Migration Activities

To prepare the repository for the test environment, the MUD administrator, Adam Straight, must now perform several tasks directly on the master repository. In other words, he will use **File > Open > Offline** rather than **File > Multiuser > Checkout**.

Adam begins by opening the Administration Tool and then opening sales.rpd in offline mode. As soon as he does this, other users are locked out, and will get Windows permissions errors if they try to check out projects. If Adam needed to open and close the file several times, he would need to remove the RPD from the shared directory while modifying it elsewhere, so that other users would not be able to check out between his changes.

Adam changes the connection pool settings to match the test environment. Note that when Administration Tool users check out projects, connection pool parameters are not included in the checkout. Usually, the master repository in the MUD directory contains the test connection pools, but each individual developer might need different settings for connecting to their own test databases. At merge and publish, the connection pools in the master repository are not overwritten by developer changes, so that they can continue to point at the shared test databases.

Adam must also ensure that the new application roles are migrated to the test system. Because there are only two, he decides to reenter them in Fusion Middleware Control on the test system. Adam also provisions some test users or groups to the new application roles so the security filters, permissions, and query limits can be tested.

Finally, Adam uploads the repository to the test system and restarts the Oracle BI Server. Using his local Administration Tool, he connects to the test Oracle BI Server in online mode and runs the consistency checker. If any application roles referenced by this repository are missing or incorrect, the consistency checker will list errors for them.

Phase I Testing

The test team can now test the repository. During testing, the test team discovers a bug: "Sales"."F50 Facts Quotas"."Sales percent of quota" was erroneously created with the expression quota/sales instead of sales/quota. The test team writes a bug report, and Scott Baker is assigned to fix the bug.

Scott opens the Administration Tool, checks out ProjQuota, makes the change, changes the connection pool to point to his local test database, and tests on his own sandbox. Then he publishes the changes to the shared MUD directory. He informs Adam that the bug is fixed and that the repository is ready for him to send to test again.

Adam notes that the connection pools are still pointed at the correct test system, because the MUD feature isolates the master repository from connection pool changes in checked out projects. Adam uploads the repository, and restarts the Oracle BI Server.

The test team tests to completion, and the repository is cleared for production.

Phase I Migration to Production

After the repository has passed the testing phase, it needs its database connection parameters updated and can then be uploaded to production. Also, the application roles must be migrated and provisioned.

Based on the plan provided by the governance team, the production operations team knows the new application roles needed. They create them as Adam did for the test environment. They also provision users or groups to those application roles, based on the security specification from the governance team.

Before migrating to production, Adam has to change the connection pool parameters to the values needed for the production database. In Eden Corporation, Adam has the privilege to see the production connection pools, but the repository developers do not. Therefore, Adam cannot change from the test to production connection pools and leave the repository in the master directory, because the developers have Windows permissions to read and write to it. Instead, he creates an XML patch of the connection pools needed for Production. Then, he copies sales.rpd to a secure directory and applies the patch, and then tests to be sure it really does connect to the production data sources. He then uploads the repository to the production system, and starts the production cluster of servers.

Tip: Use the Oracle BI Server XML API to automate connection pool changes required during migrations to production and other environments. See "Moving from Test to Production Environments" in *Oracle Fusion Middleware XML Schema Reference for Oracle Business Intelligence Enterprise Edition* for more information.

Because the master repository still points to the test databases, the Administration Tool users can still be allowed to see it. Meanwhile, new versions of the production

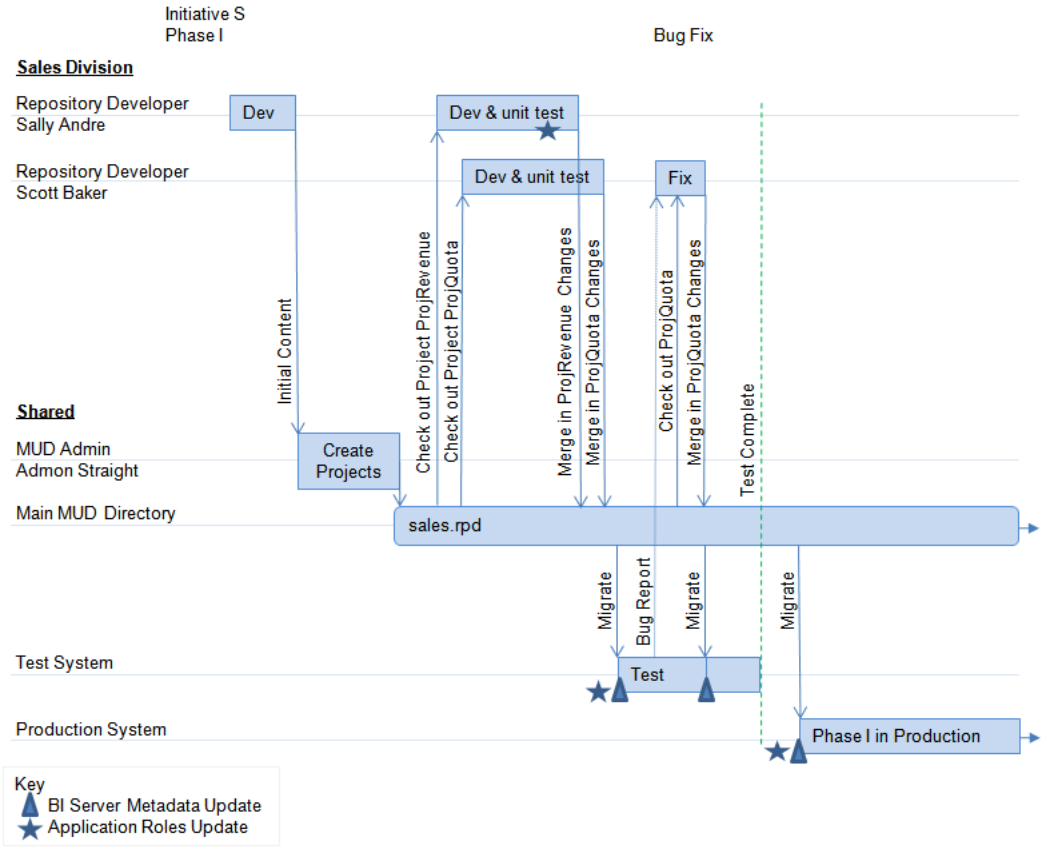
repository can be built at any time by applying the connection pool changes in the XML patch file.

Production validations are now performed. Similar to the migration to the test system, an important validation is to run the consistency checker in online mode to ensure that the application roles are all correct. When this validation is complete, Phase I is in production.

Phase I Summary

Figure B–5 shows the parallel activities for Phase I.

Figure B–5 Summary of Phase I Activities



Phase II - Branching, Fixing, and Patching

In Phase II, development will continue on a new Phase II branch, while a Main branch will track the production application. To manage this work, Adam will add a branch project, and set up a second master repository shared directly, one for Main, and one for the new Phase II branch.

Sally will add more content to ProjRevenue. While she works on that, Scott will add brand new content. After Scott merges and publishes, Adam will create the new project, ProjTarget, and move Scott's new content into it. Meanwhile, they will have to handle any bugs that occur in production, which is still on the main sales.rpd branch.

The following sections describe Phase II development:

- Setting Up the Second Branch

Setting Up the Second Branch

Adam begins by creating another MUD directory to hold the master for the new branch. He sets the Windows share security so that Sally and Scott can read or write to it.

Next, Adam places the main repository into the main MUD directory. He adds a new project for the branch, which encompasses all the existing functionality. Then, he closes the repository, and checks out the branch project in his local Administration Tool repository folder. He copies it to the branch MUD directory, where it now serves as the master for the branch.

Developers Check Out Projects

Sally and Scott check out their projects again, and begin developing Sales Initiative Phase II in parallel with each other, and in parallel with Phase I being in production. Because Scott is adding new content that will become a new project, he needs to check out one or more other projects that will provide the shared objects to which he needs to map or join in the new content. He chooses to check out ProjQuota.

Patch Fix for the Main Branch

While Sally and Scott are developing Phase II, an urgent CEO request is escalated to them. The CEO wants the key sales managers to see a new measure called "Sales Quota Variance" on their dashboards within two days.

Scott closes his work on the new project on the Phase II branch; it will stay checked out. Then, he checks out the project that will contain the new measure, ProjQuote, from the main branch master repository (sales.rpd). He creates the new measure and corresponding presentation column, tests it locally, and publishes the changes back to the main branch.

Scott then reopens the checked-out Phase II repository from his local drive and continues development.

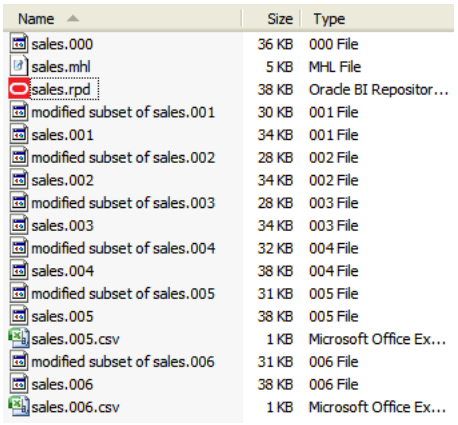
Meanwhile, Adam sends the new sales.rpd to the test environment, where the test team validates the fix.

Next, Adam prepares to send the fixed repository to Production. Rather than send the entire repository, however, he sends a patch of the change.

To create the patch, Adam compares the modified repository to the one that is currently running in production. The repository running in production is the same as the main repository just before the new changes were merged in, so it is one of the backup repositories in the MUD directory. The current repository running in production is the backup called sales.006, the same one he identified as the original for the upcoming branch merge. He copies this to sales.006.rpd so the Administration Tool can see and open the file. (He cannot simply rename it, because it may be needed for another merge later.)

Figure B–6 shows the files in the MUD directory, including sales.rpd and sales.006.

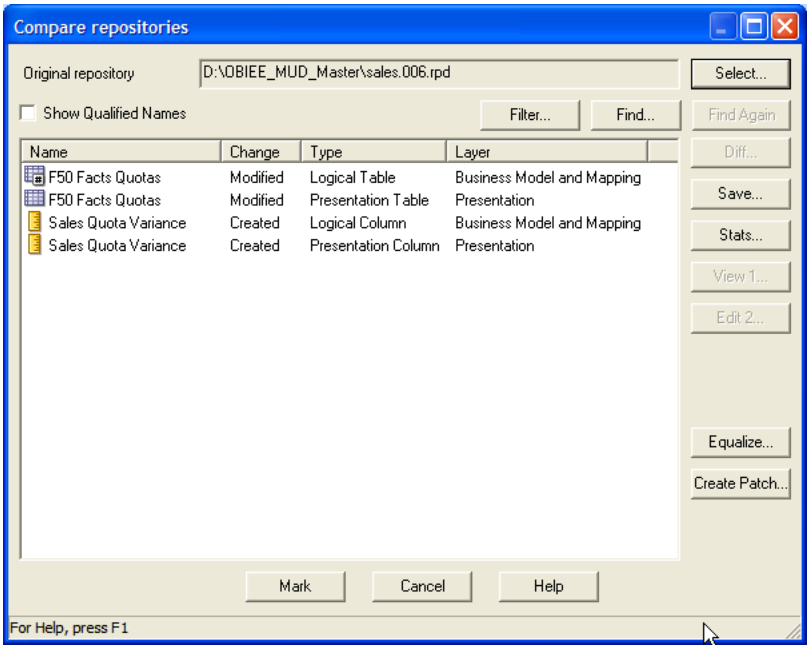
Figure B–6 Renaming sales.006 to sales.006.rpd



Next, Adam opens the repository containing the update, sales.rpd. He selects **File > Compare**, and chooses the sales.006.rpd as the old version to compare. The Compare repositories dialog shows the differences between versions that will be included in the patch.

Figure B–7 shows the Compare repositories dialog.

Figure B–7 Compare Repositories Dialog for sales.rpd and sales.006.rpd



Next, Adam clicks **Create Patch** and saves the result as Patch_variance.xml. The patch contains just the objects needed to apply the two new columns, and their associated interconnections.

Tip: More complex patches might also delete objects, or overwrite objects to merge in new property values.

Adam's patch appears as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Repository xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <DECLARE>
```

```
  <LogicalTable name="F50 Facts Quotas" parentName="&quot;Sales&quot;"
  parentId="2000:68667" parentUid="2160843965" id="2035:69454" uid="2160843966"
  x="718" y="288">
    <Description/>
    <Columns>
      <RefLogicalColumn id="2006:69460" uid="2160844041"
      qualifiedName="&quot;Sales&quot;.&quot;F50 Facts Quotas&quot;.&quot;Quota
      Amount&quot;"/>
      <RefLogicalColumn id="2006:69786" uid="2160845070" qualifiedName=
      "&quot;Sales&quot;.&quot;F50 Facts Quotas&quot;.&quot;
      Sales percent of quota&quot;"/>
      <RefLogicalColumn id="2006:70033" uid="2160845342" qualifiedName=
      "&quot;Sales&quot;.&quot;F50 Facts Quotas&quot;.&quot;
      Sales Quota Variance&quot;"/>
    </Columns>
    <TableSources>
      <RefLogicalTableSource id="2037:69456" uid="2160844747"
      qualifiedName="&quot;Sales&quot;.&quot;F50 Facts Quotas&quot;.&quot;
      F50 Facts Quotas&quot;"/>
    </TableSources>
  </LogicalTable>
  <LogicalColumn name="Sales Quota Variance" parentName=
  "&quot;Sales&quot;.&quot;F50 Facts Quotas&quot;" parentId="2035:69454"
  parentUid="2160843966" id="2006:70033" uid="2160845342" isDerived="true"
  isWriteable="false">
    <Description><![CDATA[quota - sales]]></Description>
    <Expr><![CDATA["Sales"."F50 Facts Quotas"."Quota Amount" - "Sales".
    "F10 Billed Rev."."Sales Revenue" ]]></Expr>
  </LogicalColumn>
  <PresentationTable name="F50 Facts Quotas" parentName=
  "&quot;Sales Quota&quot;.&quot;&quot;"
  parentId="4004:69706" parentUid="2160844968" id="4008:69707"
  uid="2160844969" hasDispName="false" hasDispDescription="false">
    <Description/>
    <Columns>
      <RefPresentationColumn id="4010:69711" uid="2160844973" qualifiedName=
      "&quot;Sales Quota&quot;..&quot;F50 Facts Quotas&quot;.&quot;
      Quota Amount&quot;"/>
      <RefPresentationColumn id="4010:70032" uid="2160845338" qualifiedName=
      "&quot;Sales Quota&quot;..&quot;F50 Facts Quotas&quot;.&quot;
      Sales percent of quota&quot;"/>
      <RefPresentationColumn id="4010:70036" uid="2160845345" qualifiedName=
      "&quot;Sales Quota&quot;..&quot;F50 Facts Quotas&quot;.&quot;
      Sales Quota Variance&quot;"/>
    </Columns>
  </PresentationTable>
  <PresentationColumn name="Sales Quota Variance" parentName="
  &quot;Sales Quota&quot;..&quot;F50 Facts Quotas&quot;" parentId=
  "4008:69707" parentUid="2160844969" id="4010:70036" uid="2160845345"
  hasDispName="false" hasDispDescription="false" overrideLogicalName="false">
    <Description><![CDATA[quota - sales]]></Description>
    <RefLogicalColumn id="2006:70033" uid="2160845342" qualifiedName=
    "&quot;Sales&quot;.&quot;F50 Facts Quotas&quot;.
    &quot;Sales Quota Variance&quot;"/>
  </PresentationColumn>
  </DECLARE>
</Repository>
```

Tip: Unlike migrating an entire repository, there is no need to make any connection pool changes before applying this patch. The correct connection pool settings are

already in the repository running in production. The patch will not affect this logic, so the connection pools will stay correct without an intervention.

Finally, Adam must have this patch migrated and applied to the production system. There are several ways to accomplish this:

1. **Patch main repository offline and upload.** Adam can apply the patch to a copy of the production repository locally on his Windows computer by using the Administration Tool to perform a patch merge. Then, he can upload the repository to the production system, like Sally did earlier in her sandbox. Because the production system is clustered, he must restart all the Oracle BI Servers after uploading the repository. Adam can restart manually through Fusion Middleware Control, one server at a time. If he performs a rolling restart in this way, end users do not see any unavailability. Alternatively, Adam or one of the operations staff can write a script using the BI Systems Management API to automate a rolling restart.
2. **Patch production repository in place using patchrpd utility:** The operations staff can log onto a production system directly, and apply the XML patch using the `patchrpd` utility. Note that if any conflict occurs, the utility will cancel the update and exit without making changes. If the update is successful, the operations staff can then perform a rolling restart, as described in the previous paragraph.
3. **Patch running system using biserverxmlcli utility:** This method is not recommended for production systems.

Tip: If you have privileges to log on to a production Oracle BI Server using the Administration Tool in online mode, you can use **File > Copy As** to copy it to your local drive.

Finishing and Merging Phase II Branch

Sally and Scott complete their changes in the new branch and publish them.

Adam now adds Scott's new content to a new project, projTarget. He performs the same steps as before to send the branch repository to the testing team.

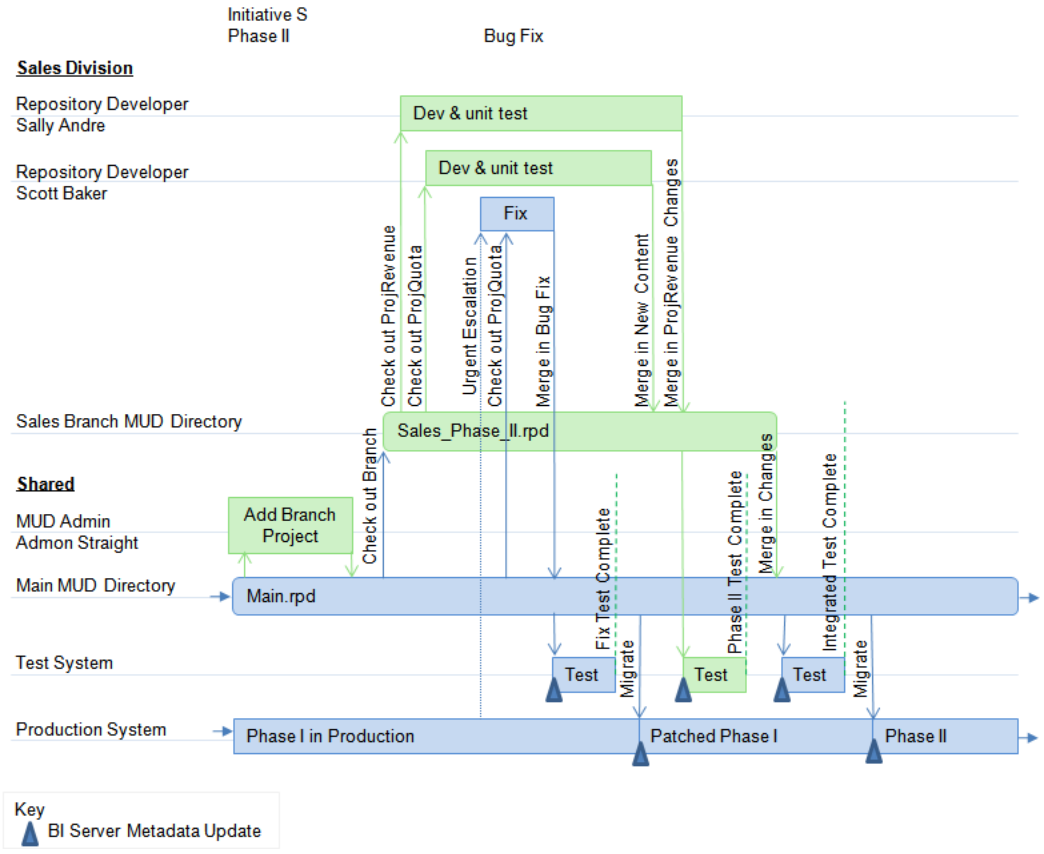
When testing is complete, the branch must be merged back into the main branch using MUD merge. Doing this merges the production patch with the newly developed content, so that can be moved to production later.

Now, sales.rpd contains all the changes, and the branch is no longer needed. Sales.rpd is sent to integrated test, to ensure the merged content does not cause any bugs in the existing content. When integrated testing is complete, Adam creates another patch containing the changes, and has the operations staff apply it to the running production system. Sales Initiative Phase II is now in production.

Phase II Summary

Figure B–8 shows the parallel activities for Phase II.

Figure B–8 Summary of Phase II Activities



Phase III - Independent Semantic Model Development

In the next phase, Sally and Scott begin development of Phase III of the Sales initiative. Meanwhile, Helen Rowe builds the first phase of the HR initiative and brings this new independent semantic model into production.

The following sections describe Phase III development:

- Security Considerations for Multiple Independent Semantic Models
- Sales Semantic Model Developers Check Out
- HR Semantic Model Developer Builds Content
- Phase III Summary

Security Considerations for Multiple Independent Semantic Models

Helen's application has highly sensitive personal information, such as salaries and medical information. Meanwhile, the Sales application has legally sensitive financial information. Due to corporate security compliance, these two teams are not allowed to see each other's data or metadata. They also have little content they could share, other than generic dimensions like time dimensions. Finally, they have different business drivers, budgets, and schedules.

For these reasons, the Eden Corporation governance committee decided to use independent semantic models in the repository: one for Sales, and the other for HR. This approach requires the two teams to ensure that there are not any shared objects, and there can be no conflicts between their content. The easiest way to ensure this is to

make sure that the names for all top-level objects do not conflict. Even variables and application roles must be different.

Tip: Some governance committees ensure that top-level objects do not conflict by requiring developers to put a prefix specific to each semantic model before the name of each top-level object, such as S_ for Sales and H_ for HR. This practice makes it easy to see which objects belong to which organizations. Other committees prefer to keep a master list of top-level objects, and require new applications to submit top-level object names for review to ensure there are no conflicts. In addition, two-way merges can catch any mistakes before overwrites can damage content or cause unexpected object name changes.

Another security requirement is the need to apply security to the separate MUD directories so that only the correct developers have access to each repository. Sally and Scott can only see and check out from the Sales MUD directory, and Helen can only see and check out from the HR MUD directory. The Main directory continues to exist, since it must hold the merged master that is actually in production, but now only Adam has privileges to see or modify that directory.

At Eden Corporation, a final security requirement is to disable the ability for independent semantic model developers to access the running repository in online mode after the merge. There is only a single repository password, so a developer who has the password and access to the repository can see and modify all its contents in offline mode. However, in online mode, the developer also needs a data access user name and password to log on to the Oracle BI Server. To enforce this security requirement, Adam must ensure that the developers have no privileges to log on to the production or test system in this way. Alternatively, the production operations staff can change the repository password to one that only they know, but this task must be performed on a Windows computer because repository passwords are changed using the Administration Tool.

Sales Semantic Model Developers Check Out

Sally and Scott check out their projects from the new, secure sales branch MUD directory. They begin their work.

HR Semantic Model Developer Builds Content

Because Helen is working alone on her secure, independent semantic model, she does not yet need to check out a project. In fact, she needs to start building her content from a new, blank repository on her local computer. She follows the usual steps of building and unit testing content incrementally.

When she is done with unit testing, Helen has a complete, free-standing repository. She sends it to Adam, who manually updates the master repository or performs a two-way merge in a separate location. Adam uses one of the following merge methods:

Manually updates the repository

1. First, Adam equalizes the two repositories to reassign IDs honoring the different names given to the top-level objects. This practice ensures that there will be no conflicts during the merge.
2. Next, Adam copies the master repository out of the MUD directory and into a local directory, performs the required manual updates to add the contents of Helen's repository into the master repository, and then copies the master repository back into the master directory.

Performs a two-way merge in a separate location

1. First, Adam equalizes the two repositories to reassign IDs honoring the different names given to the top-level objects. This practice ensures that there will be no conflicts during the merge.
2. Next, Adam copies the master repository out of the MUD directory and into a local directory, performs a two-way merge by using the Merge Repository Wizard, and then copies the master repository back into the master directory.

Tip: To create a blank repository, select **File > New Repository**. Then, provide a name (such as blank.rpd) and a repository password. Choose **No** for **Import Metadata** and then click **Finish**.

After the merge, Adam creates a new project for managing the content going forward, hr_payroll. He adds Helen's content to the project. Adam then checks it out of main and posts it to the HR Branch MUD directory. Using a project checkout makes managing IDs and merges easier later.

Adam adjusts connection pool parameters, and migrates the repository to the test computer. When a bug is found, Helen checks out the hr_payroll project, fixes it, unit tests it, and publishes it. (Note that she checks her functional project out of the checked-out branch project.) Adam migrates it to the test system for further testing. When testing is complete, he merges the completed HR branch repository back into the main branch, and sends the integrated repository to integration testing on the test system.

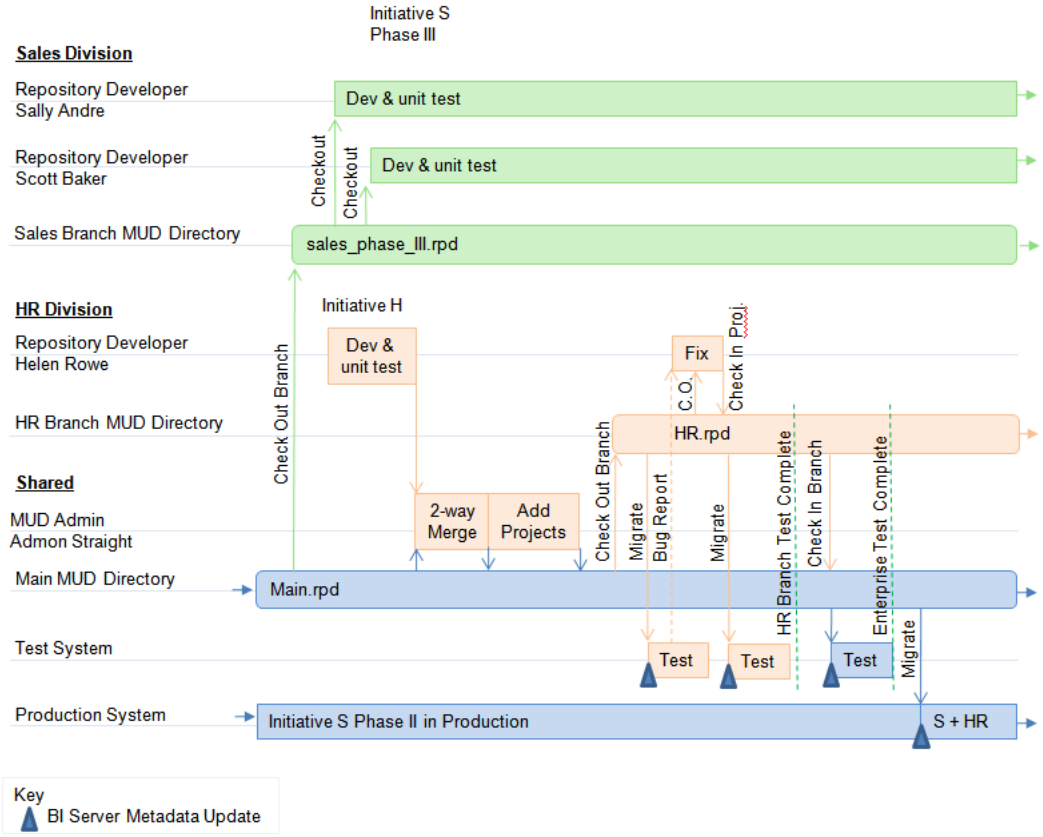
When the integrated repository completes testing, it is ready for migration to production. Again, the options are complete repository migration, or applying a patch to the production environment using `patchrpd`. Both methods require a rolling restart.

After this step, the production repository contains content for both Initiative S and Initiative H.

Phase III Summary

Figure B–9 shows the parallel activities for Phase III.

Figure B–9 Summary of Phase III Activities



C

Logical SQL Reference

This appendix provides syntax and usage information for the Logical SQL statements understood by the Oracle BI Server. Oracle BI Server Logical SQL includes standard SQL, plus special functions (SQL extensions) such as `AGO`, `TODATE`, `EVALUATE`, and others. Logical SQL queries resolve to Presentation layer objects.

This appendix contains the following topics:

About Logical SQL in Oracle Business Intelligence

The Oracle BI Server accepts SQL `SELECT` statements from client tools. Additionally, the Oracle BI Administration Tool enables you to define logical columns with complex expressions. This appendix explains the syntax and semantics for the `SELECT` statement and for the expressions you can use in the Administration Tool to create derived columns.

The abstraction provided by the Presentation layer and Business Model and Mapping layer enables clients to query data with Logical SQL only, so that the interaction with actual physical sources is handled by the Oracle BI Server. The complexity of the multiple source languages needed to communicate with each data source type is hidden from users and clients.

In Answers, you can view the Logical SQL queries issued by Oracle BI Presentation Services for particular analyses by viewing the SQL Issued section of the Advanced tab of the Analysis editor. If you have the appropriate privileges, then you can also view SQL by displaying the Manage Sessions page in the Administration tab. Click **View Log** from the Manage Sessions page to see further details.

In Answers, there are also several places where you can issue Logical SQL. If you have the appropriate privileges, then you can use the Issue SQL page in the Administration tab to enter any SQL code to send to the Oracle BI Server. If an analysis does not contain hierarchical columns, member selections, or groups, then you can use the Advanced SQL Clauses fields in the Advanced tab of the Analysis editor. You can also enter SQL in the New Filter dialog.

In the Administration Tool, Logical SQL appears mostly in the form of expressions related to objects in the Business Model and Mapping layer. You typically create SQL functions in Expression Builder; see "About the Expression Builder Dialogs" for a summary of the places in the Administration Tool where you can build Logical SQL expressions.

Other clients, like Oracle BI Publisher, Oracle's Hyperion Interactive Reporting, the Oracle BI Add-in for Microsoft Office, and Essbase, also provide their own interfaces to view and issue Logical SQL to the Oracle BI Server.

SQL Syntax and Semantics

This section explains SQL syntax and semantics. The following topics are included:

Syntax and Usage Notes for the SELECT Statement

The `SELECT` statement, or query specification, is the way to query a decision support system through the Oracle BI Server. A `SELECT` statement returns a table to the client that matches the query. It is a table in the sense that the results are in the form of rows and columns.

The `SELECT` statement is the basis for querying any structured query language (SQL) database. The Oracle BI Server accepts logical requests to query objects in a repository, and users (or query tools) make those logical requests with ordinary SQL `SELECT` statements. The server then translates the logical requests into physical queries against one or more data sources, combines the results to match the logical request, and returns the answer to the end user.

The `SELECT` statement in Logical SQL differs from standard SQL in that tables do not need to be joined. Any join conditions supplied in the query are ignored because the join conditions are predefined in the Oracle BI repository.

This section provides the basic syntax for the `SELECT` statement, as well as definitions for individual clauses. The syntax descriptions cover only basic syntax and features unique to the Oracle BI Server. For a more comprehensive description of SQL syntax, see a third-party reference book on SQL or a reference manual on SQL from your database vendors. For Oracle Database, see *Oracle Database SQL Language Reference*.

This section contains the following topics:

Basic Syntax for the SELECT Statement

Syntax for the `SELECT` statement is as follows:

```
SELECT [DISTINCT] select_list
FROM from_clause
[WHERE search_condition]
[GROUP BY column {, column}
     [HAVING search_condition]]
[ORDER BY column {, column}]
```

Where:

`select_list` is the list of columns specified in the request. See "SELECT List Syntax" for more information.

`FROM` *`from_clause`* is the list of tables in the request. Optionally includes certain join information for the request. See "FROM Clause Syntax" for more information.

`WHERE` *`search_condition`* specifies any combination of conditions to form a conditional test. A `WHERE` clause acts as a filter that lets you constrain a request to obtain results that answer a particular question. Together with the columns you select, filters determine what your results will contain. See "WHERE Clause Syntax" for more information.

`GROUP BY` *`column`* `{,` *`column`*`}` specifies a column (or alias) belonging to a table defined in the data source. See "GROUP BY Clause Syntax" for more information.

`HAVING` *`search_condition`* specifies any combination of conditions to form a conditional test. The syntax is identical to that for the `WHERE` clause.

`ORDER BY` *`column`* `{,` *`column`*`}` specifies the columns to order the results by. See "ORDER BY Clause Syntax" for more information.

Usage Notes

The Oracle BI Server treats the `SELECT` statement as a logical request. If aggregated data is requested in the `SELECT` statement, a `GROUP BY` clause is automatically assumed by the server. Any join conditions supplied in the query are ignored because the join conditions are all predefined in the Oracle BI repository.

The Oracle BI Server accepts the following SQL syntaxes for comments:

- /* */ C-style comments
- // Double slash for single-line comments
- # Number sign for single-line comments

Subquery Support

The Oracle BI Server supports certain subqueries, as well as UNION, UNION ALL, INTERSECT, and EXCEPT operations in logical requests. This functionality increases the range of business questions that can be answered, eases the formulation of queries, and provides some ability to query across multiple business models.

The Oracle BI Server supports the following subquery predicates in any conditional expression (for example, within WHERE, HAVING, or CASE statements):

```
IN, NOT IN
Any, >=Any, =Any, <Any, <=Any, <>Any
All, >=All, =All, <All,<=All, <>All
EXISTS, NOT EXISTS
```

In Answers, advanced users and developers can use the Advanced SQL Clauses fields in the Advanced tab of the Analysis editor to specify various SQL clauses, such as GROUP BY, HAVING, and DISTINCT, to include in the SQL queries that are sent to the Oracle BI Server. If an analysis contains hierarchical columns, selections, or groups, then certain Advanced SQL Clauses fields are not available.

SELECT List Syntax

The *select_list* lists the columns in the request. All columns need to be from a single business model. Table names can be included (as Table.Column), but are optional unless column names are not unique within a business model. If column names contain spaces, enclose column names in double quotes. The DISTINCT keyword does not need to be included, because the Oracle BI Server always does a distinct query. Columns that are being aggregated do not need to include the aggregation function (such as SUM), as aggregation rules are known to the server and aggregation is performed automatically.

Syntax

```
...
* |
  (column | expr) [[AS] alias]
  {, (column | expr) [[AS] alias] }
...
```

Where:

* Indicates all columns in the resultant table in the FROM clause.

column is a column (or alias) belonging to a table defined in the data source.

expr is any valid SQL expression.

Note: You cannot use * to select all columns from the Advanced tab of the Analysis editor in Answers. Instead, you must specify particular columns.

FROM Clause Syntax

The Oracle BI Server accepts any valid SQL FROM clause syntax. To simplify FROM clause creation, you can specify the name of a subject area instead of a list of tables. The Oracle BI Server determines the proper tables and the proper join specifications based on the columns the request asks for and the configuration of the Oracle BI repository.

WHERE Clause Syntax

The Oracle BI Server accepts any valid SQL WHERE clause syntax. There is no need to specify any join conditions in the WHERE clause, because the joins are all configured

within the Oracle BI repository. Any join conditions specified in the WHERE clause are ignored.

The Oracle BI Server also supports the following subquery predicates in any conditional expression (WHERE, HAVING or CASE statements):

```
IN, NOT IN
Any, >=Any, =Any, <Any, <=Any, <>Any
All, >=All, =All, <All,<=All, <>All
EXISTS, NOT EXISTS
```

GROUP BY Clause Syntax

With auto aggregation on the Oracle BI Server, there is no need to submit a GROUP BY clause. When no GROUP BY clause is specified, the GROUP BY specification defaults to all of the nonaggregation columns in the SELECT list. If you explicitly use aggregation functions in the select list, you can specify a GROUP BY clause with different columns and the Oracle BI Server computes the results based on the level specified in the GROUP BY clause.

See "Rules for Queries with Aggregate Functions" for additional details, as well as some examples of using the GROUP BY clause in queries against the Oracle BI Server.

ORDER BY Clause Syntax

The Oracle BI Server accepts any valid SQL ORDER BY clause syntax, including referencing columns by their order in the select list (such as ORDER BY 3, 1, 5).

In the ORDER BY clause, you can use the following syntax to alter the sort order for nulls in the query:

```
ORDER BY col1 NULLS LAST, ORDER BY col2 NULLS FIRST
```

For logical columns with sort order columns assigned to them, you can use the ORDER BY clause to disregard the sort order column and instead sort by the column's value. Note the following syntax:

```
ORDER BY { { <column_index> | <expr> } [ DISPLAY | SORTKEY ] [ ASC | DESC ] [NULLS
{ FIRST | LAST } ] }
```

Where:

DISPLAY sorts based on the order of the display value of the expression regardless of whether a sort column is assigned to the logical column. By default, Oracle BI Server assumes DISPLAY when a sort column *is not set* for the logical column.

SORTKEY sorts based on the logical column's assigned sort column. By default, Oracle BI Server assumes SORTKEY when a sort column *is set* for the logical column.

Syntax and Usage Notes for SELECT_PHYSICAL

The SELECT_PHYSICAL command provides the functionality to directly query objects in the Physical layer of the metadata repository, and to nest such a statement within a query against the Business Model and Mapping layer or the Presentation layer.

Though a SELECT_PHYSICAL query bypasses the Presentation layer and the Business Model and Mapping layer, the Oracle BI Server still performs parsing, interpretation, and query generation on a SELECT_PHYSICAL query before passing it to the database.

A SELECT_PHYSICAL command can contain any element allowed in standard Oracle BI Server SQL with the following constraints:

- The SELECT_PHYSICAL command does not explicitly reference structures in the repository Business Model and Mapping layer or the Presentation layer
- The SELECT_PHYSICAL command does not require implicit logical transformation
- The SELECT_PHYSICAL command cannot contain certain aggregate functions - see "Aggregate Functions Not Supported in SELECT_PHYSICAL Queries" for details

> **Note:** SELECT_PHYSICAL statements are not cached.

You can set up an ODBC connection to the Oracle BI Server to be a dedicated physical connection over which all SELECT queries are treated as SELECT_PHYSICAL queries. To do this, select **Route Requests To Physical Layer** in the ODBC data source for the Oracle BI Server. See "Integrating Other Clients with Oracle Business Intelligence" in *Oracle Fusion Middleware Integrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

SELECT_PHYSICAL statements are logged as Physical Request entries.

The topics in this section are the following:

- Syntax for the SELECT_PHYSICAL Statement
- Aggregate Functions Not Supported in SELECT_PHYSICAL Queries
- Queries Supported by SELECT_PHYSICAL
- Using the NATURAL_JOIN Keyword
- Special Usages of SELECT_PHYSICAL

Syntax for the SELECT_PHYSICAL Statement

Basic syntax for SELECT_PHYSICAL queries is equivalent to "Basic Syntax for the SELECT Statement" with the term SELECT_PHYSICAL replacing the word SELECT, namely:

```
SELECT_PHYSICAL [DISTINCT] select_list
FROM from_clause
[WHERE search_condition]
[GROUP BY column {, column}
     [HAVING search_condition]]
[ORDER BY column {, column}]
```

> **Notes:** The SELECT_PHYSICAL statement is close to the standard ANSI SQL SELECT statement. For example, you cannot omit the GROUP BY clause nor, where relevant, the HAVING clause in a SELECT_PHYSICAL aggregate query.

In SELECT_PHYSICAL queries, you must fully qualify the table names in the FROM list. Each fully qualified table name must match a table name in the physical layer of the repository.

A fully qualified table name consists of up to four components, database name, catalog name, schema name, and table name. Each component is surrounded by double quotes (") with a period (.) separator between components. For example, "SQL_DB"."My_Catalog"."My_Schema"."Customers" for a SQL Server table, and "FoodMart"..."Sales" for a cube table.

Refer to the corresponding topics in "Basic Syntax for the SELECT Statement" for more information about the different clauses and sub-clauses of the SELECT_PHYSICAL command.

Aggregate Functions Not Supported in SELECT_PHYSICAL Queries

The following aggregate functions are not supported in SELECT_PHYSICAL queries:

- AGO
- BOTTOMN
- FILTER
- FIRST
- LAST
- RCOUNT
- RMAX
- RMIN
- RSUM
- TODATE
- TOPN

Queries Supported by SELECT_PHYSICAL

The Oracle BI Server supports the use of SELECT_PHYSICAL for the following types of logical query:

- **Standard Non-Aggregate Queries**

  Standard non-aggregate SELECT_PHYSICAL commands follow the same rules as standard non-aggregate SELECT commands. They can also include scalar functions, such as String, Math, and Calendar Date/Time functions. For example:

```
SELECT_PHYSICAL productid, categoryid
FROM "My_DB"."My_Schema"."products"
WHERE categoryid > 5;

SELECT_PHYSICAL LEFT(productname,10)
FROM "My_DB"."My_Schema"."products"
WHERE productname is not null;
```

- **Queries with Aggregate Functions**

  In general, all aggregate functions supported in SELECT queries are also supported in SELECT_PHYSICAL queries. See "Aggregate Functions Not Supported in SELECT_PHYSICAL Queries" for a list of the exceptions to this rule.

  For aggregates supported in SELECT_PHYSICAL commands, each aggregate must have an explicitly specified aggregation level, using the GROUP BY clause or the BY clause. For example:

```
SELECT_PHYSICAL employeeid, SUM(quantity by)
FROM "My_DB"."My_Schema"."employees";

SELECT_PHYSICAL employeeid, SUM(quantity)
FROM "My_DB"."My_Schema"."employees"
GROUP BY employeeid
HAVING SUM(quantity) > 100;
```

- **Subqueries**

  The Oracle BI Server supports the following types of query:

  - Queries where both the parent query and the subquery use SELECT_PHYSICAL
  - Parent query uses SELECT and subquery uses SELECT_PHYSICAL

  Subqueries are supported on both filters and on projections embedded in a Case statement.

  For example:

```
SELECT_PHYSICAL *
FROM "My_DB"."My_Schema"."products"
WHERE supplierid IN
 (SELECT_PHYSICAL supplierid
  FROM "My_DB"."My_Schema"."suppliers");

SELECT productid
FROM snowflakesales.product
WHERE categoryid IN
 (SELECT_PHYSICAL categoryid
  FROM "My_DB"."My_Schema"."categories");

SELECT CASE WHEN b.categoryid IN
 (SELECT_PHYSICAL a.categoryid
  FROM "My_DB"."My_Schema"."products" a)
 THEN b.categoryid END
FROM categories b;
```

- **Queries with Derived Tables**

  Both SELECT and SELECT_PHYSICAL queries can have derived tables in their FROM clause. The tables can be derived using either SELECT or SELECT_PHYSICAL. For example:

```
SELECT_PHYSICAL COUNT(DISTINCT t.rto)
FROM
 (SELECT_PHYSICAL employeeid AS id, reportsto AS rto
  FROM "My_DB"."My_Schema"."employees") t;

SELECT productid, categoryid
FROM
 (SELECT_PHYSICAL productid, categoryid
  FROM "My_DB"."My_Schema"."products" a
  LEFT OUTER JOIN "My_DB"."My_Schema"."categories" b
  ON a.categoryid = b.categoryid);

SELECT y.cid, sum(x.qty)
FROM
 (SELECT productid pid, categoryid cid, qtysold qty
  FROM sales.product) x
 RIGHT OUTER JOIN
 (SELECT_PHYSICAL CASE categoryid WHEN 1 THEN null ELSE categoryid END cid
  FROM "My_DB"."My_Schema"."categories") y
 ON x.cid = y.cid
 GROUP BY y.cid;
```

- **Cross-Database Queries**

  You can use SELECT_PHYSICAL to join tables in different databases. For example:

```
SELECT_PHYSICAL a.productid, b.categoryid
FROM "My_DB"."My_Schema"."products" a
FULL OUTER JOIN
"My_DB2"."My_Schema"."categories" b
ON a.categoryid = b.categoryid
```

Using the NATURAL_JOIN Keyword

SELECT_PHYSICAL queries support the NATURAL JOIN syntax, which enables you to use predefined join expressions. For ADF data sources, the ViewLink in ADF becomes active. The NATURAL JOIN join type, however, is not exposed for use in Logical Table Sources (for example, LEFT OUTER JOIN).

You can only use the NATURAL JOIN keyword in SELECT_PHYSICAL queries. The NATURAL JOIN behavior in Oracle Business Intelligence is different from the ANSI NATURAL JOIN. The following examples illustrate how joins are executed with and without the NATURAL JOIN syntax:

```
SELECT PHYSICAL *
FROM A, B;
```

In this example, no join is executed between A and B (even if one is defined in the metadata).

```
SELECT_PHYSICAL *
FROM A NATURAL JOIN B;
```

In this example, the physical join between A and B is executed. For ADF data sources, the join expression defined by the underlying ViewLink is used.

```
SELECT_PHYSICAL *
FROM C, A NATURAL JOIN B;
```

In this example, even if C is joined to A in the metadata, only the A-B join is active. The C-A join is not used.

Special Usages of SELECT_PHYSICAL

You can use session variables and the INDEXCOL function in a SELECT_PHYSICAL command, as in the following examples:

```
SELECT_PHYSICAL VALUEOF(NQ_SESSION.REGION)
FROM "My_DB"."My_Schema"."products";

SELECT_PHYSICAL INDEXCOL(VALUEOF(NQ_SESSION.INDEXCOLINDEX), productid, categoryid)
FROM "My_DB"."My_Schema"."products";
```

Limiting and Offsetting Rows Returned

You can use the FETCH and OFFSET clauses to constrain the number of rows returned by the SELECT statement and to skip a specified number of rows from the beginning of the result set. Both clauses are optional and can be used together, or independently. The fetch and offset clauses are part of the SELECT statement and are placed at the end.

These clauses are useful for situations where you have a large result set (such as with a large dimension), and you want to present, for example, the first 100 rows to the user. The Oracle BI Server stops processing when the limit is reached, improving overall performance and conserving resources. In addition, the limit is pushed to the back-end database in many cases so that the database can optimize the query.

Technically, both clauses can be used without an ORDER BY clause, but the results would be non-deterministic. Because of this, both clauses should always be used with ORDER BY.

If OFFSET is not specified, the default value is 0, which means that results are returned starting from the first row. If FETCH is not specified, it means that there is no limitation on the number rows returned.

Both clauses are evaluated after the WHERE clause, aggregation, HAVING clause, window analytic function, and ORDER BY clause. Both clauses can be used with SELECT_PHYSICAL in addition to SELECT.

Syntax for OFFSET Clause

```
OFFSET n ROW[S]
```

n is the number of rows you want to skip from the beginning of the result set. Note that *n* must be greater than zero.

Syntax for FETCH Clause

```
FETCH FIRST | NEXT n ROW[S] ONLY
```

n is the number of rows you want to retrieve. Note that *n* must be greater than zero.

Typically, FIRST is used when the limit clause is used independently of the offset clause, while NEXT is used when the limit clause is used in conjunction with the offset clause.

Example

```
SELECT employeeid, firstname, revenue
FROM sales.employee
ORDER BY revenue desc
OFFSET 2 ROWS
FETCH NEXT 4 ROWS ONLY
```

The following table lists the entire result set without the OFFSET and FETCH clauses. When the OFFSET and FETCH clauses are included, only the rows shown in bold are returned.

Employeeid	FirstName	Revenue
4	Margaret	250187.45
3	Janet	213051.30
1	**Nancy**	**202143.71**
2	**Andrew**	**202143.71**
7	**Robert**	**177749.26**
8	**Laura**	**141295.99**
9	*Anne*	133301.03
6	*Michael*	82964.00
5	*Steven*	78198.10

Limitations of the FETCH and OFFSET Clauses

Because ORDER BY clauses are ignored in UNION ALL set-operator blocks, using these clauses in such queries would be non-deterministic. Do not use FETCH and OFFSET with these queries.

Rules for Queries with Aggregate Functions

The Oracle BI Server simplifies the SQL statements needed to craft aggregate queries. This section outlines the rules that the Oracle BI Server follows for whether a query contains a GROUP BY clause and, if a GROUP BY clause is specified, what results you should expect from the query. The rules outlined in this section apply to all aggregates used in SQL statements (SUM, AVG, MIN, MAX, COUNT(*), and COUNT).

Computing Aggregates of Baseline Columns

A baseline column is a column that has no aggregation rule defined in the Aggregation tab of the Logical Column dialog in the repository. Baseline columns map to nonaggregated data at the level of granularity of the logical table to which they belong. If you perform aggregation (SUM, AVG, MIN, MAX, or COUNT) on a baseline column through a SQL request, the Oracle BI Server calculates the aggregation at the level based on the following rules:

- If there is no GROUP BY clause specified, the level of aggregation is grouped by all of the nonaggregate columns in the SELECT list.
- If there is a GROUP BY clause specified, the level of aggregation is based on the columns specified in the GROUP BY clause.

For example, consider the following query, where the column revenue is defined in the repository as a baseline column (no aggregation rules specified in the Logical Column > Aggregation tab):

```
SELECT year, product, SUM(revenue)
FROM time, products, facts
```

The results appear in the following list by year, products, and then sum of revenue.

YEAR	PRODUCT	SUM(REVENUE)
1998	Coke	500
1998	Pepsi	600
1999	Coke	600
1999	Pepsi	550
2000	Coke	800
2000	Pepsi	600

This query returns results grouped by year and product, or in other words, it returns one row for each product and year combination. The sum calculated for each row is the sum of all the sales for that product in that year. It is logically the same query as the following:

```
SELECT year, product, SUM(revenue)
FROM time, products, facts
GROUP BY year, product
```

If you change the GROUP BY clause to only group by year, then the sum calculated is the sum of all products for the year, as follows:

```
SELECT year, product, SUM(revenue)
FROM time, products, facts
GROUP BY year
```

The results appear in the following list by year, products, and then sum of revenue:

YEAR	PRODUCT	SUM(REVENUE)
1998	Coke	1100
1998	Pepsi	1100
1999	Coke	1150
1999	Pepsi	1150
2000	Coke	1400
2000	Pepsi	1400

If you add a column to the query requesting the COUNT of revenue, the Oracle BI Server calculates the number of records used to calculate the results for each group. In this case, it is a year, as shown in the following example:

```
SELECT year, product, SUM(revenue), COUNT(revenue)
FROM time, products, facts
GROUP BY year
```

The results appear in the following list by year, products, sum of revenue, and then revenue count:

YEAR	PRODUCT	SUM(REVENUE)	COUNT(REVENUE)
1998	Coke	1100	6000
1998	Pepsi	1100	6000
1999	Coke	1150	6500
1999	Pepsi	1150	6500
2000	Coke	1400	8000
2000	Pepsi	1400	8000

Computing Aggregates of Measure Columns

A measure column is a column that has a default aggregation rule defined in the Aggregation tab of the Logical Column dialog in the repository. Measure columns always calculate the aggregation with which they are defined. If you perform explicit aggregation (SUM, AVG, MIN, MAX, or COUNT) on a measure column through a SQL request, you are actually asking for an aggregate of an aggregate. For these nested aggregates, the Oracle BI Server calculates the aggregation based on the following rules:

- A request for a measure column without an aggregate function defined in a SQL statement is always grouped at the level of the nonaggregate columns in the SELECT list, regardless of whether the query specifies a GROUP BY clause.
- If there is no GROUP BY clause specified, the nested aggregate is a grand total of each group determined by all of the nonaggregate columns in the SELECT list.

- If there is a GROUP BY clause specified, the nested aggregation calculates the total for each group as specified in the GROUP BY clause.

For example, consider the following query, where the column SumOfRevenue is defined in the repository as a measure column with a default aggregation rule of SUM (SUM aggregation rule specified in the Aggregation tab of the Logical Column dialog):

```
SELECT year, product, SumOfRevenue, SUM(SumOfRevenue)
FROM time, products, facts
```

The following query results are grouped by year and product, or in other words, it returns one row for each product and year combination. The sum calculated for each row in the SumOfRevenue column is the sum of all the sales for that product in that year because the measure column is always at the level defined by the nonaggregation columns in the query.

YEAR	PRODUCT	SUMofREVENUE	SUM(SUMofREVENUE)
1998	Coke	500	3650
1998	Pepsi	600	3650
1999	Coke	600	3650
1999	Pepsi	550	3650
2000	Coke	800	3650
2000	Pepsi	600	3650

If you set the GROUP BY clause to only group by year, then the sum calculated in the SumOfRevenue column is the sum of each product for the year, and the sum calculated in the SUM(SumOfRevenue) column is total sales of all products for the given year. The following is the query:

```
SELECT year, product, SumOfRevenue, SUM(SumOfRevenue)
FROM time, products, facts
GROUP BY year
```

In the following result set, the sum calculated for each row in the SumOfRevenue column is the sum of all the sales for that product in that year because the measure column is always at the level defined by the nonaggregation columns in the query. The SUM(SumOfRevenue) is the same for each row corresponding to a given year, and that sum represents the total sales for that year. In this case, it is the sales of Coke plus the sales of Pepsi.

YEAR	PRODUCT	SUMofREVENUE	SUM(SUMofREVENUE)
1998	Coke	500	1100
1998	Pepsi	600	1100
1999	Coke	600	1150
1999	Pepsi	550	1150
2000	Coke	800	1400
2000	Pepsi	600	1400

Display Function Reset Behavior

A display function is a function that operates on the result set of a query. The display functions the Oracle BI Server supports (RANK, TOPN, BOTTOMN, PERCENTILE, NTILE, MAVG, MEDIAN, and varieties of standard deviation) are specified in the SELECT list of a SQL query. Queries that use display functions conform to the following rules:

- If no GROUP BY clause is specified, the display function operates across the entire result set, or in other words, the grouping level for the display function matches that of the query.
- If there is a GROUP BY clause specified, the display function resets its values for each group as specified in the GROUP BY clause.

For example, in the following query, SumOfRevenue is defined as a measure column with the default aggregation rule of SUM:

```
SELECT year, product, SumOfRevenue, RANK(SumOfRevenue)
FROM time, products, facts
```

In the following query result set, there is no GROUP BY clause specified, so the rank is calculated across the entire result set:

YEAR	PRODUCT	SUMofREVENUE	RANK(SUMofREVENUE)
1998	Coke	500	6
1998	Pepsi	600	2
1999	Coke	600	2
1999	Pepsi	550	5
2000	Coke	800	1
2000	Pepsi	600	2

If you change the GROUP BY clause to group by year and product, then the rank is reset for each year, as follows:

```
SELECT year, product, SUM(revenue), RANK(sum(revenue) by year)
FROM time, products, facts
GROUP BY year, product
```

In the following result set, the rank is reset each time the year changes, and because there are two rows for each year, the value of the rank is always 1 or 2:

YEAR	PRODUCT	SUMofREVENUE	RANK(SUM(REVENUE) by year)
1998	Coke	500	2
1998	Pepsi	600	1
1999	Coke	600	1
1999	Pepsi	550	2
2000	Coke	800	1
2000	Pepsi	600	2

Alternative Syntax

When using an aggregate function, you can calculate a specified level of aggregation using BY within the aggregate function. If you do this, you do not need a GROUP BY clause.

For example, the following query returns the column year_revenue that displays revenue aggregated by year:

```
SELECT year, product, revenue, SUM(revenue BY year) as year_revenue
FROM softdrinks
```

The same syntax can be used with display functions. The following query calculates overall rank of revenue for each product for each year (each row in the entire result set), and also the rank of each product's revenue within each year:

```
SELECT year, product, revenue, rank(revenue), RANK(revenue by year)
FROM softdrinks ORDER BY 1, 5
```

Using FILTER to Compute a Conditional Aggregate

In SQL query language, traditional aggregates, such as SUM, COUNT, MIN, and MAX are evaluated on a group of tuples (an ordered list of objects, each of a specified type), determined by the GROUP BY clause. All the aggregates specified in the SELECT clause of a query are evaluated over the same subset of tuples. Conditional aggregates extend SQL by restricting their input using a predicate.

FILTER is an operator that restricts the set of rows used to compute its aggregate argument to rows that satisfy the USING condition. The FILTER operator is a Logical SQL construct. It may be used in logical queries referring to the metadata, or in logical columns that use existing logical columns as the source.

Syntax

Conditional aggregates are only notational concepts and they do not represent executable operators. Conditional aggregates are expressed in the form of a function as shown in the following statement:

```
FILTER(measure_expr USING boolean_expr)
```

Where:

measure_expr is an expression that contains at least one measure. The following is a list of examples:

- The expression Sales + 1 is allowed if Sales is a measure.
- The expression productid is not allowed if productid is a scalar attribute.

boolean_expr is a boolean expression (evaluates to TRUE or FALSE) that does not contain any measures. This expression may not contain any nested queries.

Example

The following is a simple example of the FILTER function:

```
SELECT year,
FILTER(sales USING product = 'coke'),
FILTER(sales USING product = 'pepsi')
FROM logBeverages
```

After navigation, this query is executed as follows:

```
SELECT year,
```

```
SUM(CASE WHEN product = 'coke' THEN sales),
SUM(CASE WHEN product = 'pepsi' THEN sales)
FROM physBeverages
WHERE product = 'coke' OR product = 'pepsi'
GROUP BY year
```

Error Handling

In the example `FILTER(x USING y)`, error messages are returned in the following situations:

- The *y* expression is not a boolean expression.
- The *y* expression contains measures.
- `FILTER` is used in outer query block.
- Explicit aggregates are used in the *x* (measure) expression. For example, `FILTER(COUNT(product), C)`.

Operators

There are two types of operators: SQL logical operators, and mathematical operators.

SQL Logical Operators

The following SQL logical operators are used to specify comparisons between expressions.

- **Between:** Used to determine boundaries for a condition. Each boundary is an expression, and the bounds do not include the boundary limits, as in less than and greater than (as opposed to less than or equal to and greater than or equal to). `BETWEEN` can be preceded with `NOT` to negate the condition.
- **In:** Specifies a comparison of a column value with a set of values.
- **Is Null:** Specifies a comparison of a column value with the null value.
- **Like:** Specifies a comparison to a literal value. Often used with wildcard characters to indicate any character string match of zero or more characters (%) or a any single character match (_).

Mathematical Operators

Mathematical operators are used to combine expression elements to make certain types of comparisons in an expression.

Table C–1 lists operators and describes their use in an expression.

Table C–1 Operators

Operator	Description
+	Plus sign for addition.
-	Minus sign for subtraction.
*	Multiply sign for multiplication.
/	Divide by sign for division.
||	Character string concatenation.
(	Open parenthesis.
)	Closed parenthesis.

Table C–1 (Cont.) Operators

Operator	Description
>	Greater than sign, indicating values higher than the comparison.
<	Less than sign, indicating values lower than the comparison.
=	Equal sign, indicating the same value.
<=	Less than or equal to sign, indicating values the same or lower than the comparison.
>=	Greater than or equal to sign, indicating values the same or higher than the comparison.
<>	Not equal to, indicating values higher or lower, but different.
AND	AND connective, indicating intersection with one or more conditions to form a compound condition.
OR	OR connective, indicating the union with one or more conditions to form a compound condition.
NOT	NOT connective, indicating a condition is not met.
,	Comma, used to separate elements in a list.

Conditional Expressions

Expressions are building blocks for creating conditional expressions that convert a value from one form to another. Expressions include:

- CASE (Switch)
- CASE (If)

CASE (Switch)

This form of the CASE statement is also referred to as the CASE(Lookup) form. The value of *expr1* is examined, then the WHEN expressions. If *expr1* matches any WHEN expression, it assigns the value in the corresponding THEN expression.

If none of the WHEN expressions match, it assigns the default value specified in the ELSE expression. If no ELSE expression is specified, the system automatically adds an ELSE NULL.

If *expr1* matches an expression in multiple WHEN clauses, only the expression following the first match is assigned.

Note: In a CASE statement, AND has precedence over OR.

Syntax

```
CASE expr1
     WHEN expr2 THEN expr3
     {WHEN expr... THEN expr...}
     ELSE expr
END
```

Where:

CASE starts the CASE statement. Must be followed by an expression and one or more WHEN and THEN statements, an optional ELSE statement, and the END keyword.

WHEN specifies the condition to be satisfied.

THEN specifies the value to assign if the corresponding WHEN expression is satisfied.

ELSE specifies the value to assign if none of the WHEN conditions are satisfied. If omitted, ELSE NULL is assumed.

END ends the CASE statement.

Example

```
CASE Score-par
  WHEN -5 THEN 'Birdie on Par 6'
  WHEN -4 THEN 'Must be Tiger'
  WHEN -3 THEN 'Three under par'
  WHEN -2 THEN 'Two under par'
  WHEN -1 THEN 'Birdie'
  WHEN 0 THEN 'Par'
  WHEN 1 THEN 'Bogey'
  WHEN 2 THEN 'Double Bogey'
  ELSE 'Triple Bogey or Worse'
END
```

In this example, the WHEN statements must reflect a strict equality. For example, a WHEN condition of WHEN < 0 THEN 'Under Par' is illegal because comparison operators are not allowed.

CASE (If)

This form of the CASE statement evaluates each WHEN condition and if satisfied, assigns the value in the corresponding THEN expression.

If none of the WHEN conditions are satisfied, it assigns the default value specified in the ELSE expression. If no ELSE expression is specified, the system automatically adds an ELSE NULL.

> **Note:** In a CASE statement, AND has precedence over OR.

Syntax

```
CASE
     WHEN request_condition1 THEN expr1
     {WHEN request_condition2 THEN expr2}
     {WHEN request_condition... THEN expr...}
     ELSE expr
END
```

Where:

CASE starts the CASE statement. Must be followed by one or more WHEN and THEN statements, an optional ELSE statement, and the END keyword.

WHEN specifies the condition to be satisfied.

THEN specifies the value to assign if the corresponding WHEN expression is satisfied.

ELSE specifies the value to assign if none of the WHEN conditions are satisfied. If omitted, ELSE NULL is assumed.

END ends the CASE statement.

Example

```
CASE
  WHEN score-par < 0 THEN 'Under Par'
  WHEN score-par = 0 THEN 'Par'
  WHEN score-par = 1 THEN 'Bogie'
  WHEN score-par = 2 THEN 'Double Bogey'
  ELSE 'Triple Bogey or Worse'
END
```

Unlike the Switch form of the CASE statement, the WHEN statements in the If form allow comparison operators. For example, a WHEN condition of WHEN < 0 THEN 'Under Par' is legal.

Expressing Literals

A literal is a nonnull value corresponding to a given data type. Literals are typically constant values, or in other words, they are values that are taken as they are. A literal value must comply with the data type that it represents.

SQL provides mechanisms for expressing literals in SQL statements. This following topics describe how to express each type of literal in SQL:

- Character Literals
- Datetime Literals
- Numeric Literals

Character Literals

A character literal represents a value of CHARACTER or VARCHAR data type. To express a character literal, enclose the character string in single quotes ('). The number of characters enclosed between the single quotes implies the length of the literal.

Examples

```
'Oracle BI Server'

'abc123'
```

Datetime Literals

The SQL 92 standard defines three kinds of 'typed' datetime literals, in the following formats:

```
DATE 'yyyy-mm-dd'
TIME 'hh:mm:ss'
TIMESTAMP 'yyyy-mm-dd hh:mm:ss'
```

To express a typed datetime literal, use the keywords DATE, TIME, or TIMESTAMP followed by a datetime string enclosed in single quotation marks, as in the preceding example. Two digits are required for all nonyear components even if the value is a single digit.

These formats are fixed and are not affected by the format specified in the NQSConfig.INI file for the parameters DATE_DISPLAY_FORMAT, TIME_DISPLAY_FORMAT, or DATE_TIME_DISPLAY_FORMAT.

Examples

```
DATE '2000-08-15'
TIME '11:55:25'
TIMESTAMP '1999-03-15 11:55:25'
```

Numeric Literals

A numeric literal represents a value of a numeric data type (such as `INTEGER`, `DECIMAL`, or `FLOAT`). To express a numeric literal, type the number as part of a SQL statement.

Do not surround numeric literals with single quotes. Doing so expresses the literal as a character literal.

Note: When `ENABLE_NUMERIC_DATA_TYPE` is set to `YES` in NQSConfig.INI, all decimal literals (or integer literals that are too large to fit in the INT data type) are parsed internally within the Oracle BI Server as NUMERIC.

When treating literals as NUMERIC, be aware of the Oracle standard double promotion rules, including the following:

DOUBLE/NUMBER = DOUBLE , DOUBLE * NUMBER = DOUBLE

Because the parsing of numeric literals happens very early in the query processing before the actual data source is known, internally, the Oracle BI Server treats decimal numbers as NUMERIC if `ENABLE_NUMERIC_DATA_TYPE` is set to `YES`, regardless of data source type.

When NUMERIC is enabled and the Oracle BI Server executes an expression internally involving decimal literals, the server treats the literals as NUMERIC even if the back-end data source does not support the NUMERIC data type. However, the type promotion rules still apply. For example, if the Oracle BI Server retrieves the data from a data source as DOUBLE and combines that with a NUMERIC literal during internal execution, the final result is still be converted to DOUBLE.

Numeric literals include:

- Integer Literals
- Decimal Literals
- Floating Point Literals

Integer Literals To express an integer constant as a literal, specify the integer as part of a SQL statement (for example, in the `SELECT` list). Precede the integer with a plus sign (+) to indicate the integer is positive, or a minus sign (-) to indicate the integer is negative. Unsigned integers are assumed to be positive.

Examples

```
234
+2
567934
```

Decimal Literals To express a decimal literal, specify a decimal number. Precede the number with a plus sign (+) to indicate the number is positive, or a minus sign (-) to indicate the number is negative. Unsigned numbers are assumed to be positive.

Examples

```
1.223
-22.456
+33.456789
```

Floating Point Literals To express floating point numbers as literal constants, enter a decimal literal followed by the letter E (either uppercase or lowercase), followed by the plus sign (+) to indicate a positive exponent, or the minus sign (-) to indicate a negative exponent. No spaces are allowed between the integer, the letter E, and the sign of the exponent.

Examples

```
333.456E-
1.23e+
```

Calculated Members

A calculated member is a user-defined dimension member whose measure values are calculated at run time.

You define a calculated member within a dimension through a formula that references other members of the same dimension. If a dimension has multiple hierarchies, all members referenced in the formula must belong to one hierarchy.

Within a calculated member, the members do not have to be at the same level in the hierarchy. For example, in a Geography hierarchy, you can create a calculated member to enable you to add together measure values for the Country member France and the City member Budapest.

The three standard components of a calculated member are:

- The presentation hierarchy on which the calculated member is based (for example, "Geography")
- The name to identify the calculated member, and to distinguish it from other members in the dimension (for example, 'My Locations')
- The formula used to calculate the calculated member, containing one or more Member clauses (for example, Member ("Geography"."Country".'France') + Member ("Geography"."City".'Budapest'))

This section contains the following topics:

- CALCULATEDMEMBER Syntax
- Rules for the CALCULATEDMEMBER Expression
- Using Solve Order to Control Formula Evaluation Sequence
- Examples of Calculated Members in Queries

CALCULATEDMEMBER Syntax

```
CALCULATEDMEMBER(presentation_hierarchy, member_identifier, calculated_member_
formula [, solve_order])
```

Where:

presentation_hierarchy identifies the fully qualified presentation hierarchy in the presentation layer on which the calculated member is based, as follows:

```
"subject_area"."presentation_table"."presentation_hierarchy"
```

Note: When qualifying presentation hierarchies and presentation hierarchy levels in both the CALCULATEDMEMBER expression and the Member clause within the *calculated_member_formula* parameter, the following rule applies:

- You must specify the qualification term ("subject_area".) if there are multiple presentation tables or presentation hierarchies with the same name in different subject areas, otherwise you can omit the term.

member_identifier is the string or numeric literal that identifies the calculated member. The type of literal depends on the data type of the dimension level-keys.

calculated_member_formula consists of one or more examples of a "member clause" connected by the standard arithmetic operators + - * / ( ). The syntax of the member clause depends on whether the presentation hierarchy is level-based or parent-child. See "Syntax for the Member Clause in Level-Based Hierarchies" and "Syntax for the Member Clause in Parent-Child Hierarchies" for details.

solve_order (optional) is a positive integer, used to determine the order of evaluation when there are calculated members from different dimensions in the same query. See "Using Solve Order to Control Formula Evaluation Sequence" for details.

Syntax for the Member Clause in Level-Based Hierarchies

MEMBER(*presentation_hierarchy_level*, *member_value*)

Where:

presentation_hierarchy_level identifies the fully qualified hierarchy level in the *presentation_hierarchy*, as follows:

"subject_area"."presentation_table"."presentation_hierarchy"."presentation_level"

member_value is the string or numeric literal that identifies the member in the *presentation_hierarchy_level*.

Syntax for the Member Clause in Parent-Child Hierarchies

MEMBER(*presentation_hierarchy*, *member_value*)

Where:

presentation_hierarchy identifies the fully qualified presentation hierarchy in the presentation layer on which the calculated member is based, as follows:

"subject_area"."presentation_table"."presentation_hierarchy"

member_value is the string or numeric literal that identifies the member in the *presentation_hierarchy*.

Rules for the CALCULATEDMEMBER Expression

The rules for calculated members relate to the CALCULATEDMEMBER expression itself and the use of the CALCULATEDMEMBER expression in queries.

- All level references in a given CALCULATEDMEMBER expression must belong to the same dimension hierarchy.
- CALCULATEDMEMBER expressions may only appear in the SELECT list of a query.

- Queries containing a CALCULATEDMEMBER expression must include at least one measure column in the SELECT list.
- Only one CALCULATEDMEMBER expression is allowed for each dimension for each SELECT list of a query block. However, CALCULATEDMEMBER expressions based on other dimensions may exist in the same query.
- You cannot include any other column from a dimension on which a calculated member is based in the following components of a query block:
  - SELECT list
  - WHERE clause
  - HAVING clause

  However, you may reference columns from the calculated member dimension in subqueries.
- Columns from other dimensions may be referenced in the same query block, as long as there are no CALCULATEDMEMBER expressions on those dimensions.

Using Solve Order to Control Formula Evaluation Sequence

By default, when the CALCULATEDMEMBER expression does not contain a solve order, the calculated members are evaluated in the order in which they appear in the SELECT list.

When there are calculated members from different dimensions in the same query block, the order in which the Oracle BI Server evaluates the calculated members may be significant.

Example C–1 illustrates how the wrong solve order can lead to incorrect results.

Example C–1 Using Solve Order

Assume you have the following account and time data:

Kwik Grains	Profit	Sales
2007 Q3	300	1000
2007 Q4	600	1500

You want to calculate the percentage profit (Profit / Sales * 100) for each time period and the totals for the two quarters.

If the solve order for your calculations is the following:

1. 'Profit%' = 'Profit'/'Sales' * 100
2. '2007 Second Half' = '2007 Q3' + '2007 Q4'

then the percentage profit for '2007 Second Half' is calculated *incorrectly* by adding the 2007 Q3 profit to the 2007 Q4 profit:

- (300/1000) + (600/1500) = 30% + 40% = 70%

with the following results:

Kwik Grains	Profit	Sales	Profit%
2007 Q3	300	1000	30
2007 Q4	600	1500	40

Kwik Grains	Profit	Sales	Profit%
2007 Second Half	900	2500	70 ***(incorrect result)***

If the solve order for your calculations is the following:

1. '2007 Second Half' = '2007 Q3' + '2007 Q4'
2. 'Profit%' = 'Profit'/'Sales' * 100

then the percentage profit for '2007 Second Half' is calculated correctly by adding the 2007 Q3 and 2007 Q4 profits and sales first, then dividing the total Profit by the total Sales:

- (300+600) / (1000+1500) = 900/2500 = 36%

with the following results:

Kwik Grains	Profit	Sales	Profit%
2007 Q3	300	1000	30
2007 Q4	600	1500	40
2007 Second Half	900	2500	36 ***(correct result)***

See Example C–3, "Using Calculated Members from Different Dimensions" for an example of a query that explicitly specifies the solve order.

Examples of Calculated Members in Queries

The examples in this section show the use of calculated members in queries, and the base data on which the calculations are performed.

Example C–2 Single Calculated Member Query

This example shows two queries, each with corresponding results.

The first query contains a calculated member.

```
SELECT CALCULATEDMEMBER(product."Product - Region",'USA - LA - Tokyo',
           MEMBER(product."Product - Region"."Country", 'USA')
         - MEMBER(product."Product - Region"."Region", 'LA')
         - MEMBER(product."Product - Region"."City", 'Tokyo')
           ) MyRegion,
       sales.Revenue Revenue, sales.QtySold QtySold
FROM product, sales;
```

Result:

```
MYREGION            REVENUE     QTYSOLD
USA - LA - Tokyo    61959.00    3959
```

The second query verifies the results of the first query. It shows the base data on which the calculation in the first query is performed.

```
SELECT * from SupplierCity where Country in ('USA', 'Japan');
```

Result:

```
CITY            REGION      COUNTRY         REVENUE         QTYSOLD
Boston          MA          USA             28146.40        2084
Osaka                       Japan           15678.30        1417
```

```
New Orleans     LA          USA           33351.95       1735
Ann Arbor       MI          USA           43569.00       1436
Tokyo                       Japan         33533.20       1134
Bend            OR          USA           23776.80       1573
```

Example C–3 Using Calculated Members from Different Dimensions

The requirement in this example is to determine the percentage increase over time in Revenue and Quantity Sold for US and Canada combined.

To achieve the correct results, the solve order is significant. You must first add Revenue and Quantity Sold for the two countries across the time periods, then perform the percentage calculation. See "Using Solve Order to Control Formula Evaluation Sequence" for more information about solve order significance.

This example shows two queries, each with corresponding results.

The first query contains the calculated members from the two dimensions Product - Region and Time, with the "addition" formula calculated first, then the "percentage" formula.

```
SELECT CALCULATEDMEMBER(product."Product - Region", 'North America',
           MEMBER(product."Product - Region"."Country", 'USA')
         + MEMBER(product."Product - Region"."Country", 'Canada'), 1
           ) MyRegion,
       CALCULATEDMEMBER(day."Time", 'Percentage Increase',
         ( MEMBER(day."Time"."Year", 1996)
         - MEMBER(day."Time"."Year", 1995) ) * 100
         / MEMBER(day."Time"."Year", 1995), 2
           ) MyTime,
       sales.Revenue RevenuePC,
       sales.QtySold QtySoldPC
FROM product, sales, day;
```

Result:

```
MYREGION         MYTIME                 REVENUEPC      QTYSOLDPC
North America    Percentage Increase    16             35
```

Note that in the preceding query, the sequence of the calculated members in the SELECT list is sufficient for correct results, even without the explicit solve orders. The solve orders are included for completeness.

The second query verifies the results of the first query. It shows the base data on which the calculations in the first query are performed.

```
SELECT CALCULATEDMEMBER(product."Product - Region", 'North America',
           MEMBER(product."Product - Region"."Country", 'USA')
         + MEMBER(product."Product - Region"."Country", 'Canada')
           ) MyRegion,
       year as Year, sales.Revenue Revenue, sales.QtySold QtySold
FROM product, sales, day;
```

Result:

```
MYREGION         YEAR      REVENUE       QTYSOLD
North America    1996      101702.75     4918
North America    1995      87265.10      3638
North America    1994      30776.00      1616
```

Variables

You can include and set variables in SQL statements. To do this, include the variable at the beginning of the SQL statement.

Syntax

```
SET VARIABLE variable_name = variable_value; SELECT_statement
```

If you are executing a query from the `nqcmd` utility, use a colon as a delimiter. Otherwise, you can use either a semicolon or a colon.

Examples

```
SET VARIABLE LOGLEVEL = 3; SELECT Products.Brand, Measures.Dollars FROM "Products"

SET VARIABLE DISABLE_CACHE_HIT=1, LOGLEVEL = 3, WEBLANGUAGE='en': SELECT
Products.Brand, Measures.Dollars FROM "Products"
```

Aggregate, Running Aggregate, Time Series, and Reporting Functions

This section contains information about aggregate functions, running aggregate functions, and time series functions:

- Aggregate Functions
- Running Aggregate Functions
- Time Series Functions
- Reporting Functions

Aggregate Functions

Aggregate functions perform operations on multiple values to create summary results.

The aggregate functions cannot be used to form nested aggregation in expressions on logical columns that have a default aggregation rule defined in the Aggregation tab of the Logical Column dialog. To specify nested aggregation, you must define a column with a default aggregation rule and then request the aggregation of the column in a SQL statement.

Aggregate functions include:

- AGGREGATE AT
- AVG
- AVGDISTINCT
- BOTTOMN
- COUNT
- COUNTDISTINCT
- COUNT(*)
- FIRST
- FIRST_PERIOD
- GROUPBYCOLUMN
- GROUPBYLEVEL

- LAST
- LAST_PERIOD
- MAX
- MEDIAN
- MIN
- NTILE
- PERCENTILE
- RANK
- STDDEV
- STDDEV_POP
- SUM
- SUMDISTINCT
- TOPN

AGGREGATE AT

This function aggregates columns based on the level or levels you specify. Using `AGGREGATE AT` guarantees that the aggregate for the measure always occurs at the levels specified after the keyword `AT`, regardless of the `WHERE` clause.

Syntax

```
AGGREGATE(expr AT level [, level1, levelN])
```

Where:

expr is any expression that references at least one measure column

level is the level at which you want to aggregate. You can optionally specify multiple levels.

You cannot specify a level from a dimension that contains levels that are being used as the measure level for the measure you specified in the first argument. For example, you cannot write the function as `AGGREGATE(yearly_sales AT month)` because "month" is from the same time dimension that is being used as the measure level for "yearly_sales."

Example

The following example shows the `AGGREGATE AT` function and example results:

```
SELECT month, year, AGGREGATE(sales AT Year)
FROM timeseriestesting
WHERE year = 1994 AND month = 12
```

Result:

```
Month    Year    AGGREGATE AT year
12       1994    7396
Row count: 1
```

Because the `AGGREGATE AT` operator is always executed before the predicates, it always returns the correct total for the time level specified after the keyword `AT`.

AVG

This function calculates the average (mean) value of an expression in a result set. It must take a numeric expression as its argument.

Note that the denominator of AVG is the number of rows aggregated. For this reason, it is usually a mistake to use AVG(*x*) in a calculation in Oracle Business Intelligence. Instead, write the expression manually so that you can control both the numerator and denominator (x/y).

Syntax

AVG(*numExpr*)

Where:

numExpr is any expression that evaluates to a numeric value.

AVGDISTINCT

This function calculates the average (mean) of all distinct values of an expression. It must take a numeric expression as its argument.

Syntax

AVG(DISTINCT *numExpr*)

Where:

numExpr is any expression that evaluates to a numeric value.

BOTTOMN

This function ranks the lowest *n* values of the expression argument from 1 to *n*, 1 corresponding to the lowest numeric value. The BOTTOMN function operates on the values returned in the result set. A request can contain only one BOTTOMN expression.

Syntax

BOTTOMN(*numExpr*, *integer*)

Where:

numExpr is any expression that evaluates to a numeric value.

integer is any positive integer. Represents the bottom number of rankings displayed in the result set, 1 being the lowest rank.

COUNT

This function calculates the number of rows having a nonnull value for the expression. The expression is typically a column name, in which case the number of rows with nonnull values for that column is returned.

Syntax:

COUNT(*expr*)

Where:

expr is any expression.

COUNTDISTINCT

This function adds distinct processing to the COUNT function.

Syntax

```
COUNT(DISTINCT expr)
```

Where:

expr is any expression.

COUNT(*)

This function counts the number of rows.

Syntax

```
COUNT(*)
```

Example

For example, if a table named Facts contained 200,000,000 rows, the sample request would return the results shown:

```
SELECT COUNT(*) FROM Facts
```

Result:

```
200000000
```

FIRST

This function selects the first non-null value of the expression argument.

Do not use the FIRST function when you need to compute the first value based on the chronological key rather than the primary level key, or if you need to return the first value regardless of whether it is null. Instead, use the FIRST_PERIOD function. See "FIRST_PERIOD" for more information.

The FIRST function is limited to defining dimension-specific aggregation rules in a repository. You cannot use it in SQL statements.

The FIRST function operates at the most detailed level specified in your explicitly defined dimension. For example, if you have a time dimension defined with hierarchy levels day, month, and year, the FIRST function returns the first day in each level.

You should not use the FIRST function as the first dimension-specific aggregate rule. It might cause queries to bring back large numbers of rows for processing in the Oracle BI Server, causing poor performance.

When a measure is based on dimensions, and data is dense, the Oracle BI Server optimizes the SQL statements sent to the database to improve performance. See "Setting Default Levels of Aggregation for Measure Columns" for more information about dense data.

Note that you cannot nest PERIODROLLING, FIRST, FIRST_PERIOD, LAST, and LAST_PERIOD functions.

Syntax

```
FIRST(expr)
```

Where:

expr is any expression that references at least one measure column.

Example

```
FIRST(sales)
```

FIRST_PERIOD

This function selects the first returned value of the expression argument. For example, the FIRST_PERIOD function can calculate the value of the first day of the year.

Note that the FIRST and FIRST_PERIOD functions have different semantics. FIRST returns the first non-missing (that is recorded) value of its argument in the given period. Whereas FIRST_PERIOD returns either the value that corresponds to the first chronological key in the given period or nothing if that value is missing.

For example, consider the following date and sales table:

Date	Inventory
Jan 5	10
Jan 10	20

If Inventory is a measure with the aggregation rule FIRST, the value in query SELECT Month, Inventory will be 10 (that is, the first recorded value).

If Inventory is a measure with aggregation rule FIRST_PERIOD, the value for the same query will be an empty set because there is no recorded Inventory on January 1, assuming you are using a standard calendar table.

Note the following FIRST_PERIOD function restrictions:

- The FIRST_PERIOD function is limited to defining dimension-specific aggregation rules in a repository.
- When using the FIRST_PERIOD function, you must select the **Data is dense** field on the Logical Column Aggregation tab in the Administration Tool. If you do not select this field, then the aggregation rule reverts to the FIRST function.
- You cannot use the FIRST_PERIOD function in SQL statements.
- The FIRST_PERIOD function is only applicable to dimensions marked as time dimensions. Use the FIRST function for other dimensions.
- Do not use the FIRST_PERIOD function when defining a dimension dependent measure on more than one time dimension. Instead, use the FIRST function when defining dimension-specific aggregation rules.
- Do not use the FIRST_PERIOD function as the first dimension-specific aggregate rule. It might cause queries to bring back large numbers of rows for processing in the Oracle BI Server, causing poor performance.
- Note that you cannot nest PERIODROLLING, FIRST, FIRST_PERIOD, LAST, and LAST_PERIOD functions.

The FIRST_PERIOD function operates at the most detailed level specified in your explicitly defined dimension. For example, if you have a time dimension defined with hierarchy levels day, month, and year, the FIRST_PERIOD function returns the first day in each level.

When a measure is based on dimensions, and data is dense, the Oracle BI Server optimizes the SQL statements sent to the database to improve performance. See "Setting Default Levels of Aggregation for Measure Columns"for more information about dense data.

Syntax

```
FIRST_PERIOD(expr)
```

Where:

expr is any expression that references at least one measure column.

Example

```
FIRST_PERIOD(sales)
```

GROUPBYCOLUMN

For use in setting up aggregate navigation. It specifies the logical columns that define the level of the aggregate data existing in a physical aggregate table.

For example, if an aggregate table contains data grouped by store and by month, specify the following syntax in the content filter (General tab of Logical Source dialog):

```
GROUPBYCOLUMN(STORE, MONTH)
```

The GROUPBYCOLUMN function is only for use in configuring a repository. You cannot use it to form SQL statements.

GROUPBYLEVEL

For use in setting up aggregate navigation. It specifies the dimension levels that define the level of the aggregate data existing in a physical aggregate table.

For example, if an aggregate table contains data at the store and month levels, and if you have defined dimensions (Geography and Customers) that contains these levels, specify the following syntax in the content filter (General tab of Logical Source dialog):

```
GROUPBYLEVEL(GEOGRAPHY.STORE, CUSTOMERS.MONTH)
```

The GROUPBYLEVEL function is only for use in configuring a repository. You cannot use it to form SQL statements.

LAST

This function selects the last non-null value of the expression.

Do not use the LAST function when you need to compute the last value based on the chronological key rather than the primary level key, or if you need to return the last value regardless of whether it is null. Instead, use the LAST_PERIOD function. See "LAST_PERIOD" for more information.

The LAST function is limited to defining dimension-specific aggregation rules in a repository. You cannot use it in SQL statements.

The LAST function operates at the most detailed level specified in your explicitly defined dimension. For example, if you have a time dimension defined with hierarchy levels day, month, and year, the LAST function returns the last day in each level.

You should not use the LAST function as the first dimension-specific aggregate rule. It might cause queries to bring back large numbers of rows for processing in the Oracle BI Server, causing poor performance.

When a measure is based on dimensions, and data is dense, the Oracle BI Server optimizes the SQL statements sent to the database to improve performance. See "Setting Default Levels of Aggregation for Measure Columns" for more information about dense data.

Note that you cannot nest PERIODROLLING, FIRST, FIRST_PERIOD, LAST, and LAST_PERIOD functions.

Syntax

```
LAST(expr)
```

Where:

expr is any expression that references at least one measure column.

Example

```
LAST(sales)
```

LAST_PERIOD

This function selects the last returned value of the expression. For example, the LAST_PERIOD function can calculate the value of the last day of the year.

The LAST_PERIOD function operates at the most detailed level specified in your explicitly defined dimension. For example, if you have a time dimension defined with hierarchy levels day, month, and year, the LAST_PERIOD function returns the last day in each level.

Note that the LAST and LAST_PERIOD functions have different semantics. LAST returns the last non-missing (that is recorded) value of its argument in the given period. Whereas LAST_PERIOD returns either the value that corresponds to the last chronological key in the given period or nothing if that value is missing.

For example, consider the following date and sales table:

Date	Inventory
Jan 1	10
Jan 5	20

If Inventory is a measure with the aggregation rule LAST, the value in query SELECT Month, Inventory will be 20 (that is, the last recorded value).

If Inventory is a measure with aggregation rule LAST_PERIOD, the value for the same query will be an empty set because there is no recorded Inventory on January 31, assuming you are using a standard calendar table.

Note the following LAST_PERIOD function restrictions:

- The LAST_PERIOD function is limited to defining dimension-specific aggregation rules in a repository.
- When using the LAST_PERIOD function, you must select the **Data is dense** field on the Logical Column Aggregation tab in the Administration Tool. If you do not select this field, then the aggregation rule reverts to the LAST function.
- You cannot use the LAST_PERIOD function in SQL statements.
- The LAST_PERIOD function is only applicable to time dimensions. Use the LAST function for other dimensions.
- Do not use the LAST_PERIOD function when defining a dimension dependent measure on more than one time dimension. Instead, use the LAST function when defining dimension-specific aggregation rules.

- Do not use the LAST_PERIOD function as the first dimension-specific aggregate rule. It might cause queries to bring back large numbers of rows for processing in the Oracle BI Server, causing poor performance.
- Note that you cannot nest PERIODROLLING, FIRST, FIRST_PERIOD, LAST, and LAST_PERIOD functions.

When a measure is based on dimensions, and data is dense, the Oracle BI Server optimizes the SQL statements sent to the database to improve performance. See "Setting Default Levels of Aggregation for Measure Columns" for more information about dense data.

Syntax

```
LAST_PERIOD(expr)
```

Where:

expr is any expression that references at least one measure column.

Example

```
LAST_PERIOD(sales)
```

MAX

This function calculates the maximum value (highest numeric value) of the rows satisfying the numeric expression argument.

Syntax

```
MAX(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

The MAX function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

MEDIAN

This function calculates the median (middle) value of the rows satisfying the numeric expression argument. When there are an even number of rows, the median is the mean of the two middle rows. This function always returns a double.

Syntax

```
MEDIAN(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

The MEDIAN function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

MIN

This function calculates the minimum value (lowest numeric value) of the rows satisfying the numeric expression argument.

Syntax

```
MIN(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

The MIN function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

NTILE

This function determines the rank of a value in terms of a user-specified range. It returns integers to represent any range of ranks. In other words, the resulting sorted data set is broken into several tiles where there are roughly an equal number of values in each tile.

NTile with *numTiles* = 100 returns what is commonly called the "percentile" (with numbers ranging from 1 to 100, with 100 representing the high end of the sort). This value is different from the results of the Oracle BI PERCENTILE function, which conforms to what is called "percent rank" in SQL 92 and returns values from 0 to 1.

Syntax

```
NTILE(numExpr, numTiles)
```

Where:

numExpr is any expression that evaluates to a numeric value.

numTiles is a positive, nonnull integer that represents the number of tiles.

If the *numExpr* argument is not null, the function returns an integer that represents a rank within the requested range.

PERCENTILE

This function calculates a percent rank for each value satisfying the numeric expression argument. The percentile rank ranges are from 0 (1st percentile) to 1 (100th percentile), inclusive.

The percentile is calculated based on the values in the result set.

Syntax

```
PERCENTILE(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

The PERCENTILE function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

RANK

This function calculates the rank for each value satisfying the numeric expression argument. The highest number is assigned a rank of 1, and each successive rank is assigned the next consecutive integer (2, 3, 4,...). If certain values are equal, they are assigned the same rank (for example, 1, 1, 1, 4, 5, 5, 7...).

The rank is calculated based on the values in the result set.

Syntax

```
RANK(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

The RANK function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

STDDEV

This function returns the standard deviation for a set of values. The return type is always a double. STDEV_SAMP is a synonym for STDDEV.

Syntax

```
STDDEV([ALL | DISTINCT] numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

If ALL is specified, the standard deviation is calculated for all data in the set.

If DISTINCT is specified, all duplicates are ignored in the calculation.

If nothing is specified (the default), all data is considered.

The STDDEV function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

STDDEV_POP

This function returns the standard deviation for a set of values using the computational formula for population variance and standard deviation.

Syntax

```
STDDEV_POP([ALL | DISTINCT] numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

If ALL is specified, the standard deviation is calculated for all data in the set.

If DISTINCT is specified, all duplicates are ignored in the calculation.

If nothing is specified (the default), all data is considered.

SUM

This function calculates the sum obtained by adding up all values satisfying the numeric expression argument.

Syntax

```
SUM(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

The SUM function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

SUMDISTINCT

This function calculates the sum obtained by adding all of the distinct values satisfying the numeric expression argument.

Syntax

```
SUM(DISTINCT numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

TOPN

This function ranks the highest *n* values of the expression argument from 1 to *n*, 1 corresponding to the highest numeric value. The TOPN function operates on the values returned in the result set. A request can contain only one TOPN expression.

Syntax

```
TOPN(numExpr, integer)
```

Where:

numExpr is any expression that evaluates to a numeric value.

integer is any positive integer. Represents the top number of rankings displayed in the result set, 1 being the highest rank.

The TOPN function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

Running Aggregate Functions

Running aggregate functions are similar to functional aggregates in that they take a set of records as input, but instead of outputting the single aggregate for the entire set of records, they output the aggregate based on records encountered so far.

This section describes the running aggregate functions supported by the Oracle BI Server. Functions include:

- MAVG
- MSUM
- RSUM
- RCOUNT
- RMAX
- RMIN

MAVG

This function calculates a moving average (mean) for the last *n* rows of data in the result set, inclusive of the current row.

The average for the first row is equal to the numeric expression for the first row. The average for the second row is calculated by taking the average of the first two rows of data. The average for the third row is calculated by taking the average of the first three rows of data, and so on until you reach the *n*th row, where the average is calculated based on the last *n* rows of data.

Syntax

```
MAVG(numExpr, integer)
```

Where:

numExpr is any expression that evaluates to a numeric value.

integer is any positive integer. Represents the average of the last *n* rows of data.

The MAVG function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

MSUM

This function calculates a moving sum for the last *n* rows of data, inclusive of the current row.

The sum for the first row is equal to the numeric expression for the first row. The sum for the second row is calculated by taking the sum of the first two rows of data. The sum for the third row is calculated by taking the sum of the first three rows of data, and so on. When the *n*th row is reached, the sum is calculated based on the last *n* rows of data.

Syntax

```
MSUM(numExpr, integer)
```

Where:

numExpr is any expression that evaluates to a numeric value.

integer is any positive integer. Represents the average of the last *n* rows of data.

The MSUM function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

Example

This example shows a query that uses the MSUM function, along with example query results.

```
select month, revenue, MSUM(revenue, 3) as 3_MO_SUM from sales_subject_area
```

Result:

```
MONTH    REVENUE    3_MO_SUM
JAN      100.00     100.00
FEB      200.00     300.00
MAR      100.00     400.00
APRIL    100.00     400.00
MAY      300.00     500.00
JUNE     400.00     800.00
JULY     500.00     1200.00
AUG      500.00     1400.00
SEPT     500.00     1500.00
OCT      300.00     1300.00
NOV      200.00     1000.00
DEC      100.00     600.00
```

RSUM

This function calculates a running sum based on records encountered so far. The sum for the first row is equal to the numeric expression for the first row. The sum for the

second row is calculated by taking the sum of the first two rows of data. The sum for the third row is calculated by taking the sum of the first three rows of data, and so on.

Syntax

```
RSUM(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

The RSUM function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

Example

This example shows a query that uses the RSUM function, along with example query results.

```
SELECT month, revenue, RSUM(revenue) as RUNNING_SUM from sales_subject_area
```

Result:

```
MONTH    REVENUE    RUNNING_SUM
JAN      100.00     100.00
FEB      200.00     300.00
MAR      100.00     400.00
APRIL    100.00     500.00
MAY      300.00     800.00
JUNE     400.00     1200.00
JULY     500.00     1700.00
AUG      500.00     2200.00
SEPT     500.00     2700.00
OCT      300.00     3000.00
NOV      200.00     3200.00
DEC      100.00     3300.00
```

RCOUNT

This function takes a set of records as input and counts the number of records encountered so far.

Syntax

```
RCOUNT(expr)
```

Where:

expr is an expression of any data type.

The RCOUNT function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

Example

This example shows a query that uses the RCOUNT function, along with example query results.

```
select month, profit, RCOUNT(profit) from sales_subject_area where profit > 200
```

Result:

```
MONTH    PROFIT    RCOUNT(profit)
MAY      300.00    2
```

```
JUNE      400.00    3
JULY      500.00    4
AUG       500.00    5
SEPT      500.00    6
OCT       300.00    7
```

RMAX

This function takes a set of records as input and shows the maximum value based on records encountered so far. The specified data type must be one that can be ordered.

Syntax

```
RMAX(expr)
```

Where:

expr is an expression of any data type. The data type must be one that has an associated sort order.

The RMAX function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

Example

This example shows a query that uses the RMAX function, along with example query results.

```
SELECT month, profit, RMAX(profit) from sales_subject_area
```

Result:

```
MONTH     PROFIT    RMAX(profit)
JAN       100.00    100.00
FEB       200.00    200.00
MAR       100.00    200.00
APRIL     100.00    200.00
MAY       300.00    300.00
JUNE      400.00    400.00
JULY      500.00    500.00
AUG       500.00    500.00
SEPT      500.00    500.00
OCT       300.00    500.00
NOV       200.00    500.00
DEC       100.00    500.00
```

RMIN

This function takes a set of records as input and shows the minimum value based on records encountered so far. The specified data type must be one that can be ordered.

Syntax

```
RMIN(expr)
```

Where:

expr is an expression of any data type. The data type must be one that has an associated sort order.

The RMIN function resets its values for each group in the query according to specific rules. See "Display Function Reset Behavior" for more information.

Example

This example shows a query that uses the RMIN function, along with example query results.

```
select month, profit, RMIN(profit) from sales_subject_area
```

Result:

```
MONTH    PROFIT    RMIN(profit)
JAN      400.00    400.00
FEB      200.00    200.00
MAR      100.00    100.00
APRIL    100.00    100.00
MAY      300.00    100.00
JUNE     400.00    100.00
JULY     500.00    100.00
AUG      500.00    100.00
SEPT     500.00    100.00
OCT      300.00    100.00
NOV      200.00    100.00
DEC      100.00    100.00
```

Time Series Functions

Time series functions operate on time-oriented dimensions. The time series functions calculate AGO, TODATE, and PERIODROLLING functions based on user supplied calendar tables, not on standard SQL date manipulation functions.

These functions let you use Expression Builder to call a logical function to perform time series calculations instead of aliasing physical tables and modeling logically.

To use time series functions on a particular dimension, you must designate the dimension as a Time dimension and set one or more keys at one or more levels as chronological keys. See "Modeling Time Series Data" for more information.

Functions include:

- AGO
- PERIODROLLING
- TODATE

AGO

This function is a time series aggregation function that calculates the aggregated value from the current time grain (as determined by the measure grain and predicates) back to a specified time period. For example, AGO can produce sales for every month of the current quarter and the corresponding quarter-ago sales.

The grain at which the time shift occurs is determined by the measure grain and predicates. For example, in the following query, the time shift occurs at the quarter level and goes back one year:

```
Select year, quarter, Ago(sales, year_level, 1)
```

However, in the following query, the time shift occurs at the day level, as determined by the associated level of the day_of_month attribute).

```
Select year, quarter, Ago(sales, year_level, 1)
Where day_of_month IN (1, 2)
```

When computing the time shift, Oracle BI Server steps through the time hierarchy. For example, if the time hierarchy is Year, then Month, then Day, then Oracle BI Server computes the time shift as follows:

1. Year is shifted by 1.
2. Moth in year remains the same.
3. Day of month remains the same.

See "About the AGO Function Level" for more information about the level of the function.

If unsupported metrics are requested, NULL values are returned and a warning entry is written to the nqquery.log file when the logging level equals three or above.

Multiple `AGO` functions can be nested if all the `AGO` functions have the same level argument. You can nest exactly one `TODATE` and multiple `AGO` functions if they each have the same level argument.

Syntax

```
AGO(expr, [time_level,] offset)
```

Where:

`expr` is an expression that references at least one measure column.

`time_level` is an optional argument that specifies the type of time period, such as quarter, month, or year.

In the Administration Tool, specify a logical level for *`time_level`*.

`offset` is an integer literal that represents the time shift amount.

Example

The following example returns last year's sales:

```
SELECT Year_ID, AGO(sales, year, 1)
```

About the AGO Function Level It is recommended that you explicitly specify the level of the `AGO` function using the `[time_level]` argument.

If you do not explicitly specify the `[time_level]` argument, the default level is determined as follows:

- If the measure used in the expression is a level-based measure in the time dimension (as set in the Administration Tool), then that same level is considered the default `AGO` level.
- Otherwise, the grain of the measure used in the expression, as determined by the `BY` clause of the measure shown in the logical request, is the default Ago level.

  For example, the result of the query:

  ```
  SELECT year, AGO(sales, 1) WHERE quarter=1
  ```

  is the same as:

  ```
  SELECT year, AGO(sales, year_level, 1) WHERE quarter=1
  ```

You can see the default `AGO` level for a given query in the Logical Request section of the query log.

PERIODROLLING

This function computes the aggregate of a measure over the period starting *x* units of time and ending *y* units of time from the current time. For example, you can use PERIODROLLING to compute sales for a period that starts at a certain quarter before and ends at a certain quarter after the current quarter.

Time series functions operate on members of time dimensions which are at or below the level of the function. Because of this, one or more columns that uniquely identify members at or below the given level must be projected in the query. Alternatively, you can apply a filter to the query that specifies a single member at or below the given level. See "Determining the Level Used by the PERIODROLLING Function" for more information about the level of the function.

You cannot nest AGO and TODATE functions within a PERIODROLLING function. Also, you cannot nest PERIODROLLING, FIRST, FIRST_PERIOD, LAST, and LAST_PERIOD functions.

If you embed other aggregate functions (like RANK, TOPN, PERCENTILE, FILTER, or RSUM) inside PERIODROLLING, the PERIODROLLING function is pushed inward. For example, PERIODROLLING(TOPN(*measure*)) is executed as TOPN(PERIODROLLING(*measure*)).

Syntax

```
PERIODROLLING(measure, x ,y [,hierarchy])
```

Where:

measure is the name of a measure column.

x is an integer that specifies the offset from the current time. Precede the integer with a minus sign (-) to indicate an offset into the past.

y specifies the number of time units over which the function will compute. To specify the current time, enter 0.

hierarchy is an optional argument that specifies the name of a hierarchy in a time dimension, such as yr, mon, day, that you want to use to compute the time window. This option is useful when there are multiple hierarchies in a time dimension, or when you want to distinguish between multiple time dimensions.

If you want to roll back or forward the maximum possible amount, use the keyword UNBOUND. For example, the function PERIODROLLING (*measure*, -UNBOUND, 0) sums over the period starting from the beginning of time until now.

You can combine PERIODROLLING and AGGREGATE AT functions to specify the level of the PERIODROLLING function explicitly. For example, if the query level is day but you want to find the sum of the previous and current months, use the following:

```
SELECT year, month, day, PERIODROLLING(AGGREGATE(sales AT month), -1)
```

Examples

```
SELECT Month_ID, PERIODROLLING(monthly_sales, -1, 1)

SELECT Month_ID, PERIODROLLING(monthly_sales, -UNBOUND, 2)

SELECT Month_ID, PERIODROLLING(monthly_sales, -UNBOUND, UNBOUND)
```

Determining the Level Used by the PERIODROLLING Function The unit of time (offset) used in the PERIODROLLING function is called the *level* of the function. This value is determined by the measure level of the measures in its first argument and the query level of the query to which the function belongs. The measure level for the measure can be set in the Administration Tool. If a measure level has been set for the measure

used in the function, the measure level is used as the level of the function. The measure level is also called the *storage grain* of the function.

If a measure level has not been set in the Administration Tool, then the query level is used. The query level is also called the *query grain* of the function. In the following example, the query level is month, and the PERIODROLLING function computes the sum of the last, current, and next month for each city for the months of March and April:

```
SELECT year, month, country, city, PERIODROLLING(sales, -1, 1)
WHERE month in ('Mar', 'Apr') AND city = 'New York'
```

When there are multiple hierarchies in the time dimension, you must specify the *hierarchy* argument in the PERIODROLLING function. For example:

```
SELECT year, fiscal_year, month, PERIODROLLING(sales, -1, 1, "fiscal_time_
hierarchy")
```

In this example, the level of the PERIODROLLING function is fiscal_year.

TODATE

This function is a time series aggregation function that aggregates a measure from the beginning of a specified time period to the currently displayed time. For example, this function can calculate Year to Date sales.

Time series functions operate on members of time dimensions which are at or below the level specified in the function. Because of this, one or more columns that uniquely identify members at or below the given level must be projected in the query. Alternatively, you can apply a filter to the query that specifies a single member at or below the given level.

If unsupported metrics are requested, NULL values are returned and a warning entry is written to the nqquery.log file when the logging level equals three or above.

A TODATE function may not be nested within another TODATE function. You can nest exactly one TODATE and multiple AGO functions if they each have the same level argument.

TODATE is different from the TO_DATE SQL function supported by some databases. Do not use TO_DATE to change to a DATE data type. Instead, use the CAST function. See "CAST" for more information.

Syntax

```
TODATE(expr, time_level)
```

Where:

expr is an expression that references at least one measure column.

time_level is the type of time period, such as quarter, month, or year.

Example

The following example returns the year-to-month sales:

```
SELECT Year_ID, Month_ID, TODATE(sales, year)
```

Reporting Functions

Reporting functions are computed after all predicates have been evaluated and are useful when computing subtotals of values visible in the report. Reporting functions are only allowed in the project list (SELECT clause).

These functions can only be used in Logical SQL and cannot be used in expressions inside the Oracle BI repository.

Functions include:

- REPORT_AGGREGATE
- REPORT_AVG
- REPORT_COUNT
- REPORT_MAX
- REPORT_MIN
- REPORT_SUM

REPORT_AGGREGATE

REPORT_AGGREGATE is a generalization of other reporting functions like REPORT_SUM, REPORT_MIN, and similar.

It is semantically equivalent to those functions when applied on a measure with the corresponding aggregation rule.

For example, if measure M is defined with the aggregation rule SUM, then:

```
REPORT_AGGREGATE(M) := REPORT_SUM(M)
```

However, if the aggregation rule is MAX, then:

```
REPORT_AGGREGATE(M) := REPORT_MAX(M)
```

REPORT_AGGREGATE is only allowed in the project list (SELECT clause) and is computed after all expressions in the WHERE clause have been evaluated, with one exception: it is computed before the aggregate predicates when applied to a non-linear measure, such as COUNTDISTINCT.

Note: If you do not want REPORT_AGGREGATE to be computed before the aggregate predicates on non-linear measures, re-write your queries so that they do not have any summaries in the WHERE clause. For example, you could rewrite the following query:

```
SELECT Year, DistinctUnitsSold,
REPORT_AGGREGATE(DistinctUnitsSold BY )
WHERE Sales > 1000
```

as:

```
SELECT Year, DistinctProductsSold, REPORT_
AGGREGATE(DistinctUnitsSold BY )
WHERE Year IN (
SELECT Year
WHERE sales > 1000
)
```

Alternatively, you can declare the non-linear measure as a derived column instead of a measure. For example:

```
COUNT(DISTINCT UnitId)
```

Because it is generally computed after all predicates have been evaluated, this function is useful when computing subtotals of values visible in the report (hence the name REPORT_AGGREGATE).

When REPORT_AGGREGATE is applied to an expression E of measures (M1, M2, and so on) and scalar attributes (S1, S2, and so on), it is distributed to the component measures, as follows:

```
REPORT_AGGREGATE(E(M1, M2, ..., S1, S2, ...)) := E(REPORT_AGGREGATE(M1), REPORT_
AGGREGATE(M2), ..., S1, S2, ...)
```

For example:

```
REPORT_AGGREGATE((sales + 10) / number_of_employees) := (REPORT_AGGREGATE(sales) +
10) / REPORT_AGGREGATE(number_of_employees)
```

REPORT_AVG

This function computes the average (mean) value of the *argument_expression* partitioned by the set of BY column expressions.

An empty BY clause specifies a single (grand total) partition.

Columns in the BY clause must be a subset of the (implicit) GROUP BY columns of the *argument_expression*.

REPORT_AVG is only allowed in the project list (SELECT clause) and is computed after all expressions in the WHERE clause.

Because REPORT_AVG is computed after all other predicates, it is suitable for computing report subtotal values.

While REPORT_AVG always uses AVG as the aggregation rule, REPORT_AGGREGATE tries to automatically determine the aggregation rule, based on its argument.

It is recommended to use REPORT_AGGREGATE by default. However, when the default behavior of REPORT_AGGREGATE does not produce the desired result, an explicit aggregation rule like REPORT_AVG can be used.

Syntax

```
REPORT_AVG(argument_expression BY column_expression, column_expression, ...)
```

Example

```
SELECT month, year, sales, REPORT_AVG(sales BY year) AS yearly_total
WHERE month LIKE 'J%' AND sales > 10
```

Result:

Month	Year	Sales	Yearly_Total
June	2011	20	25
July	2011	30	25
January	2012	40	45
June	2012	50	45

REPORT_COUNT

This function calculates the number of rows having a nonnull value for the *argument_expression* partitioned by the set of BY column expressions.

An empty BY clause specifies a single (grand total) partition.

Columns in the BY clause must be a subset of the (implicit) GROUP BY columns of the *argument_expression*.

REPORT_COUNT is only allowed in the project list (SELECT clause) and is computed after all expressions in the WHERE clause.

Because REPORT_COUNT is computed after all other predicates, it is suitable for computing report subtotal values.

While REPORT_COUNT always uses COUNT as the aggregation rule, REPORT_AGGREGATE tries to automatically determine the aggregation rules, based on its argument.

It is recommended to use REPORT_AGGREGATE by default. However, when the default behavior of REPORT_AGGREGATE does not produce the desired result, an explicit aggregation rule like REPORT_COUNT can be used.

Syntax

```
REPORT_COUNT(argument_expression BY column_expression, column_expression, ...)
```

Example

```
SELECT month, year, sales, REPORT_COUNT(sales BY year) AS yearly_total
WHERE month LIKE 'J%' AND sales > 10
```

Result:

Month	Year	Sales	Yearly_Total
June	2011	20	2
July	2011	30	2
January	2012	40	2
June	2012	50	2

REPORT_MAX

This function calculates the maximum value (highest numeric value) of the rows satisfying the numeric *argument_expression* partitioned by the set of BY column expressions.

An empty BY clause specifies a single (grand total) partition.

Columns in the BY clause must be a subset of the (implicit) GROUP BY columns of the *argument_expression*.

REPORT_MAX is only allowed in the project list (SELECT clause) and is computed after all expressions in the WHERE clause.

Because REPORT_MAX is computed after all other predicates, it is suitable for computing report subtotal values.

While REPORT_MAX always uses MAX as the aggregation rule, REPORT_AGGREGATE tries to automatically determine the aggregation rules, based on its argument.

It is recommended to use REPORT_AGGREGATE by default. However, when the default behavior of REPORT_AGGREGATE does not produce the desired result, an explicit aggregation rule like REPORT_MAX can be used.

Syntax

```
REPORT_MAX(argument_expression BY column_expression, column_expression, ...)
```

Example

```
SELECT month, year, sales, REPORT_MAX(sales BY year) AS yearly_total
WHERE month LIKE 'J%' AND sales > 10
```

Result:

Month	Year	Sales	Yearly_Total
June	2011	20	30
July	2011	30	30
January	2012	40	50
June	2012	50	50

REPORT_MIN

This function calculates the minimum value (lowest numeric value) of the rows satisfying the *argument_expression* partitioned by the set of BY column expressions.

An empty BY clause specifies a single (grand total) partition.

Columns in the BY clause must be a subset of the (implicit) GROUP BY columns of the *argument_expression*.

REPORT_MIN is only allowed in the project list (SELECT clause) and is computed after all expressions in the WHERE clause.

Because REPORT_MIN is computed after all other predicates, it is suitable for computing report subtotal values.

While REPORT_MIN always uses MIN as the aggregation rule, REPORT_AGGREGATE tries to automatically determine the aggregation rules, based on its argument.

It is recommended to use REPORT_AGGREGATE by default. However, when the default behavior of REPORT_AGGREGATE does not produce the desired result, an explicit aggregation rule like REPORT_MIN can be used.

Syntax

```
REPORT_MIN(argument_expression BY column_expression, column_expression, ...)
```

Example

```
SELECT month, year, sales, REPORT_MIN(sales BY year) AS yearly_total
WHERE month LIKE 'J%' AND sales > 10
```

Result:

Month	Year	Sales	Yearly_Total
June	2011	20	20
July	2011	30	20
January	2012	40	40
June	2012	50	40

REPORT_SUM

This function computes the sum of the *argument_expression* partitioned by the set of BY column expressions.

An empty BY clause specifies a single (grand total) partition.

Columns in the BY clause must be a subset of the (implicit) GROUP BY columns of the *argument_expression*.

REPORT_SUM is only allowed in the project list (SELECT clause) and is computed after all expressions in the WHERE clause.

Because REPORT_SUM is computed after all other predicates, it is suitable for computing report subtotal values.

While REPORT_SUM always uses SUM as the aggregation rule, REPORT_AGGREGATE tries to automatically determine the aggregation rule, based on its argument.

It is recommended to use REPORT_AGGREGATE by default. However, when the default behavior of REPORT_AGGREGATE does not produce the desired result, an explicit aggregation rule like REPORT_SUM can be used. For example, use REPORT_SUM to compute totals or subtotals as a simple visual sum of the values displayed in a report rather than using the same aggregation rule as the base measure.

Syntax

```
REPORT_SUM(argument_expression BY column_expression, column_expression, ...)
```

Example

```
SELECT month, year, sales, REPORT_SUM(sales BY year) AS yearly_total
WHERE month LIKE 'J%' AND sales > 10
```

Result:

Month	Year	Sales	Yearly_Total
June	2011	20	50
July	2011	30	50
January	2012	40	90
June	2012	50	90

String Functions

String functions perform various character manipulations, and they operate on character strings. Functions include:

- ASCII
- BIT_LENGTH
- CHAR
- CHAR_LENGTH
- CONCAT
- INSERT
- LEFT
- LENGTH

- LOCATE
- LOWER
- OCTET_LENGTH
- POSITION
- REPEAT
- REPLACE
- RIGHT
- SPACE
- SUBSTRING
- TRIMBOTH
- TRIMLEADING
- TRIMTRAILING
- UPPER

ASCII

This function converts a single character string to its corresponding ASCII code, between 0 and 255. If the character expression evaluates to multiple characters, the ASCII code corresponding to the first character in the expression is returned.

Syntax

```
ASCII(strExpr)
```

Where:

strExpr is any expression that evaluates to a character string.

BIT_LENGTH

This function returns the length, in bits, of a specified string. Each Unicode character is 2 bytes in length (equal to 16 bits).

Syntax

```
BIT_LENGTH(strExpr)
```

Where:

strExpr is any expression that evaluates to character string.

CHAR

This function converts a numeric value between 0 and 255 to the character value corresponding to the ASCII code.

Syntax

```
CHAR(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value between 0 and 255.

CHAR_LENGTH

This function returns the length, in number of characters, of a specified string. Leading and trailing blanks are not counted in the length of the string.

Syntax

```
CHAR_LENGTH(strExpr)
```

Where:

strExpr is any expression that evaluates to a character string.

CONCAT

There are two forms of this function. The first form concatenates two character strings. The second form uses the character string concatenation character to concatenate more than two character strings.

Syntax for Form 1 (To Concatenate Two Strings)

```
CONCAT(strExpr1, strExpr2)
```

Where:

strExprs are expressions that evaluate to character strings, separated by commas.

Example

This example request returns the results shown.

```
SELECT DISTINCT CONCAT('abc', 'def') FROM employee
CONCAT('abc', 'def')
```

Result:

```
abcdef
```

Syntax for Form 2 (To Concatenate More Than Two Strings)

```
CONCAT(strExpr1, strExpr2 || strExpr3)
```

Where:

strExprs are expressions that evaluate to character strings, separated by commas and the character string concatenation operator || (double vertical bars). First, *strExpr2* is concatenated with *strExpr3* to produce an intermediate string, then both *strExpr1* and the intermediate string are concatenated by the CONCAT function to produce the final string.

Example

This example request returns the results shown.

```
SELECT DISTINCT CONCAT('abc','def' || 'ghi') FROM employee
```

Result:

```
abcdefghi
```

INSERT

This function inserts a specified character string into a specified location in another character string.

Syntax

```
INSERT(strExpr1, integer1, integer2, strExpr2)
```

Where:

strExpr1 is any expression that evaluates to a character string. Identifies the target character string.

integer1 is any positive integer that represents the number of characters from the beginning of the target string where the second string is to be inserted.

integer2 is any positive integer that represents the number of characters in the target string to be replaced by the second string.

strExpr2 is any expression that evaluates to a character string. Identifies the character string to be inserted into the target string.

Example

In the first string, starting at the second position (occupied by the number 2), three characters (the numbers 2, 3, and 4) are replaced by the string abcd.

```
SELECT INSERT('123456', 2, 3, 'abcd') FROM table
```

Result:

```
1abcd56
1abcd56
...
```

LEFT

Returns a specified number of characters from the left of a string.

Syntax

```
LEFT(strExpr, integer)
```

Where:

strExpr is any expression that evaluates to a character string.

integer is any positive integer that represents the number of characters from the left of the string to return.

Example

This example returns the three leftmost characters from the character string 123456:

```
SELECT LEFT('123456', 3) FROM table
```

Result:

```
123
123
...
```

LENGTH

This function returns the length, in number of characters, of a specified string. The length is returned excluding any trailing blank characters.

Syntax

```
LENGTH(strExpr)
```

Where:

strExpr is any expression that evaluates to a character string.

LOCATE

This function returns the numeric position of a character string in another character string. If the character string is not found in the string being searched, the function returns a value of 0.

If you want to specify a starting position to begin the search, include the integer argument. The numeric position to return is determined by counting the first character in the string as occupying position 1, regardless of the value of the integer argument.

Syntax

```
LOCATE(strExpr1, strExpr2 [, integer])
```

Where:

strExpr1 is any expression that evaluates to a character string. Identifies the string for which to search.

strExpr2 is any expression that evaluates to a character string. Identifies the string to be searched.

integer is any positive (nonzero) integer that represents the starting position to begin to look for the character string. The integer argument is optional.

Examples

This example returns 4 as the numeric position of the letter d in the character string abcdef:

```
LOCATE('d', 'abcdef')
```

This example returns 0, because the letter g is not found within the string being searched.

```
LOCATE('g', 'abcdef')
```

This example returns 4 as the numeric position of the letter d in the character string abcdef. The search begins with the letter c, the third character in the string. The numeric position to return is determined by counting the letter 'a' as occupying position 1.

```
LOCATE('d' 'abcdef', 3)
```

This example returns 0, because the letter b occurs in the string before the starting position to begin the search.

```
LOCATE('b' 'abcdef', 3)
```

LOWER

This function converts a character string to lowercase.

Syntax

```
LOWER(strExpr)
```

Where:

strExpr is any expression that evaluates to a character string.

OCTET_LENGTH

This function returns the number of bits, in base 8 units (number of bytes), of a specified string.

Syntax

```
OCTET_LENGTH(strExpr)
```

Where:

strExpr is any expression that evaluates to a character string.

POSITION

This function returns the numeric position of *strExpr1* in a character expression. If *strExpr1* is not found, the function returns 0. See also "LOCATE" for related information.

Syntax

```
POSITION(strExpr1 IN strExpr2)
```

Where:

strExpr1 is any expression that evaluates to a character string. Identifies the string to search for in the target string.

strExpr2 is any expression that evaluates to a character string. Identifies the target string to be searched.

Examples

This example returns 4 as the position of the letter d in the character string abcdef:

```
POSITION('d', 'abcdef')
```

This example returns 0 as the position of the number 9 in the character string 123456, because the number 9 is not found.

```
POSITION('9', '123456')
```

REPEAT

This function repeats a specified expression *n* times.

Syntax

```
REPEAT(strExpr, integer)
```

Where:

strExpr is any expression that evaluates to a character string.

integer is any positive integer that represents the number of times to repeat the character string.

Example

This example repeats abc four times:

```
REPEAT('abc', 4)
```

REPLACE

This function replaces one or more characters from a specified character expression with one or more other characters.

Syntax

```
REPLACE(strExpr1, strExpr2, strExpr3)
```

Where:

strExpr1 is any expression that evaluates to a character string. This is the string in which characters are to be replaced.

strExpr2 is any expression that evaluates to a character string. This second string identifies the characters from the first string that are to be replaced.

strExpr3 is any expression that evaluates to a character string. This third string specifies the characters to substitute into the first string.

Example

In the character string abcd1234, the characters 123 are replaced by the character string zz:

```
Replace('abcd1234', '123', 'zz')
```

Result:

```
abcdzz4
```

RIGHT

This function returns a specified number of characters from the right of a string.

Syntax

```
RIGHT(strExpr, integer)
```

Where:

strExpr is any expression that evaluates to a character string.

integer is any positive integer that represents the number of characters from the right of the string to return.

Example

This example returns the three rightmost characters from the character string 123456:

```
SELECT right('123456', 3) FROM table
```

Result:

```
456
```

SPACE

This function inserts blank spaces.

Syntax

```
SPACE(integer)
```

Where:

integer is any positive integer that indicates the number of spaces to insert.

SUBSTRING

This function creates a new string starting from a fixed number of characters into the original string.

Syntax

```
SUBSTRING(strExpr FROM starting_position)
```

Where:

strExpr is any expression that evaluates to a character string.

starting_position is any positive integer that represents the number of characters from the start of the left side of the string where the result is to begin.

TRIMBOTH

This function strips specified leading and trailing characters from a character string.

Syntax

```
TRIM(BOTH character FROM strExpr)
```

Where:

character is any single character. If you omit this specification (and the required single quotes), a blank character is used as the default.

strExpr is any expression that evaluates to a character string.

TRIMLEADING

This function strips specified leading characters from a character string.

Syntax

```
TRIM(LEADING character FROM strExpr)
```

Where:

character is any single character. If you omit this specification (and the required single quotes), a blank character is used as the default.

strExpr is any expression that evaluates to a character string.

TRIMTRAILING

This function strips specified trailing characters from a character string.

Syntax

```
TRIM(TRAILING character FROM strExpr)
```

Where:

character is any single character. If you omit this specification (and the required single quotes), a blank character is used as the default.

strExpr is any expression that evaluates to a character string.

UPPER

This function converts a character string to uppercase.

Syntax

```
UPPER(strExpr)
```

Where:

strExpr is any expression that evaluates to a character string.

Math Functions

The math functions perform mathematical operations. Functions include:

- ABS
- ACOS
- ASIN
- ATAN
- ATAN2
- CEILING
- COS
- COT
- DEGREES
- EXP
- EXTRACTBIT
- FLOOR
- LOG
- LOG10
- MOD
- PI
- POWER
- RADIANs
- RAND

- RANDFROMSEED
- ROUND
- SIGN
- SIN
- SQRT
- TAN
- TRUNCATE

ABS

This function calculates the absolute value of a numeric expression.

Syntax

`ABS(numExpr)`

Where:

numExpr is any expression that evaluates to a numeric value.

ACOS

This function calculates the arc cosine of a numeric expression.

Syntax

`ACOS(numExpr)`

Where:

numExpr is any expression that evaluates to a numeric value.

ASIN

This function calculates the arc sine of a numeric expression.

Syntax

`ASIN(numExpr)`

Where:

numExpr is any expression that evaluates to a numeric value.

ATAN

This function calculates the arc tangent of a numeric expression.

Syntax

`ATAN(numExpr)`

Where:

numExpr is any expression that evaluates to a numeric value.

ATAN2

This function calculates the arc tangent of y/x, where y is the first numeric expression and x is the second numeric expression.

Syntax

```
ATAN2(numExpr1, numExpr2)
```

Where:

numExpr is any expression that evaluates to a numeric value.

CEILING

This function rounds a noninteger numeric expression to the next highest integer. If the numeric expression evaluates to an integer, the CEILING function returns that integer.

Syntax

```
CEILING(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

COS

This function calculates the cosine of a numeric expression.

Syntax

```
COS(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

COT

This function calculates the cotangent of a numeric expression.

Syntax

```
COT(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

DEGREES

This function converts an expression from radians to degrees.

Syntax

```
DEGREES(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

EXP

This function sends the value to the power specified.

Syntax

```
EXP(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

EXTRACTBIT

This function retrieves a bit at a particular position in an integer. It returns an integer of either 0 or 1 corresponding to the position of the bit. The primary use case for this function is to extract 'cell status' in the Hyperion Financial Management cube source. The `EXTRACTBIT` function cannot be pushed into any database, and is always internally executed (in the Oracle BI Server).

Syntax

```
Int ExtractBit(Arg1, Arg2)
```

Where:

Arg1 is an expression of the following types: `INT`, `SMALLINT`, `UNIT`, `SMALLUNIT`, `TINYINT`, `TINYUNIT`. If Arg1 is of double type, it is necessary to cast the column to an `INT` first.

Arg2 is an expression of type integer. The value should range from 1 to *length_of_Arg1*. 1 retrieves the Least Significant Bit. If the Arg2 is beyond the length of the integer, then 0 is returned. An error message is triggered when the Arg2 is less than 1.

FLOOR

This function rounds a noninteger numeric expression to the next lowest integer. If the numeric expression evaluates to an integer, the `FLOOR` function returns that integer.

Syntax

```
FLOOR(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

LOG

This function calculates the natural logarithm of an expression.

Syntax

```
LOG(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

LOG10

This function calculates the base 10 logarithm of an expression.

Syntax

```
LOG10(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

MOD

This function divides the first numeric expression by the second numeric expression and returns the remainder portion of the quotient.

Syntax

```
MOD(numExpr1, numExpr2)
```

Where:

numExpr is any expression that evaluates to a numeric value.

Examples

This example request returns a value of 0:

```
MOD(9, 3)
```

This example request returns a value of 1:

```
MOD(10, 3)
```

PI

This function returns the constant value of pi (the circumference of a circle divided by its diameter).

Syntax

```
PI()
```

POWER

This function takes the first numeric expression and raises it to the power specified in the second numeric expression.

Syntax

```
POWER(numExpr1, numExpr2)
```

Where:

numExpr1 is any expression that evaluates to a numeric value.

RADIANs

This function converts an expression from degrees to radians.

Syntax

```
RADIANS(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

RAND

Returns a pseudo-random number between 0 and 1.

Syntax

```
RAND()
```

RANDFROMSEED

Returns a pseudo-random number based on a seed value. For a given seed value, the same set of random numbers are generated.

Syntax

```
RAND(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

ROUND

This function rounds a numeric expression to *n* digits of precision.

Syntax

```
ROUND(numExpr, integer)
```

Where:

numExpr is any expression that evaluates to a numeric value.

integer is any positive integer that represents the number of digits of precision.

Example

This example returns 2.17 as the result.

```
ROUND(2.166000, 2)
```

SIGN

This function returns the following:

- A value of 1 if the numeric expression argument evaluates to a positive number.
- A value of -1 if the numeric expression argument evaluates to a negative number.
- 0 (zero) if the numeric expression argument evaluates to zero.

Syntax

```
SIGN(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

SIN

This function calculates the sine of a numeric expression.

Syntax

```
SIN(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

SQRT

This function calculates the square root of the numeric expression argument. The numeric expression must evaluate to a nonnegative number.

Syntax

```
SQRT(numExpr)
```

Where:

numExpr is any expression that evaluates to a nonnegative numeric value.

TAN

This function calculates the tangent of a numeric expression.

Syntax

```
TAN(numExpr)
```

Where:

numExpr is any expression that evaluates to a numeric value.

TRUNCATE

This function truncates a decimal number to return a specified number of places from the decimal point.

Syntax

```
TRUNCATE(numExpr, integer)
```

Where:

numExpr is any expression that evaluates to a numeric value.

integer is any positive integer that represents the number of characters to the right of the decimal place to return.

Examples

This example returns 45.12:

```
TRUNCATE(45.12345, 2)
```

This example returns 25.12:

```
TRUNCATE(25.126, 2)
```

Calendar Date/Time Functions

The calendar date/time functions manipulate data of the data types `DATE` and `DATETIME` based on a calendar year. You must select these functions with another column; they cannot be selected alone.

Month and day names are returned in mixed case by default. If you have existing expressions that assume month/day names are returned in uppercase, you must either adjust the expressions, or set the parameters `USE_UPPERCASE_MONTH_NAMES` and `USE_UPPERCASE_DAY_NAMES` to `YES` in NQSConfig.INI. See "NQSConfig.INI File Configuration Settings" in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

Functions such as MONTHNAME can be used to format date columns, but the locale is fixed by the data source or the Oracle BI Server's NQSConfig.INI file. It is not possible for you to specify a locale for the logical SQL date formatting functions because these are based on the ODBC standard, which does not provide for a locale argument.

To format date columns according to the user's selected locale, Oracle suggests that when the locale is fixed, the content designer specifies a standard date format or a custom date format on an analysis' date column. See "Custom Format Strings for Date and Time Fields" and "Column Properties: Data Format tab" in *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

Functions include:

- CURRENT_DATE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- DAY_OF_QUARTER
- DAYNAME
- DAYOFMONTH
- DAYOFWEEK
- DAYOFYEAR
- HOUR
- MINUTE
- MONTH
- MONTH_OF_QUARTER
- MONTHNAME
- NOW
- QUARTER_OF_YEAR
- SECOND
- TIMESTAMPADD
- TIMESTAMPDIFF
- WEEK_OF_QUARTER
- WEEK_OF_YEAR

- YEAR

CURRENT_DATE

This function returns the current date. The date is determined by the system in which the Oracle BI Server is running.

Syntax

```
CURRENT_DATE
```

CURRENT_TIME

This function returns the current time. The time is determined by the system in which the Oracle BI Server is running.

Syntax

```
CURRENT_TIME(integer)
```

Where:

integer is any integer representing the number of digits of precision with which to display the fractional second. The argument is optional; the function returns the default precision when no argument is specified.

CURRENT_TIMESTAMP

This function returns the current date/timestamp. The timestamp is determined by the system in which the Oracle BI Server is running.

Syntax

```
CURRENT_TIMESTAMP(integer)
```

Where:

integer is any integer representing the number of digits of precision with which to display the fractional second. The argument is optional; the function returns the default precision when no argument is specified.

DAY_OF_QUARTER

This function returns a number (between 1 and 92) corresponding to the day of the quarter for the specified date.

Syntax

```
DAY_OF_QUARTER(dateExpr)
```

Where:

dateExpr is any expression that evaluates to a date.

DAYNAME

This function returns the name of the day of the week for a specified date.

Syntax

```
DAYNAME(dateExpr)
```

Where:

dateExpr is any expression that evaluates to a date.

DAYOFMONTH

This function returns the number corresponding to the day of the month for a specified date.

Syntax

```
DAYOFMONTH(dateExpr)
```

Where:

dateExpr is any expression that evaluates to a date.

DAYOFWEEK

This function returns a number between 1 and 7 corresponding to the day of the week, Sunday through Saturday, for a specified date. For example, the number 1 corresponds to Sunday, and the number 7 corresponds to Saturday.

Syntax

```
DAYOFWEEK(dateExpr)
```

Where:

dateExpr is any expression that evaluates to a date.

DAYOFYEAR

This function returns the number (between 1 and 366) corresponding to the day of the year for a specified date.

Syntax

```
DAYOFYEAR(dateExpr)
```

Where:

dateExpr is any expression that evaluates to a date.

HOUR

This function returns a number (between 0 and 23) corresponding to the hour for a specified time. For example, 0 corresponds to 12 a.m. and 23 corresponds to 11 p.m.

Syntax

```
HOUR(timeExpr)
```

Where:

timeExpr is any expression that evaluates to a time.

MINUTE

This function returns a number (between 0 and 59) corresponding to the minute for a specified time.

Syntax

```
MINUTE(timeExpr)
```

Where:

timeExpr is any expression that evaluates to a time.

MONTH

This function returns the number (between 1 and 12) corresponding to the month for a specified date.

Syntax

```
MONTH(dateExpr)
```

Where:

dateExpr is any expression that evaluates to a date.

MONTH_OF_QUARTER

This function returns the number (between 1 and 3) corresponding to the month in the quarter for a specified date.

Syntax

```
MONTH_OF_QUARTER(dateExpr)
```

Where:

dateExpr is any expression that evaluates to a date.

MONTHNAME

This function returns the name of the month for a specified date.

Syntax

```
MONTHNAME(dateExpr)
```

Where:

dateExpr is any expression that evaluates to a date.

NOW

This function returns the current timestamp. The `NOW` function is equivalent to the `CURRENT_TIMESTAMP` function.

Syntax

```
NOW()
```

QUARTER_OF_YEAR

This function returns the number (between 1 and 4) corresponding to the quarter of the year for a specified date.

Syntax

```
QUARTER_OF_YEAR(dateExpr)
```

Where:

dateExpr is any expression that evaluates to a date.

SECOND

This function returns the number (between 0 and 59) corresponding to the seconds for a specified time.

Syntax

```
SECOND(timeExpr)
```

Where:

timeExpr is any expression that evaluates to a time.

TIMESTAMPADD

This function adds a specified number of intervals to a specified timestamp, and returns a single timestamp.

In the simplest scenario, this function adds the specified integer value to the appropriate component of the timestamp, based on the interval. Adding a week translates to adding seven days, and adding a quarter translates to adding three months. A negative integer value results in a subtraction (such as going back in time).

An overflow of the specified component (such as more than 60 seconds, 24 hours, 12 months, and so on) necessitates adding an appropriate amount to the next component. For example, when adding to the day component of a timestamp, this function considers overflow and takes into account the number of days in a particular month (including leap years when February has 29 days).

When adding to the month component of a timestamp, this function verifies that the resulting timestamp has enough days for the day component. For example, adding 1 month to 2000-05-31 does not result in 2000-06-31 because June does not have 31 days. This function reduces the day component to the last day of the month, 2000-06-30 in this example.

A similar issue arises when adding to the year component of a timestamp having a month component of February and a day component of 29 (that is, last day of February in a leap year). If the resulting timestamp does not fall on a leap year, the function reduces the day component to 28.

These actions conform to the behavior of Microsoft SQL Server and the native OCI interface for Oracle Database.

Syntax

```
TIMESTAMPADD(interval, intExpr, timestamp)
```

Where:

interval is the specified interval. Valid values are:

- SQL_TSI_SECOND
- SQL_TSI_MINUTE
- SQL_TSI_HOUR
- SQL_TSI_DAY
- SQL_TSI_WEEK
- SQL_TSI_MONTH
- SQL_TSI_QUARTER
- SQL_TSI_YEAR

intExpr is any expression that evaluates to an integer value.

timestamp is any valid timestamp. This value is used as the base in the calculation.

A null integer expression or a null timestamp passed to this function results in a null return value.

Examples

The following query asks for the resulting timestamp when 3 days are added to 2000-02-27 14:30:00. Since February, 2000 is a leap year, the query returns a single timestamp of 2000-03-01 14:30:00.

```
SELECT TIMESTAMPADD(SQL_TSI_DAY, 3, TIMESTAMP'2000-02-27 14:30:00')
FROM Employee WHERE employeeid = 2;
```

The following query asks for the resulting timestamp when 7 months are added to 1999-07-31 0:0:0. The query returns a single timestamp of 2000-02-29 00:00:00. Notice the reduction of day component to 29 because of the shorter month of February.

```
SELECT TIMESTAMPADD(SQL_TSI_MONTH, 7, TIMESTAMP'1999-07-31 00:00:00')
FROM Employee WHERE employeeid = 2;
```

The following query asks for the resulting timestamp when 25 minutes are added to 2000-07-31 23:35:00. The query returns a single timestamp of 2000-08-01 00:00:00. Notice the propagation of overflow through the month component.

```
SELECT TIMESTAMPADD(SQL_TSI_MINUTE, 25, TIMESTAMP'2000-07-31 23:35:00')
FROM Employee WHERE employeeid = 2;
```

Caution: The TIMESTAMPADD function is turned on by default for Microsoft SQL Server, ODBC, IBM DB2, and Oracle databases. Because DB2 and Oracle semantics do not fully support this function, the answers from this function might not match with what the Oracle BI Server computes.

TIMESTAMPDIFF

This function returns the total number of specified intervals between two timestamps.

This function first determines the timestamp component that corresponds to the specified interval parameter, and then looks at the higher order components of both timestamps to calculate the total number of intervals for each timestamp. For example, if the specified interval corresponds to the month component, the function calculates the total number of months for each timestamp by adding the month component and

twelve times the year component. Then the function subtracts the first timestamp's total number of intervals from the second timestamp's total number of intervals.

> **Note:** This section describes the TIMESTAMPDIFF behavior when the function is calculated in the Oracle BI Server. If this function is calculated in the data source, then the result might be different from the behavior described in this section. If the TIMESTAMPDIFF function result is different from the desired result, then you can disable TIMESTAMP_DIFF_SUPPORTED in the Features tab for the database object in the Administration Tool to ensure that the function is calculated in the Oracle BI Server. However, making this change might adversely affect performance.

The TIMESTAMPDIFF function rounds up to the next integer whenever fractional intervals represent a crossing of an interval boundary. For example, the difference in years between 1999-12-31 and 2000-01-01 is one year because the fractional year represents a crossing from one year to the next (such as 1999 to 2000). By contrast, the difference between 1999-01-01 and 1999-12-31 is zero years because the fractional interval falls entirely within a particular year (that is, 1999). Microsoft SQL Server exhibits the same rounding behavior, but IBM DB2 does not; it always rounds down.

When calculating the difference in weeks, the function calculates the difference in days and divides by seven before rounding. Additionally, the function takes into account how the parameter FIRST_DAY_OF_THE_WEEK has been configured in the NQSConfig.INI file. For example, with Sunday as the start of the week, the difference in weeks between 2000-07-06 (a Thursday) and 2000-07-10 (the following Monday) results in a value of 1 week. With Tuesday as the start of the week, however, the function would return zero weeks since the fractional interval falls entirely within a particular week. When calculating the difference in quarters, the function calculates the difference in months and divides by three before rounding.

The Oracle BI Server pushes down the TIMESTAMPADD and TIMESTAMPDIFF functions to Microsoft SQL Server, Oracle Database, IBM DB2, and ODBC databases by default.

Syntax

```
TIMESTAMPDIFF(interval, timestamp1, timestamp2)
```

Where:

interval is the specified interval. Valid values are:

- SQL_TSI_SECOND
- SQL_TSI_MINUTE
- SQL_TSI_HOUR
- SQL_TSI_DAY
- SQL_TSI_WEEK
- SQL_TSI_MONTH
- SQL_TSI_QUARTER
- SQL_TSI_YEAR

timestamp1 and *timestamp2* are any valid timestamps.

A null timestamp parameter passed to this function results in a null return value.

Example

The following example query asks for a difference in days between timestamps 1998-07-31 23:35:00 and 2000-04-01 14:24:00. It returns a value of 610. Notice that the leap year in 2000 results in an additional day.

```
SELECT TIMESTAMPDIFF
(SQL_TSI_DAY, TIMESTAMP'1998-07-31 23:35:00',TIMESTAMP'2000-04-01 14:24:00')
FROM Employee WHERE employeeid = 2;
```

Caution: The TIMESTAMPDIFF function is turned on by default for Microsoft SQL Server, ODBC, IBM DB2, and Oracle databases. Because DB2 and Oracle semantics do not fully support this function, the answers from this function might not match with what the Oracle BI Server computes.

WEEK_OF_QUARTER

This function returns a number (between 1 and 13) corresponding to the week of the quarter for the specified date.

Syntax

```
WEEK_OF_QUARTER(dateExpr)
```

Where:

dateExpr is any expression that evaluates to a date.

WEEK_OF_YEAR

This function returns a number (between 1 and 53) corresponding to the week of the year for the specified date.

Syntax

```
WEEK_OF_YEAR(dateExpr)
```

Where:

dateExpr is any expression that evaluates to a date.

YEAR

This function returns the year for the specified date.

Syntax

```
YEAR(dateExpr)
```

Where:

dateExpr is any expression that evaluates to a date.

Conversion Functions

The conversion functions convert a value from one form to another. You can also use the VALUEOF function in a filter to reference the value of an Oracle BI system variable. Functions include:

CAST

This function changes the data type of an expression or a null literal to another data type. For example, you can cast a customer_name (a data type of Char or Varchar) or birthdate (a datetime literal). The following are the supported data types to which the value can be changed:

```
CHARACTER, VARCHAR, INTEGER, FLOAT, SMALLINT, DOUBLE PRECISION, DATE, TIME,
TIMESTAMP, BIT, BIT VARYING
```

Depending on the source data type, some destination types are not supported. For example, if the source data type is a `BIT` string, the destination data type must be a character string or another `BIT` string.

Use `CAST` to change to a `DATE` data type.

The following describes unique characteristics of the `CHAR` and `VARCHAR` data types:

- **Casting to a CHAR data type.** You must use a size parameter. If you do not add a size parameter, a default of 30 is added. Syntax options appear in the following list:
  - The recommended syntax is:

    `CAST(`*`expr`*`|NULL AS CHAR(`*`n`*`))`

    For example:

    ```
    CAST(companyname AS CHAR(35))
    ```

  - You can also use the following syntax:

    `CAST(`*`expr`*`|NULL AS `*`data_type`*`)`

    For example:

    ```
    CAST(companyname AS CHAR)
    ```

    Note: If you use this syntax, the Oracle BI Server explicitly converts and stores as `CAST(`*`expr`*`|NULL AS CHAR(30))`

- **Casting to a VARCHAR data type.** You must use a size parameter. If you omit the size parameter, you cannot can save the change.

Examples

```
CAST(hiredate AS CHAR(40)) FROM employee

SELECT CAST(hiredate AS VARCHAR(40)), CAST(age AS double precision),
```

```
CAST(hiredate AS timestamp), CAST(age AS integer) FROM employee

CAST("db"."table"."col" AS date)
```

CHOOSE

This function takes an arbitrary number of parameters and returns the first item in the list that the user has permission to see. However, administrators must model the column permissions in the Administration Tool to enable this behavior. See "INDEXCOL" for an alternate method.

Syntax

```
CHOOSE(expr1, expr2, ..., exprN)
```

For example, a single query can be written to return security-based revenue numbers for the entire organization. The function could look like the following:

```
CHOOSE(L1-Revenue, L2-Revenue, L3-Revenue, L4-Revenue)
```

If the user issuing this function has access to the column L1-Revenue, then that column value would be returned. If the user does not have visibility to the column L1-Revenue but does have visibility to L2-Revenue, then L2-Revenue is returned.

IFNULL

This function tests if an expression evaluates to a null value, and if it does, assigns the specified value to the expression.

Syntax

```
IFNULL(expr, value)
```

Where:

expr is the expression to evaluate.

value is the value to assign if the expression evaluates to a null value.

INDEXCOL

This function can use external information to return the appropriate column for the logged-in user to see. The Oracle BI Server handles this function in the following ways:

- **ODBC Procedures.** `NQSGetLevelDrillability` and `NQSGenerateDrillDownQuery` return the context-specific drill-down information based on the expression translated from `INDEXCOL`. This applies to both `INDEXCOL` expressions specified in the Logical SQL query and `INDEXCOL` expressions specified in a derived logical column.
- **Query Log and cache.** The Logical SQL query with `INDEXCOL` function appears in the SQL string in the query log. But the logical request does not show the `INDEXCOL` function because the Oracle BI Server translates `INDEXCOL` to one of the expressions in its expression list in the logical request generator.

  The query cache uses the resulting translated expression for cache hit detection.
- **Usage Tracking.** Usage tracking inserts the Logical SQL query string with the `INDEXCOL` function.

- **Security.** As long as the user has the privileges to access the columns in the expression translated from INDEXCOL, then the query executes.

  When the first argument to INDEXCOL is a session variable and if a default expression is expected to be returned even if the initialization block fails, then you should set a default value for the session variable. Otherwise, the query fails because the session variable has no value definition.

Syntax

```
INDEXCOL(integer_literal, expr_list)
```

Where:

expr_list equals the following:

```
expr1 [, expr_list ]
```

The INDEXCOL function takes in an integer literal value as its first argument, followed by a variable length expression list and translates to a single expression from the expression list. The literal value is the 0-based index of the expression in the expression list to translate to. Consider the following expression:

```
INDEXCOL(integer_literal, expr1, expr2, …)
```

If the literal value is 0, the above expression is equivalent to *expr1*. If the literal value is 1, then the value is equivalent to *expr2*, and so on.

The primary use case for INDEXCOL is for the first argument to contain a session variable. Specifying a constant literal would result in INDEXCOL always choosing the same expression.

Example With Hierarchy Levels

Company ABC has a geography dimension with the hierarchy Country, State, City. The CEO can access the Country level down to the City level, and the sales manager can access the State and City levels, and the sales people can only access the City level. Table C–2 shows the back-end database for Company ABC.

Table C–2 IndexCol Example of Back-End Database

USER_NAME	TITLE	GEO_LEVEL	CURRENCY	CURRENCY_COL
Bob	CEO	0	US Dollars	0
Harriet	Sales Manager	1	Japanese Yen	1
Jackson	Sales Manager	1	Japanese Yen	1
Mike	Sales Person	2	Japanese Yen	1
Jim	Sales Person	2	US Dollars	0

The following steps illustrate one way to create a single query where each user sees the top level to which they have access:

- The administrator creates a new session variable called GEOOGRAPHY_LEVEL that is populated by the following initialization block: `SELECT GEO_LEVEL from T where USER_NAME = ':USER'`.

  This assumes that the Oracle BI Server instance has the same user names.

- Using `SELECT INDEXCOL(VALUEOF(NQ_SESSION.GEOGRAPHY_LEVEL), Country, State, City), Revenue FROM Sales`, the following occurs:

- Bob logs in and INDEXCOL translates to the Country column because the GEOGRAPHY_LEVEL session variable is 0. He gets the same result and can drill down on Country to State as if he had used SELECT Country, Revenue FROM Sales.
- Jackson logs in and INDEXCOL translates to the State column because the GEOGRAPHY_LEVEL session variable for Jackson is 1. He gets the same result and can drill down on State to City as if he had used SELECT State, Revenue FROM Sales.
- Mike logs in and INDEXCOL translates to the City column because the GEOGRAPHY_LEVEL session variable for Mike is 2. He gets the same result and cannot drill down on City as if he had used SELECT City, Revenue FROM Sales.

TO_DATETIME

This function converts string literals of dateTime format to a DateTime data type.

Syntax

```
TO_DATETIME('string1', 'DateTime_formatting_string')
```

Where:

string1 is the string literal you want to convert

DateTime_formatting_string is the DateTime format you want to use, such as yyyy.mm.dd hh:mi:ss. For this argument, yyyy represents year, mm represents month, dd represents day, hh represents hour, mi represents minutes, and ss represents seconds.

Examples

```
SELECT TO_DATETIME('2009-03-03 01:01:00', 'yyyy-mm-dd hh:mi:ss') FROM
snowflakesales

SELECT TO_DATETIME('2009.03.03 01:01:00', 'yyyy.mm.dd hh:mi:ss') FROM
snowflakesales
```

VALUEOF

Use the VALUEOF function to reference the value of a repository variable. Repository variables are defined using the Administration Tool. You can use the VALUEOF function both in Expression Builder in the Administration Tool, and when you edit the SQL statements for an analysis from the Advanced tab of the Analysis editor in Answers.

Syntax

Variables should be used as arguments of the VALUEOF function. Refer to static repository variables by name. Note that variable names are case sensitive. For example, to use the value of a static repository variables named prime_begin and prime_end:

```
CASE WHEN "Hour" >= VALUEOF("prime_begin")AND "Hour" < VALUEOF("prime_end") THEN
'Prime Time' WHEN ... ELSE...END
```

You must refer to a dynamic repository variable by its fully qualified name. If you are using a dynamic repository variable, the names of the initialization block and the repository variable must be enclosed in double quotes ("), separated by a period, and

contained within parentheses. For example, to use the value of a dynamic repository variable named REGION contained in an initialization block named Region Security, use the following syntax:

```
SalesSubjectArea.Customer.Region = VALUEOF("Region Security"."REGION")
```

The names of session variables must be preceded by NQ_SESSION, separated by a period, and contained within parentheses, including the NQ_SESSION portion. For example, to use the value of a session variable named REGION, use the following syntax in Expression Builder or a filter:

```
"SalesSubjectArea"."Customer"."Region" = VALUEOF(NQ_SESSION.REGION)
```

If the variable name contains or begins with any characters other than ASCII alphanumeric characters ([A-Z] [a-z] [0-9]) or an underscore (_), then enclose the name in double quotes ("). Examples of such characters include a space, single quote, or double quote. The following example contains a single quote and a space:

```
"SalesSubjectArea"."Customer"."Region" = VALUEOF("NQ_SESSION"."Steven's Regions")
```

If the variable name that you specify contains double quotes ("), then escape the double quotes with a set of double quotes. For example:

```
"SalesSubjectArea"."Customer"."Region" = VALUEOF("NQ_SESSION"."""Top Sales""
Region")
```

Although using initialization block names with session variables (just as with other repository variables) may work, you should use NQ_SESSION. NQ_SESSION acts like a wildcard that matches all initialization block names. This lets you change the structure of the initialization blocks in a localized manner without impacting requests.

VARIABLE_REPLACE

This function queries a measure with the session variable value specified. Use this function to query the measure using one or more session variable values. You can specify multiple name-value pairs. As long as the user has the permission to set the session variable, then the query executes.

You can only use this function when entering SQL at the command line using the nqcmd utility.

Syntax

```
VARIABLE_REPLACE(measure, variable_name, variable_value [, variable_name1,
variable_value1, ...])
```

Where:

measure is a measure fact column that has a logical table source mapping containing session variables.

variable_name is a session variable name with the prefix NQ_SESSION.

variable_value is the substitute value for the session variable. String and integer are supported values. If the variable name contains a space, then it must be enclosed in double quotes.

Examples

This example shows the session variable being replaced with a string literal 'A'.

```
VARIABLE_REPLACE(Revenue, 'NQ_SESSION.VAR', 'A')
```

This example shows the session variable being replaced with the integer 1.

```
VARIABLE_REPLACE(Revenue, 'NQ_SESSION.VAR', 1)
```

Lookup Functions

Multilingual schemas typically store translated fields in separate tables called lookup tables. Lookup tables contain translations for descriptor columns in several languages, while the base tables contain the data in the base language. *Lookup* is when a query joins the base table and lookup table to obtain the translated values for each row in the base table. A `LOOKUP` function is typically used in the Business Model and Mapping layer as an expression in a translated logical table column.

See the following sections in *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for full information:

- "Supporting Multilingual Data"
- "About the LOOKUP Function Syntax"

Database Functions

Users and administrators can create requests by directly calling database functions from either Oracle BI Answers, or by using a logical column (in the logical table source) within the metadata repository. Key uses for these functions include the ability to pass through expressions to get advanced calculations, as well as the ability to access custom written functions or procedures on the underlying database.

You cannot use these functions with XML data sources.

Note the following:

- The EVALUATE_SUPPORT_LEVEL parameter in NQSConfig.INI controls the use of the EVALUATE family of database functions within Oracle BI Answers. Oracle recommends leaving EVALUATE_SUPPORT_LEVEL set to its default value of `0` to prevent the use of these functions within Oracle BI Answers. Setting EVALUATE_SUPPORT_LEVEL to a value of `1` or `2` enables users to insert arbitrary SQL expressions into an analysis using Oracle BI Answers, which potentially compromises data access security. See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information about the EVALUATE_SUPPORT_LEVEL parameter.
- The EVALUATE_SUPPORT_LEVEL parameter in NQSConfig.INI does not control use of the EVALUATE family of database functions within the metadata repository.

Functions include:

- EVALUATE
- EVALUATE_ANALYTIC
- EVALUATE_AGGR
- EVALUATE_PREDICATE

EVALUATE

This function, which is for scalar functions that take input values and return an output value for a single row, passes the specified database function with optional referenced columns as parameters to the back-end data source for evaluation. This function is intended for scalar calculations, and is useful when you want to use a specialized database function that is not supported by the Oracle BI Server, but that is understood by the underlying data source.

The embedded database function may require one or more columns. These columns are referenced by %1 ... %N within the function. The actual columns must be listed after the function.

The ability to use EVALUATE is disabled by default. To enable support for this function, change the EVALUATE_SUPPORT_LEVEL parameter in NQSConfig.INI. See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

Syntax

```
EVALUATE('db_function(%1...%N)' [AS data_type] [, column1, columnN])
```

Where:

db_function is any valid database function understood by the underlying data source.

data_type is an optional parameter that specifies the data type of the return result. Use this parameter whenever the return data type cannot be reliably predicted from the input arguments. However, do not use this parameter for type casting; if the function needs to return a particular data type, add an explicit cast. You can typically omit this parameter when the database-specific function has a return type not supported by the Oracle BI Server, but is used to generate an intermediate result that does not need to be returned to the Oracle BI Server.

column1 through *columnN* is an optional, comma-delimited list of column names.

Examples

This example shows an embedded database function.

```
SELECT EVALUATE('instr(%1, %2)', address, 'Foster City') FROM employees
```

Examples Using EVALUATE_AGGREGATE and EVALUATE to Leverage Unique Essbase Functions

The following examples use the EVALUATE_AGGREGATE and EVALUATE functions. Note that expressions are applied to columns in the logical table source that refers to the physical cube.

Use EVALUATE_AGGREGATE to implement custom aggregations. For example, you may want to compare overall regional profit to profits for the top three products in the region. You can define a new measure to represent the profits for top three products resulting in the Logical SQL statement:

```
SELECT Region, Profit, EVALUATE_AGGREGATE('SUM(TopCount(%1.members, 3, %2), %3)',
Products, Profit, Profit) Top_3_prod_Profit FROM SampleBasic
```

The Oracle BI Server generates the following expression for the custom aggregation:

```
member [Measures].[MS1] AS
'SUM(Topcount([Product].Generations(6).members,3,[Measures].[Profit]),[Measures].[
Profit])'
```

Use the EVALUATE function on projected dimensions to implement scalar functions that are computed post-aggregation. EVALUATE may change the grain of the query, if its definition makes explicit references to dimensions (or attributes) that are not in the query.

For example, if you would like to see the Profits for the top five products ranked by Sales sold in a Region, after creating the applicable measure, the resulting Logical SQL statement is as follows

```
SELECT Region, EVALUATE('TopCount(%1.members, 5, %2)' as VARCHAR(20), Products,
Sales), Profits
FROM SampleBasic
```

The Oracle BI Server generates the following expression to retrieve the top five products:

```
set [Evaluate0] as
'{Topcount([Product].Generations(6).members,5,[Measures].[Sales]) }'
```

EVALUATE_ANALYTIC

This function, which takes a row set of one or more rows and returns results for each row in the set, passes the specified database analytic function with optional referenced columns as parameters to the back-end data source for evaluation. Its functions are used to model SQL analytic functions, also known as window functions.

The embedded database function may require one or more columns. These columns are referenced by %1 ... %N within the function. The actual columns must be listed after the function.

The ability to use EVALUATE_ANALYTIC is disabled by default. To enable support for this function, change the EVALUATE_SUPPORT_LEVEL parameter in NQSConfig.INI. See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

Syntax

```
EVALUATE_ANALYTIC('db_function(%1...%N)' [AS data_type] [, column1, columnN])
```

Where:

db_function is any valid database analytic function understood by the underlying data source.

data_type is an optional parameter that specifies the data type of the return result. Use this parameter whenever the return data type cannot be reliably predicted from the input arguments. However, do not use this parameter for type casting; if the function needs to return a particular data type, add an explicit cast. You can typically omit this parameter when the database-specific analytic function has a return type not supported by the Oracle BI Server, but is used to generate an intermediate result that does not need to be returned to the Oracle BI Server.

column1 through *columnN* is an optional, comma-delimited list of columns.

Examples

This example shows an embedded database analytic function.

```
EVALUATE_ANALYTIC('dense_rank() over(order by %1 )' AS INT,sales.revenue)
```

If the preceding example needs to return a double, then an explicit cast should be added, as follows:

```
CAST(EVALUATE_ANALYTIC('Rank(%1.dimension.currentmember, %2.members)',
"Foodmart93"."Time"."Month" as Double)
```

EVALUATE_AGGR

This function passes the specified database function with optional referenced columns as parameters to the back-end data source for evaluation. This function is intended for aggregate functions with a `GROUP BY` clause.

The embedded database function may require one or more columns. These columns are referenced by %1 ... %N within the function. The actual columns must be listed after the function.

The ability to use `EVALUATE_AGGR` is disabled by default. To enable support for this function, change the EVALUATE_SUPPORT_LEVEL parameter in NQSConfig.INI. See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

Syntax

```
EVALUATE_AGGR('db_agg_function(%1...%N)' [AS data_type] [, column1, columnN)
```

Where:

db_agg_function is any valid aggregate database function understood by the underlying data source.

data_type is an optional parameter that specifies the data type of the return result. Use this parameter whenever the return data type cannot be reliably predicted from the input arguments. However, do not use this parameter for type casting; if the function needs to return a particular data type, add an explicit cast. You can typically omit this parameter when the database-specific function has a return type not supported by the Oracle BI Server, but is used to generate an intermediate result that does not need to be returned to the Oracle BI Server.

column1 through *columnN* is an optional, comma-delimited list of columns.

Example

```
EVALUATE_AGGR('REGR_SLOPE(%1, %2)', sales.quantity, market.marketkey)
```

EVALUATE_PREDICATE

This function passes the specified database function with optional referenced columns as parameters to the back-end data source for evaluation. This function is intended for functions with a return type of Boolean.

The embedded database function may require one or more columns. These columns are referenced by %1 ... %N within the function. The actual columns must be listed after the function.

Note that `EVALUATE_PREDICATE` is not supported for use with Essbase data sources.

The ability to use `EVALUATE_PREDICATE` is disabled by default. To enable support for this function, change the EVALUATE_SUPPORT_LEVEL parameter in NQSConfig.INI. See *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for more information.

Syntax

```
EVALUATE_PREDICATE('db_function(%1...%N)', [, column1, columnN)
```

Where:

db_function is any valid database function with a return type of Boolean that is understood by the underlying data source.

column1 through *columnN* is an optional, comma-delimited list of columns.

If you want to model a database function for comparison purposes, you should not use EVALUATE_PREDICATE. Instead, use EVALUATE and put the comparison outside the function. For example, do *not* use EVALUATE_PREDICATE as follows:

```
EVALUATE_PREDICATE('dense_rank() over (order by 1% ) < 5', sales.revenue)
```

Instead, use EVALUATE, as follows:

```
EVALUATE('dense_rank() over (order by 1% ) ', sales.revenue) < 5
```

Example

```
SELECT year, Sales AS DOUBLE,CAST(EVALUATE('OLAP_EXPRESSION(%1,''LAG(units_cube_
sales, 1, time, time LEVELREL time_levelrel)'')', OLAP_CALC) AS DOUBLE) FROM
"Global".Time, "Global"."Facts - sales" WHERE EVALUATE_PREDICATE('OLAP_
CONDITION(%1, ''LIMIT time KEEP ''''1'''', ''''2'''', ''''3'''', ''''4'''' '')
=1', OLAP_CALC) ORDER BY year;
```

Hierarchy Navigation Functions

The hierarchy navigation functions enable you to identify relationships between members of hierarchies. The hierarchy navigation functions include:

- DEPTH
- IDOF
- ISANCESTOR
- ISCHILD
- ISDESCENDANT
- ISLEAF
- ISPARENT
- ISROOT
- ISSIBLING
- PARENT

The IDOF and ISLEAF functions apply to both level-based and parent-child hierarchies, while the other functions apply only to parent-child hierarchies.

See Chapter 10, "Working with Logical Dimensions" for information about level-based and parent-child hierarchies, including information about creating parent-child relationship tables (closure tables) for relational sources.

DEPTH

The DEPTH function returns an integer greater than 0 that indicates the depth of the member from the root member. The depth of a root member is 1.

Syntax

```
DEPTH(pc_presentation_hierarchy)
```

Where *pc_presentation_hierarchy* identifies the fully qualified parent-child presentation hierarchy, as follows:

```
"subject_area"."presentation_table"."pc_presentation_hierarchy"
```

The qualification term (`"subject_area".`) is optional unless there are multiple presentation tables or presentation hierarchies with the same name in different subject areas.

Example

```
DEPTH("employees"."emp_hierarchy")
```

IDOF

The `IDOF` function returns either a representation of the member key column values for a given member in a parent-child hierarchy, or a representation of the level key column values for a given level in a level-based hierarchy.

If the given hierarchy or level has a key that contains exactly one logical column, the function returns the value of that logical column. If the given level has a key that contains more than one logical column, the function returns a concatenated string. For example:

"keycolumn1value"."keycolumn2value"."keycolumn3value"…

Note that parent-child hierarchies do not have keys that contain more than one logical column.

If the given hierarchy or level has no key, an error is displayed.

Syntax

For parent-child hierarchies:

```
IDOF(pc_presentation_hierarchy)
```

Where *pc_presentation_hierarchy* identifies the fully qualified parent-child presentation hierarchy, as follows:

```
"subject_area"."presentation_table"."pc_presentation_hierarchy"
```

For level-based hierarchies:

```
IDOF(level)
```

Where *level* is a fully-qualified presentation level that is based on a level in a level-based dimension, as follows:

```
"subject_area"."presentation_table"."presentation_hierarchy"."presentation_level"
```

Examples

```
IDOF("hr"."employees"."emp_hierarchy")

IDOF("market_data"."products"."product"."product")
```

ISANCESTOR

The `ISANCESTOR` function enables you to find the ancestors of one or more members of a parent-child hierarchy, either all the ancestors of the members, or the ancestors at a specified hierarchical distance from the members.

Each member of the parent-child hierarchy is compared with the specified members to determine if it is an ancestor. The ISANCESTOR function returns the Boolean value True for each ancestor of the specified members, else it returns False.

When you use the function to find ancestors of multiple members, then an OR operation is performed. That is, find the ancestors of Joe or Juana. If you need an AND operation, then call the ISANCESTOR function multiple times, once for each member.

You can use the ISANCESTOR function in a query both within CASE statements and in WHERE clause conditions.

You can use the ISANCESTOR function in both Presentation layer queries, and in the Business Model and Mapping layer (for example, when creating a derived column). Note that the syntax of the function depends on where you are using it.

Presentation Layer Syntax

```
ISANCESTOR(pc_presentation_hierarchy, member_identifiers [, distance])
```

Where:

pc_presentation_hierarchy identifies the fully qualified parent-child presentation hierarchy, as follows:

```
"subject_area"."presentation_table"."pc_presentation_hierarchy"
```

The qualification term (`"subject_area".`) is optional unless there are multiple presentation tables or presentation hierarchies with the same name in different subject areas.

member_identifiers is the string or numeric literals that identify the one or more members in *pc_presentation_hierarchy*. Separate multiple literals with commas and enclose the group in parentheses, such as (2, 3). The type of literal depends on the data type of the dimension level keys.

distance (optional) is a positive integer that identifies the distance from the specified members to the parent-child hierarchy level at which to search for ancestors.

By default, if *distance* is not specified, the ISANCESTOR function searches the current parent-child level that contains *member_identifiers* and all levels above.

Business Model and Mapping Layer Syntax

```
ISANCESTOR(logical_dimension, member_identifiers [, distance])
```

Where:

logical_dimension identifies the fully qualified dimension that contains the parent-child hierarchy, as follows:

```
"business_model"."dimension_name"
```

The qualification term (`"business_model".`) is optional unless there are multiple dimensions with the same name in different business models.

member_identifiers is the string or numeric literals that identify the one or more members in *logical_dimension*. Separate multiple literals with commas and enclose the group in parentheses, such as (2, 3). The type of literal depends on the data type of the dimension level keys.

distance (optional) is a positive integer that identifies the distance from the specified members to the parent-child hierarchy level at which to search for ancestors.

By default, if *distance* is not specified, the ISANCESTOR function searches the current parent-child level that contains *member_identifiers* and all levels above.

Example

The following example selects all the ancestor employees of the employee Joe in a parent-child hierarchy. The returned list includes the employee Joe.

```
SELECT emp_name
FROM "employees"
WHERE ISANCESTOR("employees"."emp_hierarchy", 'Joe')
```

ISCHILD

The ISCHILD function enables you to find the children of one or more members of a parent-child hierarchy, that is, all the members that are one hierarchical level below the specified members.

> **Note:** The ISCHILD function is the same as the ISDESCENDANT function with a distance parameter of 1.

The ISCHILD function returns the Boolean value True for each child of the specified members, else it returns False.

When you use the function to find children of multiple members, then an OR operation is performed. That is, find the children of Joe or Juana. If you need an AND operation, then call the ISCHILD function multiple times, once for each member.

You can use the ISCHILD function in a query both within CASE statements and in WHERE clause conditions.

You can use the ISCHILD function in both Presentation layer queries, and in the Business Model and Mapping layer (for example, when creating a derived column). Note that the syntax of the function depends on where you are using it.

Presentation Layer Syntax

```
ISCHILD(pc_presentation_hierarchy, member_identifiers)
```

Where:

pc_presentation_hierarchy identifies the fully qualified parent-child presentation hierarchy, as follows:

```
"subject_area"."presentation_table"."pc_presentation_hierarchy"
```

The qualification term (`"subject_area".`) is optional unless there are multiple presentation tables or presentation hierarchies with the same name in different subject areas.

member_identifiers is the string or numeric literals that identify the one or more members in *pc_presentation_hierarchy*. Separate multiple literals with commas and enclose the group in parentheses, such as (2, 3). The type of literal depends on the data type of the dimension level keys.

Business Model and Mapping Layer Syntax

```
ISCHILD(logical_dimension, member_identifiers)
```

Where:

logical_dimension identifies the fully qualified dimension that contains the parent-child hierarchy, as follows:

```
"business_model"."dimension_name"
```

The qualification term (`"business_model".`) is optional unless there are multiple dimensions with the same name in different business models.

member_identifiers is the string or numeric literals that identify the one or more members in *logical_dimension*. Separate multiple literals with commas and enclose the group in parentheses, such as (2, 3). The type of literal depends on the data type of the dimension level keys.

Example

The following example selects all the children of the employee Joe in a parent-child hierarchy.

```
SELECT emp_name
FROM "employees"
WHERE ISCHILD("employees"."emp_hierarchy", 'Joe')
```

ISDESCENDANT

The `ISDESCENDANT` function enables you to find the descendants of one or more members of a parent-child hierarchy, either all the descendants of the members, or the descendants at a specified hierarchical distance from the members.

Each member of the parent-child hierarchy is compared with the specified members to determine if it is a descendant. The `ISDESCENDANT` function returns the Boolean value True for each descendant of the specified members, else it returns False.

When you use the function to find descendants of multiple members, then an OR operation is performed. That is, find the descendants of Joe or Juana. If you need an AND operation, then call the `ISDESCENDANT` function multiple times, once for each member.

You can use the `ISDESCENDANT` function in a query both within `CASE` statements and in `WHERE` clause conditions.

You can use the `ISDESCENDANT` function in both Presentation layer queries, and in the Business Model and Mapping layer (for example, when creating a derived column). Note that the syntax of the function depends on where you are using it.

Presentation Layer Syntax

```
ISDESCENDANT(pc_presentation_hierarchy, member_identifiers [, distance])
```

Where:

pc_presentation_hierarchy identifies the fully qualified parent-child presentation hierarchy, as follows:

```
"subject_area"."presentation_table"."pc_presentation_hierarchy"
```

The qualification term (`"subject_area".`) is optional unless there are multiple presentation tables or presentation hierarchies with the same name in different subject areas.

member_identifiers is the string or numeric literals that identify the one or more members in *pc_presentation_hierarchy*. Separate multiple literals with commas and

enclose the group in parentheses, such as (2, 3). The type of literal depends on the data type of the dimension level keys.

distance (optional) is a positive integer, that identifies the distance from the specified members to the parent-child hierarchy level at which to search for descendants.

By default, if *distance* is not specified, the ISDESCENDANT function searches the current parent-child level that contains *member_identifiers* and all levels below.

Business Model and Mapping Layer Syntax

```
ISDESCENDANT(logical_dimension, member_identifiers [, distance])
```

Where:

logical_dimension identifies the fully qualified dimension that contains the parent-child hierarchy, as follows:

```
"business_model"."dimension_name"
```

The qualification term ("business_model".) is optional unless there are multiple dimensions with the same name in different business models.

member_identifiers is the string or numeric literals that identify the one or more members in *logical_dimension*. Separate multiple literals with commas and enclose the group in parentheses, such as (2, 3). The type of literal depends on the data type of the dimension level keys.

distance (optional) is a positive integer that identifies the distance from the specified members to the parent-child hierarchy level at which to search for descendants.

By default, if *distance* is not specified, the ISDESCENDANT function searches the current parent-child level that contains *member_identifiers* and all levels below.

Example

The following example selects all the descendant employees of the employee Joe in a parent-child hierarchy. The returned list includes the employee Joe.

```
SELECT emp_name
FROM "employees"
WHERE ISDESCENDANT("employees"."emp_hierarchy", 'Joe')
```

ISLEAF

The ISLEAF function applies to both level-based and parent-child hierarchies. For both types of hierarchy, a leaf member is defined as a member that has no child members.

Each member of the hierarchy is examined to determine if it is a leaf member. The ISLEAF function returns the Boolean value True for each leaf member, else it returns False.

You can use the ISLEAF function in a query both within CASE statements and in WHERE clause conditions.

You can use the ISLEAF function in both Presentation layer queries, and in the Business Model and Mapping layer (for example, when creating a derived column). Note that the syntax of the function depends on where you are using it.

Presentation Layer Syntax

```
ISLEAF(presentation_hierarchy)
```

Where:

presentation_hierarchy identifies the fully qualified presentation hierarchy, either level-based or parent-child, as follows:

```
"subject_area"."presentation_table"."presentation_hierarchy"
```

The qualification term (`"subject_area".`) is optional unless there are multiple presentation tables or presentation hierarchies with the same name in different subject areas.

Business Model and Mapping Layer Syntax

```
ISLEAF(logical_dimension)
```

Where:

logical_dimension identifies the fully qualified dimension that contains the hierarchy you want to navigate, either level-based or parent-child, as follows:

```
"business_model"."dimension_name"
```

The qualification term (`"business_model".`) is optional unless there are multiple dimensions with the same name in different business models.

Example

The following example selects all the employees in a hierarchy that are leaf members, that is, the employees who have no members below them in the hierarchy.

```
SELECT emp_name
FROM "employees"
WHERE ISLEAF("employees"."emp_hierarchy")
```

ISPARENT

The `ISPARENT` function enables you to find the parents of one or more members of a parent-child hierarchy, that is, all the members that are one hierarchical level above the specified members.

> **Note:** The `ISPARENT` function is the same as the `ISANCESTOR` function with a distance parameter of 1.

The `ISPARENT` function returns the Boolean value True for each parent of the specified members, else it returns False.

When you use the function to find parents of multiple members, then an OR operation is performed. That is, find the parents of Joe or Juana. If you need an AND operation, then call the `ISPARENT` function multiple times, once for each member.

You can use the `ISPARENT` function in a query both within `CASE` statements and in `WHERE` clause conditions.

You can use the `ISPARENT` function in both Presentation layer queries, and in the Business Model and Mapping layer (for example, when creating a derived column). Note that the syntax of the function depends on where you are using it.

Presentation Layer Syntax

```
ISPARENT(pc_presentation_hierarchy, member_identifiers)
```

Where:

pc_presentation_hierarchy identifies the fully qualified parent-child presentation hierarchy, as follows:

```
"subject_area"."presentation_table"."pc_presentation_hierarchy"
```

The qualification term (`"subject_area".`) is optional unless there are multiple presentation tables or presentation hierarchies with the same name in different subject areas.

member_identifiers is the string or numeric literals that identify the one or more members in *pc_presentation_hierarchy*. Separate multiple literals with commas and enclose the group in parentheses, such as (2, 3). The type of literal depends on the data type of the dimension level keys.

Business Model and Mapping Layer Syntax

```
ISPARENT(logical_dimension, member_identifiers)
```

Where:

logical_dimension identifies the fully qualified dimension that contains the parent-child hierarchy, as follows:

```
"business_model"."dimension_name"
```

The qualification term (`"business_model".`) is optional unless there are multiple dimensions with the same name in different business models.

member_identifiers is the string or numeric literals that identify the one or more members in *logical_dimension*. Separate multiple literals with commas and enclose the group in parentheses, such as (2, 3). The type of literal depends on the data type of the dimension level keys.

Example

The following example selects all the parents of the employee Joe in a parent-child hierarchy.

```
SELECT emp_name
FROM "employees"
WHERE ISPARENT("employees"."emp_hierarchy", 'Joe')
```

ISROOT

A presentation hierarchy member is defined as a root member if it has no ancestors above it in a parent-child presentation hierarchy.

Each member of the parent-child hierarchy is examined to determine if it is a root member. The `ISROOT` function returns the Boolean value True for each root member, else it returns False.

You can use the `ISROOT` function in a query both within `CASE` statements and in `WHERE` clause conditions.

You can use the `ISROOT` function in both Presentation layer queries, and in the Business Model and Mapping layer (for example, when creating a derived column). Note that the syntax of the function depends on where you are using it.

Presentation Layer Syntax

```
ISROOT(pc_presentation_hierarchy)
```

Where:

pc_presentation_hierarchy identifies the fully qualified parent-child presentation hierarchy, as follows:

```
"subject_area"."presentation_table"."pc_presentation_hierarchy"
```

The qualification term (`"subject_area".`) is optional unless there are multiple presentation tables or presentation hierarchies with the same name in different subject areas.

Business Model and Mapping Layer Syntax

```
ISROOT(logical_dimension)
```

Where:

logical_dimension identifies the fully qualified dimension that contains the parent-child hierarchy, as follows:

```
"business_model"."dimension_name"
```

The qualification term (`"business_model".`) is optional unless there are multiple dimensions with the same name in different business models.

Example

The following example selects all the employees in a parent-child hierarchy that are root members, that is, the employees who have no ancestors above them in the hierarchy.

```
SELECT emp_name
FROM "employees"
WHERE ISROOT("employees"."emp_hierarchy")
```

ISSIBLING

The `ISSIBLING` function enables you to find the siblings of one or more members of a parent-child hierarchy, that is, all the members that have the same parent as the specified members. The sibling members can be parents themselves of other members. Root members with null parents are siblings of each other.

The `ISSIBLING` function returns the Boolean value True for each sibling of the specified members, else it returns False.

When you use the function to find siblings of multiple members, then an OR operation is performed. That is, find the siblings of Joe or Juana. If you need an AND operation, then call the `ISSIBLING` function multiple times, once for each member.

You can use the `ISSIBLING` function in a query both within `CASE` statements and in `WHERE` clause conditions.

You can use the `ISSIBLING` function in both Presentation layer queries, and in the Business Model and Mapping layer (for example, when creating a derived column). Note that the syntax of the function depends on where you are using it.

Presentation Layer Syntax

```
ISSIBLING(pc_presentation_hierarchy, member_identifiers)
```

Where:

pc_presentation_hierarchy identifies the fully qualified parent-child presentation hierarchy, as follows:

```
"subject_area"."presentation_table"."pc_presentation_hierarchy"
```

The qualification term (`"subject_area".`) is optional unless there are multiple presentation tables or presentation hierarchies with the same name in different subject areas.

member_identifiers is the string or numeric literals that identify the one or more members in *pc_presentation_hierarchy*. Separate multiple literals with commas and enclose the group in parentheses, such as (2, 3). The type of literal depends on the data type of the dimension level keys.

Business Model and Mapping Layer Syntax

```
ISSIBLING(logical_dimension, member_identifiers)
```

Where:

logical_dimension identifies the fully qualified dimension that contains the parent-child hierarchy, as follows:

```
"business_model"."dimension_name"
```

The qualification term (`"business_model".`) is optional unless there are multiple dimensions with the same name in different business models.

member_identifiers is the string or numeric literals that identify the one or more members in *logical_dimension*. Separate multiple literals with commas and enclose the group in parentheses, such as (2, 3). The type of literal depends on the data type of the dimension level keys.

Example

The following example selects all the siblings of the employee Joe in a parent-child hierarchy.

```
SELECT emp_name
FROM "employees"
WHERE ISSIBLING("employees"."emp_hierarchy", 'Joe')
```

PARENT

The `PARENT` function returns the member key column value for the parent of a given member in a parent-child hierarchy.

If the given hierarchy has no key, an error is displayed.

Syntax

```
PARENT(pc_presentation_hierarchy)
```

Where *pc_presentation_hierarchy* identifies the fully qualified parent-child presentation hierarchy, as follows:

```
"subject_area"."presentation_table"."pc_presentation_hierarchy"
```

The qualification term (`"subject_area".`) is optional unless there are multiple presentation tables or presentation hierarchies with the same name in different subject areas.

Example

```
PARENT("employees"."emp_hierarchy")
```

System Functions

The system functions return values relating to the session. Functions include:

- USER
- DATABASE

USER

This function returns the user name for the Oracle BI repository to which you are logged on.

Syntax

```
USER()
```

DATABASE

This function returns the name of the default subject area.

Syntax

```
DATABASE()
```

D

Merge Rules

This appendix provides information about the merge rules and behavior of the Oracle BI repository merge process.

When you use the Merge Repository Wizard in the Oracle BI Administration Tool, the `patchrpd` utility, or publish changes to the network in a multiuser development environment, sophisticated rules determine how objects are merged. Some decisions about merging objects are made automatically by the system, while other decisions appear as prompts in the Define Merge Strategy screen.

This appendix contains the following topics:

About the Merge Process

There are three types of Oracle BI repository merges:

- Full merges (sometimes called upgrade merges) are typically used during development processes, when there are two different repositories that need to be merged. The Administration Tool provides a three-way merge feature that lets you merge two repositories that have both been derived from a third, original repository. Full merges can also be used to import objects from one repository into another. See "Performing Full Repository Merges" for more information.
- Patch merges are used when you are applying the differential between two versions of the same repository. For example, you might want to use a patch merge to apply changes from the development version of a repository to your production repository, or to upgrade your Oracle BI Applications repository. See "Performing Patch Merges" for more information.
- Multiuser development merges are used when you are publishing changes to projects using a multiuser development environment. See "About the Multiuser Development Merge Process" for more information.

The merge process typically involves three versions of an Oracle BI repository: the **original** repository, **modified** repository, and **current** repository. The original repository is the original unedited file (the parent repository), while the modified and current repository are the two changed files you want to merge. The current repository is the one currently open in the Administration Tool.

The original, modified, and current repository may mean different things, depending on your situation. For example:

- In a development-to-production scenario, you have an original parent file, a current file that contains the latest development changes, and a modified file that is the deployed copy of the original file.
- In an Oracle BI Applications repository upgrade scenario, the current file is the latest version of the repository shipped by Oracle, and the original file is the original repository shipped by Oracle. The modified file is the file that contains the customizations you made to the original file.

Note that patch merge can be used with both of these situations. In a patch merge, you open the current file and select the original file, then generate the patch. To apply the patch, you open the modified file and select the original file, then apply the patch. See "Performing Patch Merges" for more information.

Merge Rules and Behavior for Full Merges

For full merges, the following general rules are applied:

- It is assumed that you generally want to keep the changes in the modified repository. For example, if an object is added to or deleted from the modified repository, the object is added or deleted without prompting.
- If an object is added to or deleted from the current repository, the Merge Repository Wizard asks whether you want to keep the changes.

  In general, the Merge Repository Wizard tries to ensure that you have the minimum set of objects necessary to service your queries. During a merge, there might be objects introduced by the current repository that are not needed in the merged repository. To address this issue, the Merge Repository Wizard asks whether new Presentation layer objects in the current repository are needed in the final merged result. If you choose to keep the new presentation objects, all the dependent logical and physical objects are added as well. However, if you choose not to keep the new presentation objects, then the dependent logical and physical objects are not kept, because no queries will require the use of these objects. The Merge Repository Wizard discards these objects to ensure that the merged repository does not get populated with unused objects.
- If an object is added to or deleted from both repositories, the object is added or deleted without prompting. If the same object was added with slight differences in its properties, the Merge Repository Wizard asks which version of the properties you want to keep.
- If an object has been modified only in the current repository, or only in the modified repository, the change is kept. If the same object is modified in both the current and modified repository, and the changes are different, then there is a merge conflict. When conflicts occur, the Merge Repository Wizard asks you to choose which change you want to keep.
- Making a decision about one object can determine a whole set of decisions, depending on the object relationships involved. For example, if you choose to keep a presentation column that has been added to the current repository, then the associated presentation table and subject area must also be kept, along with the logical column, physical column, and other associated objects upon which it is based. Alternatively, if you choose not to keep a subject area that has been added to the current repository, then you are not prompted to choose whether to keep its

associated objects. Adding a join may require the addition of its base tables, while changing an expression may cause physical columns to be added.

- Object relationships can be interconnected through their properties. In addition to strings and numbers, the internal value of a property can be other repository objects. Because of this, a change to one object might cause a corresponding change to an interrelated object.

  For example, assume you change the data source of Init Block B from a connection pool to Custom Authenticator A. In addition to the data source property change to the initialization block object, a corresponding property change occurs in the custom authenticator object (because the value of the initialization block property for Custom Authenticator A is now Init Block B).

  Because the decisions made for these properties must be synchronized, if you select **Current** as the decision for the data source property of Init Block B, then the decision for the initialization block property of Custom Authenticator A will also be **Current**.

In the Merge Repository Wizard, any decisions that require user input are displayed on the Define Merge Strategy screen.

Special Merge Algorithms for Logical Table Sources and Other Objects

In addition to the general rules governing how objects are merged and which situations require prompting, there are special rules for certain types of objects and situations.

This section contains the following topics:

Merging Objects that Use the Vector Merge Algorithm

Some objects, such as levels, application roles, and object permissions, use a vector merge algorithm that determines the parent/child relationships between objects.

Objects that use the vector merge algorithm include:

- Levels in a dimension, levels associated with a logical column, and child levels
- Dimensions and tables in a logical display folder
- Aggregation content in a logical table source
- Security objects like user and application role membership and permissions
- Initialization block LDAP server settings and execution precedence

The Oracle BI Server determines the initial state of object relationships in each repository during the merge process. For example, the following list shows the different possibilities for object permissions and how they relate to users and application roles:

- M - Missing. The application role, user, or object is not present in the repository.
- D - Default. Permissions are inherited from the parent application role.

- Y - Yes. The permission is explicitly granted to the user or application role.
- N - No. The permission is explicitly denied to the user or application role.

The Merge Repository Wizard determines the appropriate relationship for the merged repository depending on the state of the object permission relationships in each repository. For example:

- For an original repository with a result of Y, a modified repository with a result of N, and a current repository with a result of M, the Merge Repository Wizard determines a result of N for the merged repository.
- For an original repository with a result of N, a modified repository with a result of Y, and a current repository with a result of M, the Merge Repository Wizard determines a result of Y for the merged repository.

Example D–1 provides a detailed explanation of how object relationships are merged for application role objects.

Example D–1 Vector Merge Example: Merging Application Roles

The following list shows the different possibilities for user and application role relationships:

- M - Missing. The application role or user is not present in the repository.
- Y - Yes. The application role or user is a member of the application role.
- N - No. The application role or user is not a member of the application role.

Table D–1 shows the merged result for different combinations of object relationships in the merging repositories.

Table D–1 Results for Merging Application Roles Based on Object Relationships

Original Repository	Modified Repository	Current Repository	Result
M	M	M	N[1]
M	M	Y	Y
M	M	N	N
M	Y	M	Y
M[2]	Y	Y	Y
M	Y	N	Y
M	N	M	N
M	N	Y	Y
M	N	N	N
Y	M	M	Y
Y	M	Y	Y
Y	M	N	N
Y	Y	M	Y
Y	Y	Y	Y
Y	Y	N	N
Y	N	M	N
Y	N	Y	N

Table D–1 (Cont.) Results for Merging Application Roles Based on Object Relationships

Original Repository	Modified Repository	Current Repository	Result
Y	N	N	N
N	M	M	N
N	M	Y	Y
N	M	N	N
N	Y	M	Y
N	Y	Y	Y
N	Y	N	Y
N	N	M	N
N	N	Y	Y
N	N	N	N

[1] This situation can happen if neither the application role nor the user are present in the original repository, but the user is present in the modified repository and the application role is present in the current repository. In this case, no membership can be assumed.

[2] M in original for this case implies that either the user or application role is not present. The missing object added in both cannot be considered the same object.

Merging Logical Table Sources

Special rules govern how to merge column mappings in logical table source objects. Each column mapping is merged individually. For each column, if the mapping has changed in either the modified or current repository, the change is kept. If the mapping has changed in both repositories, the Oracle BI Server attempts to merge the mappings automatically.

Note that the deletion of a column is not considered to be a change in its mapping. If a column is not present in the modified repository, then the mapping in the current repository is used instead.

If there are differences in aggregation content, then the aggregation content specified by level has priority. In other words, if the aggregation content in one repository is by level and the aggregation content in another repository is by column, then the aggregation content by level is retained.

Merging Security Filters

If a filter for an application role has changed in only one repository, then the change is kept. If the filter has changed in both repositories, the Oracle BI Server attempts to merge the filters automatically.

If an object is required for merging a particular filter (such as a presentation column) and is not present, then that filter is considered invalid and does not appear in the merged repository. Note, however, that this rule does not apply to variables. If a variable is required for merging a particular filter, the Oracle BI Server ensures that the variable is retained in the merged repository.

Inferring the Use Logical Column Property for Presentation Columns

Presentation columns have both a Name property and a Use Logical Column Name property. In some cases, these properties can come into conflict. For example, Table D–2 shows a scenario where this situation could occur.

Table D–2 Conflicting Presentation Column Name and Use Logical Column Name Properties

Repository	Presentation Column Name	Logical Column Name	Use Logical Column Name
Original	Sales	GroupSales	No
Current	Sales	Sales	Yes
Modified	GroupSales	GroupSales	Yes

If the regular merge rules for the objects in Table D–2 are applied, the merged repository contains a presentation column called GroupSales and a logical column called Sales, with the Use Logical Column Name property set to Yes. However, this result is incorrect, because the name of the presentation column is different from the name of the logical column.

To avoid this situation, the Oracle BI Server infers the value of the Use Logical Column Name property. Using this logic, the merged repository for the example in Table D–2 has a presentation column called GroupSales, a logical column called Sales, and a Use Logical Column Name property set to No.

Merging Aliases

During the full merge process, users are not prompted to make decisions about aliases. Aliases from the current and modified repositories are merged automatically.

In multiuser development merges, however, users are prompted to choose whether to keep aliases from the current repository, keep aliases from the modified repository, or merge choices to keep aliases from both repositories.

Also note the following:

- If object names change because of the merge process, then the previous names are added as aliases.
- Any aliases that are not associated with presentation objects are deleted.

Merge Rules and Behavior for Multiuser Development Merges

The rules for multiuser development merges are very similar to the full merge rules, with the following important differences:

- Changes to security settings are not retained when you perform a MUD merge to prevent developers from overwriting passwords and other important objects in the master repository.
- The database and connection pool properties in the master repository take precedence over the same properties on the developer's computer. This precedence are applied without a prompt during a multiuser development merge.
- Inserts (created objects) are applied automatically. Because a subset of the master repository is being used as the original repository, most objects in the master repository appear to be new. This would result in many unnecessary prompts that the developer would have to manually approve. Therefore, new objects are created without a prompt during a multiuser development merge.
- Conflicts that are not inserts but are resolved because of the automatic inserts are applied without a prompt during a multiuser development merge.

To change security settings or database features in a multiuser development environment, you must edit the master repository directly. To do this, remove the master repository from the multiuser development directory, edit it in offline mode, then move it back.

Merge Rules and Behavior for Patch Merges

The rules for patch merges are also similar to the full merge rules, except that the behavior for deleting objects is different. For example, if an object is deleted in the current repository, the default behavior for patch merges is to always ask the user whether the object should be discarded or retained. This is different from full merges, which often accept deletions from the current repository without prompting.

Using Patchrpd to Automate the Patch Process

You can use the -U and -V options in the `patchrpd` command-line utility to automate the patching process. The -U option enables the patching process to complete by accepting default decisions for conflicts, while the -V option enables you to specify an output file for recording all merge conflicts so that you can examine them and possibly take action later.

Follow these guidelines to use `patchrpd` to automate the patch process:

- **Apply patching automatically using default rules.** Include the -U option in `patchrpd` to always apply the default decisions for conflicts. For example, if both repositories (current and modified) change the name of an object, the default decision is to keep the name in the modified repository, to avoid overwriting user customizations. If you do not include this parameter, `patchrpd` displays a warning and exits if a conflict is detected.
- **Record conflicts in an output decision file.** Include the -V option to cause `patchrpd` to generate a decision file that shows all the conflicts from the merge. The decision file lists the decisions that would have been displayed in the Define Merge Strategy screen of the Merge Wizard if the merge had been performed in the Administration Tool. The decision file provides a record of all items that can be influenced by user input.
- **Modify the decision file to change the result.** After you run `patchrpd`, there are two ways to make changes to the resulting repository:
  - You can use the Load Decision File button in the Define Merge Strategy screen of the Merge Wizard to load the merge decisions, and then change the decisions if needed. You can then complete the merge in the Administration Tool. Alternatively, you can save the modified decision file using the Save Decisions to File button, and then re-run `patchrpd` with the decision file as an input using the -D option to reapply the patch with the new decisions.
  - You can edit the decision file by hand, and then re-run `patchrpd` with the decision file as an input using the -D option to reapply the patch with the new decisions.

Follow these guidelines to set up `patchrpd` to match the Administration Tool's merge functionality:

- **Administration Tool's full merge -** Use the -A option (apply changes on the whole repository) and -M option (use upgrade mode).

- **Administration Tool's patch merge with the "use subset patching" option selected -** Do not include the -M option. Excluding the -M option means that the default merge mode will be set to patch.
- **Administration Tool's patch merge with the "use subset patching" option not selected -** Use the -A option and do not include the -M option. Excluding the -M option means that the default merge mode will be set to patch.

See "Using patchrpd to Apply a Patch" for full information about patchrpd usage and syntax.

E

Deleting Unwanted Objects from the Repository

This appendix explains how to use the command-line pruning utility, `prunerpd`, to delete unwanted objects in the Oracle BI repository.

Note that you can only use `prunerpd` with binary repositories in RPD format.

This appendix contains the following topics:

About the Object Pruning Utility

If you have a large number of extraneous or unwanted objects in your repository, you can delete the unwanted objects using the `prunerpd` command-line utility. You can use `prunerpd` on both Windows and UNIX systems.

You can delete unwanted repository objects such as databases, tables, columns, initialization blocks, and variables. However, note that the pruning utility does not remove objects from the Oracle BI Presentation Catalog.

Deleting objects from the repository has a cascading effect. For example, if a physical column is deleted, then any mapped logical columns are deleted, as well as any associated presentation columns. See "Deletion Rules for the Object Pruning Utility" for more information.

Using the Object Pruning Utility

You must first create the input file that contains the list of repository objects to be deleted. Then, you must run the utility at the command line, passing the input file as an argument

This section contains the following topics:

Creating the Input File

The prune utility accepts the list of repository objects you want to delete as a text file. The utility can accept multiple input files at a time. The syntax rules for the input file

are shown in Table E–1.

Note: Object names in the input file must match the fully qualified name that is used in the repository. Wildcards (such as "*" and "?") are not supported in the object name.

Table E–1 Syntax Rules for Input File

Object Type	Example	Action
Database	D "Paint"	Deletes the database named "Paint."
Table	▪ T "W_AGREE_D" ▪ T "DB"."Catalog"."Schema"."Table"	▪ Deletes the table or alias named "W_AGREE_D" from the Physical layer. ▪ Deletes the table or alias named "Table" from the schema named "Schema," contained in the catalog named "Catalog," located in the database named "DB," from the Physical layer.
Column	C "W_AGREE_MD"."AGREE_CD"	Deletes the column named "AGREE_CD" located in a table or alias named "W_AGREE_D" from the Physical layer.
Initialization block	I "External Metadata Strings"	Deletes the initialization block named "External Metadata Strings."
Variable	V CURR_USER	Deletes the variable named "CURR_USER."

For example, a text file that contains instructions to delete a database named "Stock Quotes" and a physical column named "S_NQ_ACCT"."USER_NAME" would include the following entry:

```
D "Stock Quotes" C "S_NQ_ACCT"."USER_NAME"
```

Use white space as a delimiter in the input file (a single space, tab, or multiple spaces).

Running the prunerpd Utility

The location of the `prunerpd` utility is:

ORACLE_HOME/user_projects/domains/bi/bitools/bin

Syntax

The `prunerpd` utility accepts the following parameters:

```
prunerpd -s source_rpd [-p rpd_password] -f input_file -o output_rpd -l output_
log_file -e error_log_file [-8]
```

Where:

source_rpd is the name and location of the target repository file.

rpd_password is the repository password for the source repository.

The password argument is optional. If you do not provide a password argument, you are prompted to enter a password when you run the command. To minimize the risk of security breaches, Oracle recommends that you do not provide a password

argument either on the command line or in scripts. Note that the password argument is supported for backward compatibility only, and will be removed in a future release. For scripting purposes, you can pass the password through standard input.

input_file is the input file name (in text format) that contains the list of repository objects to be removed. Separate multiple file names by spaces. Enclose spaces within a filename with double quotes (" ").

output_rpd is the name and location of the output repository file, also known as the pruned repository.

output_log_file is the name and location of the output log file. All actions performed on the repository are written to this file, including descriptions. The output log file is in XML format. Other messages, such as progress indicators, are sent to the standard output stream.

error_log_file is the name and location of the error log file. The pruning utility writes exceptions and errors to this log. The error log file is in XML format. Other errors are sent to the standard output error stream.

-8 specifies UTF-8 encoding.

Example

```
prunerpd -s C:/OBI/Server/Repository/BIApps.rpd
-f "C:/Remove Oracle EBS Objects.txt"
-o "C:/OBI/Server/Repository/BIApps Pruned.rpd"
-l "C:/temp/BIApps Prunning.log" -e "C:/temp/ BIApps Prunning.err"
Give password: my_repos_password
```

Deletion Rules for the Object Pruning Utility

Deleting repository objects has a cascading effect. This section describes the deletion rules.

Physical Layer Rules

- If a physical column or a table is deleted, then all of the affected keys, foreign keys, and complex joins are deleted as well. The internal obsolete attribute definition (attr defn) that links a logical column to a physical column is also removed.
- Empty schemas, catalogs, and databases are removed.
- If a table is deleted, then all its columns are deleted.

Logical Table Rules

- If a regular column (not an aggregate or derived column) is not mapped in any logical source, then it is deleted. The keys, including the level key and the logical key, are also removed.
- If the source column for a derived column or its referenced variable is deleted (corrupted), then the column is removed.
- If an aggregate rule or override aggregate rule for an aggregate column is corrupted (due to a logical column deletion), then the column is removed.
- If a logical table is removed (because its underlying physical table was deleted), then the keys, foreign keys, logical joins, sources, and source folder are removed.
- If a logical table source does not have any valid mapping, then it is deleted.

- If a logical table source is retained, but its aggregate content or filters are corrupted, then the corresponding expressions are set to null. The join specification is also removed.
- If a logical table, dimension, or business model is empty (contains no meaningful child), then it is deleted.

Presentation Layer Rules

- If a logical column is removed (because its underlying physical column was deleted), then any corresponding presentation columns are removed.
- If a presentation table or subject area does not contain children, then it is removed.

Security Rules

- If a security filter for a user or application role becomes corrupt due to deletion, then the filter is removed. If all filters are removed for a user or application role, then the internal privilege object is deleted.
- Even if all filters for an application role are deleted, the application role is still maintained.
- To remove an application role from the repository, you must explicitly delete it. See *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition* for information about deleting application roles.

Variable Rules

- Initialization blocks are deleted if the underlying connection pool is deleted.
- Repository and session variables are deleted if the associated initialization blocks are deleted.
- If a session variable is deleted and its parent initialization block does not contain variables, then the initialization block is removed.
- If an initialization block is deleted, then its variables are removed.

Marketing Rules

- Qualified list items are deleted if the associated cache catalog, GUID column, or qualified column is deleted.
- Target levels are deleted if the associated catalog (Segmentation Catalog name) is deleted.
- List catalogs are deleted if the associated catalog, table, or column is deleted.
- Conforming dimensions are deleted if the associated catalog, table, or column is deleted.

F

Data Types Supported by Oracle BI Enterprise Edition

This appendix lists and describes the data types supported by Oracle BI EE. This appendix contains information about the data type limitations, other Oracle BI Server limitations, and floating point limitations. It also contains instructions for using the Oracle BI Server `nqcmd` utility to run the `NQSGetSQLDataTypes` procedure to obtain information about the data types.

When you import metadata from a data source into the repository's physical layer, each column is assigned a data type. The data type is associated with a specific storage format, constraints, and a valid range of values.

This appendix contains the following topics:

Data Type Categories Supported by Oracle BI EE

This topic lists the data types by category (for example, numeric data and date data) that Oracle BI EE supports. See "Using the NQSGetSQLDataTypes Procedure to Access Data Type Information" and "Oracle BI EE Data Type Limitations" for information about specific data types.

Textual Data

Oracle BI EE supports the following textual data types:

- CHAR
- LONGVARCHAR
- VARCHAR

Numeric Data

Oracle BI EE supports the following numeric data types:

- BIGINT
- DECIMAL

- DOUBLE
- FLOAT
- INTEGER
- NUMERIC
- REAL
- SMALLINT
- TINYINT

Date and Time Data

Oracle BI EE supports the following data and time data types:

- DATE
- TIME
- TIMESTAMP

Binary Data

Oracle BI EE supports the following binary data types:

- BIT
- BINARY
- LONGVARBINARY
- VARBINARY

Using the NQSGetSQLDataTypes Procedure to Access Data Type Information

To access a list of data types supported by Oracle BI EE, use the Oracle BI Server utility `nqcmd` to run the `NQSGetSQLDataTypes` procedure. For example: `call NQSGetSQLDataTypes(0);`

When you run this procedure, the results contain a list of supported data types and information specific to each data type, such as case sensitivity and searchability.

See "Using nqcmd to Test and Refine the Repository" for more information about opening and using the Oracle BI Server `nqcmd` utility.

Oracle BI EE Data Type Limitations

The following table contains each supported data type and its limitations. An administrator or repository builder can use this information to evaluate whether a particular data type is suitable for a given column or set of values, and to determine whether the data type is capable of representing all the required values.

For example, the INTEGER column in the Oracle database supports a very large range of values (up to 38 decimal digits), but the INTEGER data type in Oracle BI EE is a 32-bit binary integer type, which is capable of holding up to nine digits without encountering data overflow (truncation) issues. If the column holds values in the range of [-2,147,483,648, 2,147,483,647], then you should use the Oracle BI EE

INTEGER data type. However, if the column stores values larger than this range, then you should use another data type like NUMERIC or even VARCHAR.

Choose the smallest (in bytes) data type that is capable of representing the column's expected range of values. Choosing a data type in this way reduces the amount of memory and disk space consumed by the BI Server for cache files, temp files, and so on.

Table F–1 Data Type Limitations

Data Type	Limitations
BIG INT	JDBC and the Administration Tool do not support this type; therefore, Oracle BI EE does not fully support the BIG INT type. The BIG INT type is intended to be same as the C int64 data type.
BINARY	Oracle BI EE does not fully support the BINARY type. BI Server supports only the fetching of columns whose data type is BINARY. The BI Server does not support the BINARY type in bind parameters or insert statements.
BIT	Oracle BI EE does not fully support the BIT type. Instead, you should use either the INT or CHAR type to represent Boolean data.
CHAR	The CHAR type's values are always padded with ending spaces that can equal up to the length specified by the data type. The CHAR type supports Unicode values. On the Windows platform, the storage is two bytes per character. On all Unix 64-bit platforms, the storage is four bytes per character.
DATE	The DATE type represents only year, month, and day components. Note that it does not represent hours, minutes, or seconds like the Oracle DATE data type.
DECIMAL	The DECIMAL type is the same as the NUMERIC type.
DOUBLE	The DOUBLE type is the same as the IEEE 754 64-bit double-precision binary floating-point data type. The internal storage is eight bytes. The significand occupies 53 bits (including the sign bit). Therefore, the precision is limited to approximately 16 decimal digits. The exponent occupies 11 bits. The range of the exponent is approximately ±307 as a base 10 decimal value. See "Floating Point Limitations" for more information about the limitations of this type.
INTEGER	The INTEGER type is a signed binary integer data type occupying four bytes. The maximum value that can be represented is 2,147,483,647, and the minimum value is -2,147,483,648.
FLOAT	The FLOAT type is the same as the IEEE 754 32-bit single-precision binary floating-point data type. The internal storage is four bytes. The significand occupies 24 bits (including the sign bit). Therefore, the precision is limited to approximately 7 decimal digits. The exponent occupies eight bits. The range of the exponent is approximately ±38 as a base 10 decimal value. See "Floating Point Limitations" for more information about the limitations of this type.
LONGVARBINARY	The LONGVARBINARY type supports up to 32,678 bytes.
LONGVARCHAR	The LONGVARCHAR type supports up to 32,678 bytes. Both the LONGVARCHAR type and the VARCHAR type support Unicode values.

Table F–1 (Cont.) Data Type Limitations

Data Type	Limitations
NUMERIC	The NUMERIC type is a true decimal data type occupying 22 bytes. The internal representation and limitations are the same as the Oracle NUMBER data type. The NUMERIC type supports positive numbers in the range of 1 x 10^-130 to 9.999...9 x 10^125 with up to 38 significant digits. The precision and scale are not stored in the repository. The scale is assumed to be 10.
REAL	The REAL type has the same description and limitations as the FLOAT type.
SMALLINT	The SMALLINT type is represented as the INTEGER type internally in the BI Server and has the same limitations as the INTEGER data type.
TIME	The TIME type represents only hour, minute, and second components.
TIMESTAMP	The TIMESTAMP type represents year, month, day, hour, minute, and second components. For some data sources on some platforms, it can also support fractions of a second.
TINYINT	The TINYINT type is represented as an INTEGER internally in BI Server. The TINYINT type and INTEGER type have the same limitations.
VARBINARY	The VARBINARY type is interchangeable with the LONGVARBINARY type. The VARBINARY type and the LONGVARBINARY type have the same limitations.
VARCHAR	The VARCHAR type is interchangeable with the LONGVARCHAR type. The VARCHAR type and LONGCARCHAR type have the same limitations. The Administration Tool allows users to enter a maximum character length of 2,147,483,647. However, the actual maximum length supported is 32,678.

Floating Point Limitations

Some numbers cannot be represented exactly with binary floating point data types such as FLOAT and DOUBLE. When converting decimal numbers to and from binary floating point representations, often there are rounding errors because of the representational limitations of binary floating point formats. For example, a decimal number such as 1.365 might be represented as 1.364999999999999 when converted to the DOUBLE type. When this number is rounded to 3 digits after the decimal point, the result is 1.365. However, if the number is rounded to 2 decimal digits, then the result is 1.36 and not 1.37.

Oracle BI Server supports the NUMERIC type for RDBMS and TimesTen data sources. To avoid the limitations of the FLOAT and DOUBLE types, Oracle suggests that you update the FLOAT and DOUBLE data types to the NUMERIC type. Note that other than switching to the NUMERIC data type, there is no workaround to fix the inherent limitations with binary floating point data types.

Other Oracle BI Server Limitations

In addition to the data type limitations, Oracle BI Server has the following limitations:

- The default maximum length of all fields in Oracle BI Server is 32,678 bytes. This default limit can be changed by setting the environment variable `OBIS_MAX_FIELD_SIZE`.
- The default maximum length of all SQL identifiers (for example, table names and column names) is 128 characters.

See "Oracle BI EE Data Type Limitations" for information about supported data types and their limitations.

Oracle Database to Oracle BI EE Data Type Mapping

When you import metadata from an Oracle database, the Administration Tool uses the mapping in Table F–2 to determine each imported column's corresponding Oracle BI Server data type. Note that how the data types map to the Oracle BI EE data types differs depending on the kind of database that you are using.

For more information about Oracle Database data types, see "Chapter 26 Oracle Data Types" in *Oracle Database Concepts*.

Table F–2 Data Type Mappings

Oracle Database Data Type	Oracle BI EE Data Type
CHAR	CHAR
NCHAR	CHAR
VARCHAR2	VARCHAR
NVARCHAR2	VARCHAR
NUMBER	NUMERIC if ENABLE_NUMERIC_DATA_TYPE = YES; otherwise, DOUBLE
NUMBER (precision, scale)	INT if scale = 0 and 1 <= precision <= 9; otherwise, same as NUMBER
BINARY_FLOAT	FLOAT
BINARY_DOUBLE	DOUBLE
DATE	DATETIME
TIMESTAMP	TIMESTAMP
TIMESTAMP WITH TIME ZONE	TIMESTAMP
TIMESTAMP WITH LOCAL TIME ZONE	TIMESTAMP
BLOB	LONGVARBINARY
CLOB	LONGVARCHAR
NCLOB	LONGVARCHAR
BFILE	Not supported
LONG	LONGVARCHAR
LONG RAW	Not supported
ROWID	CHAR
XML Type	LONGVARBINARY
UriType	Not supported

G

Exchanging Metadata with Databases to Enhance Query Performance

This appendix explains how to use Oracle Database or IBM DB2 to enhance the data warehouse performance and functionality of queries that run on the Oracle BI Server.

This appendix contains the following topics:

About Exchanging Metadata with Databases

By exchanging Oracle Business Intelligence metadata from the Oracle BI Server with your Oracle Database or IBM DB2 database, you enable the database to accelerate the performance of data warehouse queries.

You use the Oracle BI Server utility `sametaexport` to exchange the metadata. When you run this utility to generate cube views for DB2, the utility is called the DB2 Cube Views Generator. When you run this utility to generate metadata for Oracle Database, the utility is called the Oracle Database Metadata Generator.

The Oracle BI Server export utility works with the following tools:

- In the Oracle Database, the SQL Access Advisor creates materialized views and index recommendations on optimizing performance. Note that in database releases prior to 10g, this feature is called the Oracle Database Summary Advisor.
- In the IBM DB2 database, IBM DB2 Cube Views creates materialized query tables (MQTs).

The `sametaexport` utility generates the information necessary for the SQL Access Advisor or IBM DB2 Cube Views tool to preaggregate the relational data and improve query performance.

Generating the Import File

Both the Oracle Database Metadata Generator and the DB2 Cube Views Generator create the files that are needed to import metadata from the Oracle BI Server into the SQL Access Advisor or an IBM DB2 database.

This section contains the following topics that are common to the two generators:

Running the Generator

The Oracle Database Metadata Generator and the DB2 Cube Views Generator are invoked from the command line or embedded in a batch file. The command-line executable is named `sametaexport`.

The `sametaexport` utility is available on both Windows and UNIX systems. However, you can only use `sametaexport` with binary repositories in RPD format.

The location of the `sametaexport` utility is:

ORACLE_HOME/bi/bifoundation/server

```
sametaexport -r "PathAndRepositoryFileName" [-p repository_password]
-f "InputFileNameAndPath" [options]
```

Table G–1 contains descriptions of the parameters in the command-line executable file.

Table G–1 Parameters for sametaexport

Parameter	Definition	Additional Information
-r	Repository file name and full path	Quotation marks are required for the file name and path only if the file path is in long format or has spaces. Use the full path if the file is not in the current directory.
-p	Repository password	The password for the given repository. The password argument is optional. If you do not provide a password argument, you are prompted to enter a password when you run the command. To minimize the risk of security breaches, Oracle recommends that you do not provide a password argument either on the command line or in scripts. Note that the password argument is supported for backward compatibility only, and will be removed in a future release. For scripting purposes, you can pass the password through standard input.
-f	Input file name and full path	Quotation marks are required for the file name and path only if the file path is in long format or has spaces. Use the full path if the file is not in the current directory. You specify input files so that you do not have to type all the required information at the command line, and so that you can type international characters. See "About the Metadata Input File" for more information.

You can include some additional parameters in the input file or at the command line to change various defaults for the Oracle Database Metadata Generator and the DB2 Cube Views Generator. Parameters specified in the input file take precedence over parameters specified at the command line. You must include these parameters only if you want to change the default values.

Table G–2 and Table G–3 describe these optional parameters.

Table G–2 Optional Parameters and Defaults for the Oracle Database Metadata Generator

Parameter Definition	Additional Information	Input File Usage Example	Command Line Usage Example
Use schema name from RPD	When set to YES, the table schema names are used as they are used in the repository. The default value is YES.	USE_SCHEMA_NAME_ FROM_RPD = NO	-schemafrom rpd NO
Default schema name	The default schema name is used as the table schema name if the value of -schemafromrpd is set to NO, or if the repository schema name cannot be determined. The default value is SIEBEL.	DEFAULT_SCHEMA_ NAME = ORACLE	-defaultschema ORACLE
Oracle schema name	The metadata from Oracle Database Metadata Generator is created under this schema. The default value is SIEBEL.	ORA_DIM_SCHEMA_ NAME = ORACLE	-orclschema ORACLE
Logging enabled	Indicates whether to keep a log of the metadata export process. Valid values are ON, OFF, and DEBUG. The default value is ON.	LOGGING = DEBUG	-logging DEBUG
Log file name	The path to the log file. If you provide an invalid path, an error occurs. If you do not provide this parameter, the default log file path is used. The default path is: *ORACLE_INSTANCE*\diagnostics\logs\ OracleBIServerComponent\ coreapplication_obis*n*\ OraDimExp.log	LOG_FILE_NAME = C:\bea_default\instances \instance1\diagnostics\ logs\generator\logfile.log	-logfile C:\bea_ default\instances\ instance1\diagnostics\ logs\generator\logfile.l og

Table G–3 Optional Parameters and Defaults for the DB2 Cube Views Generator

Parameter Definition	Additional Information	Input File Usage Example	Command Line Usage Example
Distinct count supported	When set to YES, allows measure containing the DISTINCT_COUNT aggregation to be exported. The recommended setting and default value is NO.	DISTINCT_COUNT_ SUPPORTED = YES	-distinct YES
Statistical functions supported	When set to YES, allows measures containing the aggregation STDDEV to be exported. The recommended setting and default value is NO.	STATISTICAL_FUNCTIONS_ SUPPORTED = YES	-stat YES
Use schema name	When set to YES, the Cube Views metadata attributes have columns from tables under a schema name, which are then specified in the parameters. When set to NO, the schema names for these tables are empty. The default value is YES.	USE_SCHEMA_NAME = NO	-useschema NO
Use schema name from RPD	When set to YES, the table schema names are used as they are used in the repository. The default value is YES.	USE_SCHEMA_NAME_ FROM_RPD = NO	-schemafromrpd NO

Table G–3 (Cont.) Optional Parameters and Defaults for the DB2 Cube Views Generator

Parameter Definition	Additional Information	Input File Usage Example	Command Line Usage Example
Default schema name	The default schema name is used as the table schema name if the value of `-schemafromrpd` is set to `NO`, or if the repository schema name cannot be determined. The default value is `SIEBEL`.	DEFAULT_SCHEMA_NAME = ORACLE	-defaultschema ORACLE
Cube views schema name	The name of the schema under which the Cube Views metadata is created. The default value is `SIEBEL`.	CUBE_VIEWS_SCHEMA_ NAME = ORACLE	-cubeschema ORACLE
Log file name	The path to the log file. If you provide an invalid path, an error occurs. If you do not provide this parameter, the default log file path is used. The default path is: *ORACLE_INSTANCE*\diagnostics\logs\OracleBIServerComponent\coreapplication_obis*n*\CubeViews.log	LOG_FILE_NAME = C:\bea_default\instances\instance1\diagnostics\logs\generator\logfile.log	-logfile C:\bea_default\instances\instance1\diagnostics\logs\generator\logfile.log
Log failures	When set to `YES`, the log file lists the metadata that was invalidated under a certain rule. The default value is `YES`.	LOG_FAILURES = NO	-logfail NO
Log success	When set to `YES`, the log file lists the metadata that has been checked under each rule and has passed the check. The default value is `NO`.	LOG_SUCCESS = YES	-logsuccess YES

About the Metadata Input File

The input file is a text file that contains the parameters that are described in Table G–4.

Table G–4 Cube Metadata Input File Parameters

Input File Name	Description
BUSINESS_MODEL	The name of the business model in the logical layer of the Oracle Business Intelligence repository that contains the metadata to export. If the business model is not found in the repository, then an error message is displayed. You can only specify one business model name in the input file. To generate metadata for multiple business models, create another input file and run the Oracle Database Metadata Generator or DB2 Cube Views Generator again.
PHYSICAL_DATABASE	The name of the database in the physical layer of the Oracle Business Intelligence repository that contains the metadata to export. When the business model derives from multiple databases, then it eliminates metadata from all databases other than the one specified here. When the physical database is not found in the repository, an error message is displayed.

Table G–4 (Cont.) Cube Metadata Input File Parameters

Input File Name	Description
RUN_AS_USER	The user name of the database user whose visibility must be duplicated for the metadata export. This parameter cannot be empty. This user must exist as a user reference in the repository.
OUTPUT_FOLDER	The full path and file name of the folder to which the SQL file will be written. If the folder does not exist when you run the Oracle Database Metadata Generator, then it will be created. See "About the Output Files" for more information.

The following text shows a sample metadata input file:

```
BUSINESS_MODEL = "1 - Sample App"
PHYSICAL_DATABASE = "1 - Sample App Data"
RUN_AS_USER = "Administrator"
OUTPUT_FOLDER = "C:\OracleBI"
```

About the Output Files

Each Generator creates different types of output files, as described in the following list:

- **Oracle Database Metadata Generator:** Generates a SQL file that is encoded in UTF-8 and stored in the specified output folder. The file name is based on the name of the business model you specified in the input file, such as my_business_model.sql.
- **DB2 Cube Views Generator:** Generates the following files in the specified output folder:
  - XML file (encoded in UTF8). One XML file is created for the specified business model. It contains all objects that were converted to cubes. Additionally, objects in the repository will be mapped to similar objects in the IBM Cube Views metadata. See "Conversion Rules for IBM DB2 Databases" for a list of objects that will not be converted.

    The name of the XML file matches the business model name, without spaces, followed by the XML extension (for example, SalesResults.xml).
  - A SQL file that contains the alias generation DLL. A SQL file is created for the specified business model only if aliases exist in the physical layer databases that are referenced in the business model. The alias file contains SQL commands that will create the aliases in the DB2 database. The name of the SQL file matches the business model name, without spaces, followed by the SQL extension (for example, SalesResults-alias.sql).

Troubleshooting Errors from the Generator

Error messages indicate that the Generator was unable to complete some or all of its tasks. After starting the Generator, you might observe the following error messages:

- `Unable to write to Log file: log_file_name.`

  The log file specified in the input file or at the command line might contain the wrong path, the user might not have write permissions to that folder, or the disk could be out-of-space.

- `Run_as_user, user_name, is invalid.`

  The user name is incorrect.

- `Repository, repository_name.rpd, is invalid or corrupt.`

  The repository name might be incorrect, it might not exist in the given path, or the user might not have permission to read it.

- `Physical Database, database_name, is invalid.`

  The physical database name does not match a valid physical database object in the repository.

- `Business Model, model_name, is invalid.`

  The business model name does not match a valid business model object in the repository.

- `Authentication information provided is invalid.`

  The repository password provided at the command line is incorrect.

- `Path: "path_name" is invalid.`

  The path or file name is incorrect, or the current user does not have read access.

Metadata Conversion Rules and Error Messages

When the Generator creates the output files, it also maps the metadata objects in the Oracle Business Intelligence repository to similar objects in the metadata of the Oracle Database or the IBM DB2 database.

This section explains the rules used to identify Oracle Business Intelligence metadata that cannot be translated (converted) into either SQL or XML format. These rules are necessary because Oracle Database and IBM Cube Views do not support some of the metadata constructs that are allowed by Oracle Business Intelligence.

Dimensional metadata in the SQL or XML file will be generated at the logical fact table source level. If a logical fact table source has an invalid logical dimension table source, then the logical dimension table source will be invalidated. If the logical fact table source is invalid, then all the logical dimension table sources that are mapped to it will also be invalidated. Invalid Oracle Business Intelligence repository metadata elements will not be converted to cubes in the SQL or XML file.

When a rule is violated, the Generator writes the error messages and the metadata that violated the rule to the log file.

Conversion Rules for Oracle Databases

The following list provides the rules for converting Oracle Business Intelligence metadata into objects in the Oracle Database:

- Attributes that contain expressions in the logical table cannot be exported.
- Tables joined using complex joins are not considered.
- Tables that are opaque views are not considered.
- Columns used as part of a key in one level cannot be used as part of another level key.

  Oracle Database prohibits the use of columns as keys in multiple levels. This prohibition requires the Oracle Database Metadata Generator to eliminate one of the two joins, usually the join that is encountered first. Therefore, the other joins are lost, which prevents them from being exported.

Conversion Rules for IBM DB2 Databases

Table G–5 lists the rules used to validate Oracle Business Intelligence repository metadata elements, error messages that are written to the log file if the rule is violated, and an explanation of what caused the rule violation. The error messages help you determine why a particular Oracle Business Intelligence metadata object was not exported to the XML file.

Table G–5 Validation Rules for Metadata Elements

Rule	Message	Explanation
ComplexJoinFactsRule	[Fact Logical Table Source]Complex Physical Joins not supported %qn has a complex Join %qn between Physical Tables %qn and %qn	If the physical fact tables are connected through complex joins, then the join is not supported. A complex join is defined as any join between two tables that do not have a foreign key relationship.
ComplexJoinDimsRule	[Dimension Logical Table Source]Complex Physical Joins not supported %qn has a complex Join %qn between Physical Tables %qn and %qn	If the dimension physical tables are connected through a complex join, then that join is not supported.
ComplexJoinFactDimRule	[Fact Logical Table Source -> Dimension Logical Table Source] Complex Physical Joins not supported. %qn has a complex Join %qn between Physical Tables %qn and %qn.	If a dimension physical table and a fact physical table are connected through a complex join, then that join is not supported and the dimension table source is invalidated.
OpaqueViewFactRule	[Fact Logical table Source] Physical SQL Select Statements not supported. %qn uses the SQL Select Statement %qn.	When the physical fact table is generated by a SQL select statement, the logical fact table source that contains the table is invalidated. All logical dimension table sources connected to this logical fact table source are also invalidated. This construct allows subquery processing.
OpaqueViewDimRule	[Dimension Logical table Source] Physical SQL Select Statements not supported. %qn uses the SQL Select Statement %qn.	When a physical dimension table is generated by a SQL select statement, the logical dimension table source containing that table is invalidated.
OuterJoinFactRule	[Fact Logical Table Source] Physical Outer Joins not supported. %qn has an outer join %qn between physical tables %qn and %qn.	If the logical fact table source has an outer join mapping, then that logical fact table source is invalidated and all logical dimension table sources mapped to this source will also be invalidated.
OuterJoinDimRule	[Dimension Logical Table Source] Physical Outer Joins not supported. %qn has an outer join %qn between physical tables %qn and %qn.	If the logical dimension table source has an outer join mapping, then that logical dimension table source is invalidated.
WhereClauseFactRule	[Fact Logical Table Source] WHERE clauses are not supported. %qn has a where condition %s.	If the fact table source uses a `WHERE` clause to filter the data that is loaded, then this table source is invalidated.

Table G–5 (Cont.) Validation Rules for Metadata Elements

Rule	Message	Explanation
WhereClauseDimRule	[Dimension Logical Table Source] WHERE clauses are not supported. %qn has a where condition %s.	If the dimension table source uses a `WHERE` clause to filter the data that is loaded, then this table source is invalidated.
TwoJoinFactDimRule	[Fact Logical Table Source -> Dimension Logical Table Source] Multiple Joins between sources not supported. %qn and %qn have at least the following joins : %qn, %qn.	If a physical fact table is mapped to two dimension tables from the same dimension source (if the fact table is not exclusively mapped to the most detailed table in the table source), then the dimension table source is invalidated.
HiddenManyManyRule	[Fact Logical Table Source -> Dimension Logical Table Source] Join between (physical or logical?) fact and dimension is not on the most detailed table. %qn between %qn and %qn is not on the most detailed table %qn {Join name, facttable, dimtable).	This is related to the TwoJoinFactDimRule. If the fact table is joined to a dimension table that is not the most detailed table in the table source, then the dimension table source is invalidated.
ComplexMeasureRule	[Column] Complex Aggregation Rules not supported. %qn uses an aggregation rule of %s which is not supported.	The supported aggregations are typically `SUM`, `COUNT`, `AVG`, `MIN`, `MAX`, `STDDEV`, `COUNTDISTINCT`, and `COUNT`.
CountDistMeasureRule	[Column] COUNT-DISTINCT Aggregation Rule not supported. %qn uses an aggregation rule of %s which is not supported.	`COUNTDISTINCT` aggregation is not supported for this particular column.
InvalidColumnLevelRule	[Level] Some columns that are part of the Primary Level Key are invalid. %qn has %qn as part of its primary key, when %qn has already been marked invalid.	The level key for this level has one or more columns that are invalid.
VariableBasedColumnRule	[Logical Table Source -> Column] Column uses a Variable in the Expression Column %qn uses a variable in its mapping.	The logical column uses repository and session variables in the expression.
OneFactToManyDimRule	[Fact Logical Table Source -> Dimension Logical Table Source] There must be a unique join path between the most detailed tables in the (logical or physical?) fact and the dimension. No join paths found between %qn and %qn (both physical table names). Found at least the following join paths: (%qn->%qn....), (%qn->%qn....)	Same as in TwoJoinFactDimRule or HiddenManyManyRule.
ManyMDTinFactRule	[Fact Logical Table Source] Fact Logical Table Source must have a unique most detailed table. %qn has at least the following most detailed tables : %qn,%qn.	A fact that has more than one table that is the most detailed table.

Table G–5 (Cont.) Validation Rules for Metadata Elements

Rule	Message	Explanation
NoMeasureFactRule	[Fact Logical Table Source] Fact Logical Table Source does not have any Measures. %qn does not have any deployable measures.	A fact table does not have any measures because all the measures have been invalidated.
NoInActiveFactRule	[Fact Logical Table Source] Fact Logical Table Source is not marked Active.	A fact source is not active.
NoInActiveDimRule	[Dimension Logical Table Source] Dimension Logical Table Source is not marked Active.	A dimension source is not active.
NoAttributeInFactRule	[Fact Logical Table Source -> Column] Attribute found in Fact. %qn in a fact source %qn does not have an aggregation rule.	No attributes in the fact source.
NoMeasureInDimRule	[Dimension Logical Table Source -> Column] Measure found in Dimension. %qn in a dimension source %qn has an aggregation rule.	No measures in the dimension source.
VisibleColumnsAttrRule	[Column] -> The run_as_user does not have visibility to this Logical Column. %qn is not accessible to the run_as_user %qn due to visibility rules.	A column does not have visibility for this user.
VisibleColumnsMeasRule	[Column] -> The run_as_user does not have visibility to this Logical Column. %qn is not accessible to the run_as_user %qn due to visibility rules.	A column does not have visibility for this user.
MultiplePrimaryKeysDimRule	[Dimension Logical Table Source] A Join uses an alternate key in the Dimension Logical Table Source. %qn between %qn and %qn in %qn uses the alternate key %qn.	A dimension physical table can contain only one primary key. It is joined to another dimension physical table using a different unique key and that join is invalid. IBM Cube Views does not accept any unique keys to be used for foreign joins and always requires the primary key.
MultiplePrimaryKeysFactRule	[Dimension Logical Table Source] A Join uses an alternate key in the Dimension Logical Table Source. %qn between %qn and %qn in %qn uses the alternate key %qn.	A fact physical table can contain only one primary key. It is joined to another fact physical table using a different unique key and that join is invalid. IBM Cube Views does not accept any unique keys to be used for foreign joins and always requires the primary key.
MultiplePrimaryKeysFactDimRule	[Fact Logical Table Source -> Dim Logical Table Source] A Join uses an alternate key between the Logical Table sources. %qn between %qn and %qn for sources %qn and %qn uses the alternate key %qn.	A fact physical table can contain only one primary key. It is joined to a dimension physical table using a different unique key and is invalid. IBM Cube Views does not accept any unique keys to be used for foreign joins and always requires the primary key.

Table G–5 (Cont.) Validation Rules for Metadata Elements

Rule	Message	Explanation
NotDB2ExpressionAttrRule	[Dimension Logical Table Source -> Column] The Column contains an Expression not supported. %qn has expression %s which is not supported.	The attribute contains an expression not supported by IBM Cube Views. This includes metadata expressions that use DateTime functions (for example, `CURRENT_DATE`).
NotDB2ExpressionMeasRule	[Fact Logical Table Source -> Column] The Column contains an Expression not supported. %qn has expression %s which is not supported.	A measure contains an expression not supported by IBM Cube Views. This includes metadata expressions that use DateTime functions (for example,. `CURRENT_DATE`).
NoAttributeDimRule	[Dimension Logical Table Source] Dimension Logical Table Source does not have any attributes visible to the run_as_user. %qn can not be queried by user %qn since none of its attributes are visible.	A dimension does not have any attributes.

Using Materialized Views in the Oracle Database with Oracle Business Intelligence

This section explains how to export metadata from Oracle Business Intelligence into the SQL Access Advisor and create materialized views using the Oracle Database Metadata Generator.

This section contains the following topics:

- About Using the SQL Access Advisor with Materialized Views
- Deploying Metadata for Oracle Database

About Using the SQL Access Advisor with Materialized Views

This feature enhances the data warehouse performance and functionality of a database. It enables the SQL Access Advisor to store metadata about the logical relationships of the data that resides in the database. Additionally, it accelerates data warehouse queries by using more efficient Oracle materialized views. These materialized views preaggregate the relational data and improve query performance. Once the metadata is stored in the SQL Access Advisor, the database administrator can optimize the database objects and improve query performance.

When processing queries, Oracle Database routes queries to tables that hold materialized views when possible. Because these tables of materialized views are smaller than the underlying base tables and the data has been pre aggregated, the queries that are rerouted to them might run faster.

Oracle Database Metadata Generator works as a metadata bridge to convert the Oracle Business Intelligence proprietary metadata into a SQL file that contains PL/SQL commands to generate dimensions in the SQL Access Advisor. After converting metadata into a SQL file, you use a tool such as SQL*Plus to import the translated metadata into the SQL Access Advisor and store it in metadata catalog tables. After importing the metadata, you create materialized views, which are used by to optimize incoming application queries.

You can use this feature with Oracle Database 9*i* and higher. See "System Requirements and Certification" for information about platform compatibility.

Note that in database releases prior to 10*g*, the SQL Access Advisor was called the Oracle Database Summary Advisor and was documented in *Oracle9i Data Warehousing Guide*.

Deploying Metadata for Oracle Database

Become familiar with the Oracle Database and its tools before attempting to deploy metadata in the Oracle Database. For more information, see "SQL Access Advisor" in *Oracle Database Performance Tuning Guide*.

Ensure that you complete the steps in "Running the Generator" before deploying metadata. To deploy cube metadata, perform the tasks described in the following sections:

- Executing the SQL File for Oracle Database
- Defining Constraints for the Existence of Joins
- Creating the Query Workload
- Creating Materialized Views

Executing the SQL File for Oracle Database

Before executing the SQL file for importing into the SQL Access Advisor, ensure that you are familiar with Oracle Database import tools. See the Oracle Database documentation set for information.

Use a tool such as SQL*Plus to execute the SQL file that the Oracle Database Metadata Generator generated. You might see error messages if the dimensions already exist or if the database schema differs from that in the RPD file. When the script executes successfully, you can see the dimensions that were created by using the database web console or the Oracle Enterprise Manager Database Control. In the Oracle Enterprise Manager Database Control, expand the following nodes: Network, Databases, database-name, Warehouse, Summary Management, Dimensions, System.

After you execute the SQL file, be aware of the following:

- No incremental metadata changes are allowed. Schema changes require that you manually delete cube model metadata in the Oracle Database and convert the Oracle Business Intelligence metadata again. For example, if you must make a change to a dimension in a cube in the Oracle BI repository, you must delete the cube model in the Oracle Database, regenerate the SQL file from the Oracle BI repository, and import it into the SQL Access Advisor.
- You cannot delete metadata using the Oracle Database Metadata Generator. Instead, you must manually delete the cube model using the Oracle Enterprise Manager Database Control.

Defining Constraints for the Existence of Joins

For more information on this topic, see the Oracle Database documentation set.

You must ensure that Oracle Database knows about the joins between the dimension tables and the fact tables. To do so, you create constraints in SQL*Plus or the Oracle Enterprise Manager Database Control. In the Oracle Enterprise Manager Database Control, you select the table on which you must create a constraint, then select the Constraint tab.

You create a different type of constraint for each kind of table, as follows:

- For dimension tables, create a `UNIQUE` key constraint.

- For fact tables, create a FOREIGN key constraint and specify the referenced schema and referenced table. In the Constraint Definition area, include the foreign key columns in the fact table and the corresponding unique keys in the dimension table. An attempt to create a foreign key on a fact table can fail if the foreign key column data does not match the unique key column data on the dimension table.

Creating the Query Workload

See the Oracle Database documentation set for detailed information about creating the query workload.

A query workload is a sample set of physical queries to optimize. Before you create the workload, you generate a Trace file with information on the slowest-running queries.

To generate the Trace file:

You can generate the Trace file of the slowest-running queries using a tool that is appropriate to your database version, as described in the following list:

- **Usage Tracking:** Use this capability in Oracle Business Intelligence to log queries and how long they take to run. Long-running Oracle Business Intelligence queries can then be executed as a script and used with the Trace feature in the Oracle Database to capture the Oracle Database SQL code for these queries.
- **Oracle Database Trace:** Use this tool to identify the slowest physical query. You can enable the Trace feature either within Oracle Enterprise Manager Database Control or by entering SQL commands with the DBMS_MONITOR package. Once you enable the Trace feature, you use a script to create a Trace file to capture the SQL code for queries in a query workload table.
- **Oracle Enterprise Manager:** Use this tool to track slow-running queries.

> **Note:** The capabilities that are described in the following sections are available in Oracle Database, rather than as part of Oracle Business Intelligence.

To analyze the information in the Trace file:

1. Use the following guidelines when reviewing the Trace file:
   - When you have traced many statements at once, such as in batch processes, quickly discard any statements that have acceptable query execution times. Focus on those statements that take the longest times to execute.
   - Check the Query column for block visits for read consistency, including all query and subquery processing. Inefficient statements are often associated with a large number of block visits. The Current column indicates visits not related to read consistency, including segment headers and blocks that will be updated.
   - Check the Disk column for the number of blocks that were read from disk. Because disk reads are slower than memory reads, the value will likely be significantly lower than the sum of the Query and Current columns. If it is not, check for issues with the buffer cache.
   - Locking problems and inefficient PL/SQL loops can lead to high CPU time values even when the number of block visits is low.
   - Watch for multiple parse calls for a single statement, because this indicates a library cache issue.

2. After identifying the problem statements in the file, check the execution plan to learn why each problem statement occurred.

To load queries into the workload:

- After you use the Trace utility to learn the names of the slowest physical queries, insert them into the `USER_WORKLOAD` table.

  Table G–6 describes the columns of the `USER_WORKLOAD` table.

- Use `INSERT` statements to populate the `QUERY` column with the SQL statements for the slowest physical queries and the `OWNER` column with the appropriate owner names.

Table G–6 Columns in USER_WORKLOAD Table

Column	Data Type	Required	Description
QUERY	Any LONG or VARCHAR type (all character types)	YES	SQL statement for the query.
OWNER	VARCHAR2 (30)	YES	User who last executed the query.
APPLICATION	VARCHAR2 (30)	NO	Application name for the query.
FREQUENCY	NUMBER	NO	Number of times that the query was executed.
LASTUSE	DATE	NO	Last date on which the query was executed.
PRIORITY	NUMBER	NO	User-supplied ranking of the query.
RESPONSETIME	NUMBER	NO	Execution time of the query in seconds.
RESULTSIZE	NUMBER	NO	Total number of bytes that the query selected.
SQL_ADDR	NUMBER	NO	Cache address of the query.
SQL_HASH	NUMBER	NO	Cache hash value of the query.

Creating Materialized Views

After you populate the query workload table, use the appropriate tool for the Oracle Database version to create materialized views. In Oracle Database 10*g*, use the SQL Access Advisor in the Oracle Enterprise Manager Database Control and specify the query workload table that you created.

The SQL Access Advisor generates recommendations on improving the performance of the fact tables that you specify. The SQL Access Advisor displays the SQL code with which it will create the appropriate materialized views. Before indicating that the SQL Access Advisor should create the materialized views, review the following tips:

- The creation of a materialized view can fail if the SQL code includes a `CAST` statement.
- Ensure that the `CREATE MATERIALIZED VIEW` statement does not specify the same query that you provided as a workload table. If the statement does specify the same query, then the materialized views will likely not reflect the true performance gain. However, if the query is executed frequently, then creating a materialized view might still be worthwhile.

- Add a forward slash (/) to the end of the `CREATE MATERIALIZED VIEW` statement after the SQL statement. Otherwise, the SQL*Plus worksheet will not recognize it as a valid statement.

> **Note:** The SQL Access Advisor can also help determine appropriate indexing schemes.

Using IBM DB2 Cube Views with Oracle Business Intelligence

This section explains how to export metadata from Oracle Business Intelligence into IBM DB2 using the DB2 Cube Views Generator.

This section contains the following topics:

About Using IBM DB2 Cube Views with Oracle Business Intelligence

The term IBM DB2 Cube Views is a registered trademark of IBM. See "System Requirements and Certification" for information about platform compatibility.

This feature enhances the data warehouse performance and functionality of a database. It enables the DB2 database to store metadata about the logical relationships of the data residing in the database. Additionally, it accelerates data warehouse queries by using more efficient DB2 materialized query tables (MQTs). These MQTs preaggregate the relational data and improve query performance.

When processing queries, the DB2 Query Rewrite functionality routes queries to the MQTs when possible. Because these tables are smaller than the underlying base tables and the data has been pre aggregated, the queries that are rerouted to them might run faster.

DB2 Cube Views Generator works as a metadata bridge to convert the Oracle Business Intelligence proprietary metadata into an IBM Cube Views XML file. After converting metadata into an XML file, you use IBM Cube Views to import the translated metadata into the DB2 database and store it in IBM Cube Views metadata catalog tables. After importing the metadata, you use the IBM Optimization Advisor to generate scripts to create materialized query tables (MQT) and their indexes. The deployed MQTs are used by the DB2 Query Reroute Engine to optimize incoming application queries.

DB2 provides an API (implemented as a stored procedure) that passes XML documents as arguments to create, modify, delete, or read the metadata objects. For more information about IBM Cube Views, see the IBM DB2 documentation.

Deploying Cube Metadata

The alias-SQL file generated by the DB2 Cube Views Generator should be executed before importing the XML file. The XML file generated by the DB2 Cube Views Generator contains the cube metadata in XML format. After importing the XML file into your DB2 database, you must create materialized query tables.

> **Note:** It is strongly recommended that you become familiar with IBM Cube Views and its tools before attempting to import the XML file. For more information, see the IBM documentation.

Ensure that you complete the steps in "Running the Generator" before deploying metadata. To deploy cube metadata, perform the tasks described in the following sections:

- Executing the Alias-SQL File for IBM Cube Views
- Importing the XML File
- Guidelines for Creating Materialized Query Tables (MQTs)

Executing the Alias-SQL File for IBM Cube Views

You must execute the alias-SQL file before you import the XML file into your DB2 database. For more information, see the IBM documentation.

The alias-SQL file that is generated by the DB2 Cube Views Generator must be executed by a SQL client on the database where the data warehouse is located. When executed, it creates aliases (synonyms) for tables in the database.

Importing the XML File

After you execute the alias-SQL file, you can import the XML file into the database. For more information, see the IBM documentation.

Note: It is strongly recommended that you become familiar with IBM Cube Views and its tools before attempting to import the XML file. For more information, see the IBM documentation.

You can import this file using the following IBM tools:

- **IBM OLAP Center (recommended).** For more information, see "Guidelines for Importing the XML File Using the IBM OLAP Center" and the IBM documentation.
- **IBM command-line client utility (db2mdapiclient.exe).** IBM ships this utility with DB2. For more information about using the command-line client utility, see the IBM documentation.
- **IBM DB2 Stored Procedure.** IBM Cube Views provides a SQL-based and XML-based application programming interface (API) that you can use to run a single stored procedure to create, modify, and retrieve metadata objects. For more information, see the IBM documentation.

Guidelines for Importing the XML File Using the IBM OLAP Center Using the IBM OLAP Center, you can import cube metadata into DB2. The IBM OLAP Center provides wizards to help you import the file. For more information, see the IBM documentation.

To import the XML file, use the following guidelines:

- Using the IBM OLAP Center tool, connect to the DB2 database.
- In the Import Wizard, choose the XML file that you want to import.
- If metadata exists that refers to database constructs that are not in the database, then an error message is displayed.
- When the wizard asks for an import option, choose to replace existing objects.
- When you are returned to the IBM OLAP Center, a diagram of the cube model is shown.

Guidelines for Changing Cube Metadata After Importing the XML File After you import the XML file, you might need to perform the following actions:

- Because Oracle OLAP does not store foreign keys as metadata, they will not exist in the converted metadata in the DB2 database. You must use the IBM Referential Integrity Utility for IBM Cube Views to generate foreign key informational constraints. You can obtain this utility on the IBM Web site.
- You might encounter other issues such as foreign key join columns being nullable. You can use the following methods to solve this problem:
  - If data in these columns are not null, then you should convert these columns to not-null columns.
  - If data in these columns are null or you prefer not to convert the column data type even if the column data is not null, then you should modify the cube model using the following guidelines:
    * In a fact-to-dimension join, you must manually eliminate this dimension object from the converted cube model and create a degenerated dimension object consisting of the foreign key of this join.
    * In a dimension-to-dimension join, you must manually eliminate the dimension object that represents the primary-key side of the join from the converted cube model and create a degenerated dimension object consisting of the foreign key of this join.
    * In a fact-to-fact join, you must manually eliminate the fact object that represents the primary-key side of the join from the converted cube model and create a degenerated dimension object consisting of the foreign key of this join.
- No incremental metadata changes will be allowed by the Cube Generator. Schema changes require that you manually delete cube model metadata in the DB2 database and convert the Oracle Business Intelligence metadata again. For example, if you must make a change to a dimension in a cube in the Oracle Business Intelligence metadata repository, then you must delete the cube model in the DB2 database, regenerate the XML file from the Oracle Business Intelligence repository, and import it into the DB2 database.
- You cannot delete metadata using the DB2 Cube Views Generator. Instead, you must manually delete the cube model using the IBM OLAP Center.
- The IBM Statistics tool and IBM Optimization Advisor must be run periodically.

For more information, see the IBM documentation.

Guidelines for Creating Materialized Query Tables (MQTs)

For more information, see the IBM documentation.

After you import the cube metadata into the database, you must run the IBM Optimization Advisor to generate SQL scripts and then execute those scripts to create the MQTs. You must provide certain parameters to the IBM Optimization Advisor to get optimal results from the implementation. The IBM Optimization Advisor wizard analyzes your metadata and recommends how to build summary tables that store and index aggregated data for SQL queries. Running the IBM Optimization Advisor can help you keep the MQTs current. Additionally, you must refresh your database after each ETL.

To create MQTs, use the following guidelines:

- In the IBM OLAP Center, choose the cube model that you want to optimize and open the IBM Optimization Advisor wizard.
- Follow the instructions in the wizard, using the following table as a guide.

When asked for:	Choose:
Summary Tables	Choose Deferred (or Immediate) and provide a tablespace for the tables
Limitations	Choose an appropriate value for the optimization parameters. You should turn on the Data-sampling option.
SQL Scripts	Creation of the scripts needed to run to create the Summary tables. Choose the filename and locations

- When the IBM Optimization Advisor closes, you must execute the SQL scripts to create the MQTs.

H

XML Schema Files for ADF Mapping Customizations

This appendix describes the app_segment_rule.xsd and mapping_rules.xsd XML schema files. XML files based on these XML validation schemas define mapping rules that control the physical to logical mapping behavior for ADF data source objects.

This appendix contains the following sections:

app_segment_rule.xsd XML Schema File

The XML schema file app_segment_rule.xsd contains mapping rules that define the physical table to logical table mapping. This section provides an annotated version of the app_segment_rule.xsd XML schema file.

Example H–1 app_segment_rule.xsd

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="alias_t">
  <xs:attribute name="aliasTableName" type="xs:string" use="required" />
 </xs:complexType>
 <xs:complexType name="aliasTableList_t">
  <xs:sequence>
   <xs:element name="AliasTable" type="alias_t" minOccurs="1" maxOccurs=
   "unbounded" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="subjectArea_t">
  <xs:attribute name="subjectAreaName" type="xs:string" use="required" />
  <xs:attribute name="table" type="xs:string" use="optional" />
 </xs:complexType>
 <xs:complexType name="subjectAreaList_t">
  <xs:sequence>
   <xs:element name="OTBISubjectArea" type="subjectArea_t" minOccurs="1"
   maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="columnmapping_t">
```

```
<!-- voColumnName specifies the ADF view object column name -->

  <xs:attribute name="voColumnName" type="xs:string" use="required" />

<!-- logicalDimColumnName specifies the logical column name -->

  <xs:attribute name="logicalDimColumnName" type="xs:string" use="required" />
 </xs:complexType>
 <xs:complexType name="columnMappingsList_t">
  <xs:sequence>

<!-- ColumnMapping specifies the set of column mappings -->

   <xs:element name="ColumnMapping" type="columnmapping_t" minOccurs="1"
   maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="tableName_t">
  <xs:attribute name="name" type="xs:string" />
 </xs:complexType>
 <xs:complexType name="relatedLogicalTablesList_t">
  <xs:sequence>
   <xs:element name="LogicalTable" type="tableName_t" minOccurs="1"
   maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>

<!-- Base VO to which DFF alias needs to be joined. -->

 <xs:complexType name="joinWith_t">
  <xs:attribute name="voName" type="xs:string" use="required"/>
 </xs:complexType>

<!-- Logical Dimension to which DFF alias needs to be mapped. -->

 <xs:complexType name="mapTo_t">
  <xs:attribute name="logTabName" type="xs:string" use="required"/>
 </xs:complexType>

<!-- Specifies DFF alias name. -->

 <xs:complexType name="createAlias_t">
  <xs:sequence>
   <xs:element name="JoinWith" type="joinWith_t" minOccurs="1"
   maxOccurs="unbounded"/>
   <xs:element name="MapTo" type="mapTo_t" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>

<!-- Specifies DFF alias name. -->

  <xs:attribute name="name" type="xs:string" use="required"/>
 </xs:complexType>

<!-- Specifies set of LTSes. -->

 <xs:complexType name="ltsObject_t">
  <xs:attribute name="ltsName" type="xs:string"/>
 </xs:complexType>
  <xs:complexType name="ltsObjects_t">
```

```
  <xs:sequence>
   <xs:element name="Lts" type="ltsObject_t" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="labelToDimMap_t">
  <xs:sequence>

<!-- For mapping the _ and _c suffixed value and value set columns in the code,
combine flattened fact view objectS corresponding to the segment mapped to the
logical table -->

   <xs:element name="FlattenedFlexVOColumnMappings" type="columnMappingsList_t"
   minOccurs="0" maxOccurs="1" />

<!-- TableSpecificColumnMappingOverrides is for future use -->

   <xs:element name="TableSpecificColumnMappingOverrides"
   type="columnMappingsList_t" minOccurs="0" maxOccurs="1" />

<!-- TableSpecificOTBISubjectAreas specifies the Oracle Transactional Business
Intelligence subjects areas. The unqualified segment Logical Tables Dim - GL
Segment 1 - 10 are dragged to these subject areas. For other use cases, specify
your own repository subject areas. -->

   <xs:element name="TableSpecificOTBISubjectAreas" type="subjectAreaList_t"
   minOccurs="0" maxOccurs="1" />

<!-- DFFBaseVOAliases specifies the alias view objects of the base view object
mapped to the same logical table. For each of these view objects, the
Administration Tool creates an alias of the DFF view object and joins it to the
base view object alias. -->

   <xs:element name="DFFBaseVOAliases" type="aliasTableList_t" minOccurs="0"
   maxOccurs="1" />
  </xs:sequence>

<!-- RelatedLogicalTables specifies the list of logical tables to which the DFF
view object must be mapped in addition to the logical table mentioned in the
logicalDimTableName attribute. -->

   <xs:element name="RelatedLogicalTables" type="relatedLogicalTablesList_t"
   minOccurs="0" maxOccurs="1" />

<!-- Use this tag to create an alias of the VO, join it with several other
VOs, and map to a specified set of logical tables-->
   <xs:element name="CreateAlias" type="createAlias_t"
   minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>

<!-- segmentLabelName is for future use -->

  <xs:attribute name="segmentLabelName" type="xs:string" use="optional" />

<!-- logicalDimTableName specifies the logical table to which the ADF view object
is mapped. ADF database properties contain the mapping between the ADF view object
("BIObject_'ADF VO Name'") and logical table mentioned here. -->

  <xs:attribute name="logicalDimTableName" type="xs:string" use="required" />

<!-- In role playing dimensions, the logical table specified in the
```

```
roleMasterLogicalTableName attribute is used as the master logical table. Whenever
the master logical table is mapped to a view object, the alias of this view object
is created and joined to the table referenced in the flattenedFlexVORoleAlias
attribute. The alias table name is "{ADF_VO_Name}_{Role_Playing_Logical_Table_
Name}". Logical columns are mapped to the columns of this alias table. -->

  <xs:attribute name="roleMasterLogicalTableName" type="xs:string"
  use="optional" />

<!-- flattenedFlexVORoleAlias specifies the flattened view object alias that needs
to be joined with the role playing view object alias table. -->

  <xs:attribute name="flattenedFlexVORoleAlias" type="xs:string" use="optional" />
 </xs:complexType>

<!-- The root element for the mapping rule document -->

 <xs:complexType name="mappingRules_t">
  <xs:sequence>

<!-- List of LTSes to be Disabled -->
   <xs:element name="LTSToBeDisabled" type="ltsObjects_t" minOccurs="0"
maxOccurs="1"/>

<!-- GlobalColumnMappings is for future use -->

   <xs:element name="GlobalColumnMappings" type="columnMappingsList_t"
   minOccurs="0" maxOccurs="1" />

<!-- GlobalOTBISubjectAreas specifies the Oracle Transactional Business
Intelligence subject areas. The unqualified segment Logical Tables Dim - GL
Segment 1 - 10 are dragged to these subject areas. The rules specified here apply
to all logical tables. For other use cases, specify your own subject areas. -->

   <xs:element name="GlobalOTBISubjectAreas" type="subjectAreaList_t"
   minOccurs="0" maxOccurs="1" />

<!-- LabelToDimensionMappings specifies the rules for a logical table -->

   <xs:element name="LabelToDimensionMappings" type="labelToDimMap_t"
   minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>

<!-- appName is the ADF Module Name from where the ADF view objects are imported.
It is also used as a key for reading the properties from the ADF database. It is
sometimes known as the Category or Module Name. -->

  <xs:attribute name="appName" type="xs:string" use="required" />

<!-- businessModelName specifies the business model in the Business Model and
Mapping layer. The ADF view objects are mapped to the logical tables in this
business model. The default business model is "Core." -->

  <xs:attribute name="businessModelName" type="xs:string" default="Core"
   use="optional" />

<!-- flattenedFlexLogicalTable specifies the logical table to which the flattened
fact view object for the KFF needs to map -->

  <xs:attribute name="flattenedFlexLogicalTable" type="xs:string"
```

```
  use="optional" />

<!-- Determines whether VOs under this AM needs to be submitted for
ETL extension -->

  <xs:attribute name="isETLSupported" type="xs:boolean" use="optional"/>

<!-- Specifies the unqualified Dimension Prefix for the AM -->

  <xs:attribute name="unqualifiedDimensionPrefix" type="xs:string"
  use="optional"/>

 </xs:complexType>

<!-- This is the actual element declaration for the entire Document. -->

 <xs:element name="document">
  <xs:complexType>
   <xs:sequence>

<!--  MappingRules specifies the set of rules. -->

    <xs:element name="MappingRules" type="mappingRules_t" minOccurs="0"
    maxOccurs="unbounded" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

app_segment_rules_*.xml Example

This section provides an example of an app_segment_rules_*.xml file, based on the validation rules contained in app_segment_rules.xsd. Note that the actual file has been shortened.

Example H–2 app_segment_rules_*.xml example

```
<?xml version="1.0" encoding="UTF-8" ?>
<document>
<!--  Begin Finance -->
<MappingRules appName="FscmTopModelAM.AccountBIAM" businessModelName="Core"
flattenedFlexLogicalTable="Dim - GL Account">
 <GlobalOTBISubjectAreas>
  <OTBISubjectArea subjectAreaName="General Ledger - Journals Real Time" />
  <OTBISubjectArea subjectAreaName="General Ledger - Transactional Balances Real Time" />
  <OTBISubjectArea subjectAreaName="Payables Invoices - Prepayment Invoice Distributions Real
  Time" />
  <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" />
  <OTBISubjectArea subjectAreaName="Payables Invoices - Trial Balance Real Time" />
...
 </GlobalOTBISubjectAreas>
 <LabelToDimensionMappings logicalDimTableName="Dim - Balancing Segment">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Balancing Segment Code" />
   <ColumnMapping voColumnName="_c" logicalDimColumnName="Balancing Segment Value Set
   Code" />
  </FlattenedFlexVOColumnMappings>
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="General Ledger - Journals Real Time" table="- Balancing
   Segment" />
   <OTBISubjectArea subjectAreaName="General Ledger - Transactional Balances Real Time"
   table="Balancing Segment" />
```

```
    <OTBISubjectArea subjectAreaName="Payables Invoices - Prepayment Invoice Distributions
    Real Time" table="- Balancing Segment" />
    <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" table=
    "- Balancing Segment" />
    <OTBISubjectArea subjectAreaName="Payables Invoices - Trial Balance Real Time"
    table="Balancing Segment" />
...
   </TableSpecificOTBISubjectAreas>
  </LabelToDimensionMappings>
  <LabelToDimensionMappings logicalDimTableName="Dim - Cost Center">
   <FlattenedFlexVOColumnMappings>
    <ColumnMapping voColumnName="_" logicalDimColumnName="Cost Center Segment Code" />
    <ColumnMapping voColumnName="_c" logicalDimColumnName="Cost Center Segment Value Set Code"
    />
   </FlattenedFlexVOColumnMappings>
   <TableSpecificOTBISubjectAreas>
    <OTBISubjectArea subjectAreaName="General Ledger - Journals Real Time" table="- Cost
    Center Segment" />
    <OTBISubjectArea subjectAreaName="General Ledger - Transactional Balances Real Time"
    table="Cost Center Segment" />
    <OTBISubjectArea subjectAreaName="Payables Invoices - Prepayment Invoice Distributions
    Real Time" table="- Cost Center" />
    <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" table="-
    Cost Center" />
    <OTBISubjectArea subjectAreaName="Payables Invoices - Trial Balance Real Time" table="Cost
  Center" />
...
   </TableSpecificOTBISubjectAreas>
  </LabelToDimensionMappings>
  <LabelToDimensionMappings logicalDimTableName="Dim - Natural Account Segment">
   <FlattenedFlexVOColumnMappings>
    <ColumnMapping voColumnName="_" logicalDimColumnName="Natural Account Segment Code" />
    <ColumnMapping voColumnName="_c" logicalDimColumnName="Natural Account Segment Value Set
    Code" />
   </FlattenedFlexVOColumnMappings>
   <TableSpecificOTBISubjectAreas>
    <OTBISubjectArea subjectAreaName="General Ledger - Journals Real Time" table="- Natural
    Account Segment" />
    <OTBISubjectArea subjectAreaName="General Ledger - Transactional Balances Real Time"
    table="Natural Account Segment" />
    <OTBISubjectArea subjectAreaName="Payables Invoices - Prepayment Invoice Distributions
    Real Time" table="- Natural Account" />
    <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" table=
    "- Natural Account" />
    <OTBISubjectArea subjectAreaName="Payables Invoices - Trial Balance Real Time"
    table="Natural Account" />
...
   </TableSpecificOTBISubjectAreas>
  </LabelToDimensionMappings>
  <LabelToDimensionMappings logicalDimTableName="Dim - GL Segment1">
   <FlattenedFlexVOColumnMappings>
    <ColumnMapping voColumnName="_" logicalDimColumnName="Account Segment1 Code" />
    <ColumnMapping voColumnName="_c" logicalDimColumnName="Account Segment1 Value Set Code" />
   </FlattenedFlexVOColumnMappings>
  </LabelToDimensionMappings>
  <LabelToDimensionMappings logicalDimTableName="Dim - GL Segment2">
   <FlattenedFlexVOColumnMappings>
    <ColumnMapping voColumnName="_" logicalDimColumnName="Account Segment2 Code" />
    <ColumnMapping voColumnName="_c" logicalDimColumnName="Account Segment2 Value Set Code" />
   </FlattenedFlexVOColumnMappings>
  </LabelToDimensionMappings>
...
  <LabelToDimensionMappings logicalDimTableName="Dim - AP Account Balancing Segment"
  roleMasterLogicalTableName="Dim - Balancing Segment" flattenedFlexVORoleAlias="Dim_FLEX_BI_
  Account_VI_APAccount">
   <TableSpecificOTBISubjectAreas>
```

```
   <OTBISubjectArea subjectAreaName="Payables Invoices - Prepayment Invoice Distributions
   Real Time" table="- Distribution Balancing Segment Value" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" table="-
   Distribution Balancing Segment Value" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Dim - AP Account Cost Center"
 roleMasterLogicalTableName="Dim - Cost Center" flattenedFlexVORoleAlias="Dim_FLEX_BI_
 Account_VI_APAccount">
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Payables Invoices - Prepayment Invoice Distributions
   Real Time" table="- Distribution Cost Center Segment Value" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" table="-
   Distribution Cost Center Segment Value" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
...
</MappingRules>
<MappingRules appName="FscmTopModelAM.LocationBIAM" businessModelName="Core"
flattenedFlexLogicalTable="Dim - Asset Location">
 <LabelToDimensionMappings logicalDimTableName="Segment1">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Segment1" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Segment2">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Segment2" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
...
</MappingRules>
<MappingRules appName="FscmTopModelAM.CategoryBIAM" businessModelName="Core"
flattenedFlexLogicalTable="Dim - Asset Category">
 <LabelToDimensionMappings logicalDimTableName="Segment1">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Segment1" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Segment2">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Segment2" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
...
 <LabelToDimensionMappings logicalDimTableName="Major Category">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Major Category" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Minor Category">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Minor Category" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
</MappingRules>
...
<MappingRules appName="FscmTopModelAM.ExternalTransactionBIAM" businessModelName="Core"
flattenedFlexLogicalTable="">
<!-- Use the below tag if you want any one of the following use cases for the "base logical
table" your DFF is mapped to in ATG -->
<!-- (a) need to expose the base logical table DFF attributes to specific presentation
subject area and table -->
<!-- (b) you have additional VO aliases mapped to the base logical table in addition to the
base VO itself -->
<!-- (c) you have additional "related logical tables" that you want automapped when the base
```

```
logical table is mapped in ATG -->
 <LabelToDimensionMappings logicalDimTableName="Dim - CE External Cash Transaction Details">
<!-- Use the below tag to indicate the presentation subject area and table where the DFF
attributes need to be exposed -->
<!-- Repeat the OTBISubjectArea tag to expose them in multiple subject areas or tables -->
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Cash Management - External Cash Transactions Real Time"
   table="External Cash Transaction Detail" />
  </TableSpecificOTBISubjectAreas>
<!-- Use the below tag to give the VO alias names, if any, that are mapped to the base
logical table in addition to the base VO -->
<!-- Repeat the AliasTable tag if you have multiple VO aliases -->
  <DFFBaseVOAliases>
   <AliasTable aliasTableName="Fact_ExternalTransactionsPVO_JournalEntryLine" />
  </DFFBaseVOAliases>
 </LabelToDimensionMappings>
</MappingRules>
...
<MappingRules appName="FscmTopModelAM.CalendarTypeDFFBIAM" businessModelName="Core"
flattenedFlexLogicalTable="">
...
 <LabelToDimensionMappings logicalDimTableName="Dim - Date Fixed Assets Calendar">
...
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Transactions Real Time"
   table="Transaction Effective Date" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
<!-- Use the below tag if you have any "role-playing logical tables" for the base logical
table. -->
 <LabelToDimensionMappings logicalDimTableName="Dim - Date Placed in Service Fixed Asset
 Calendar" roleMasterLogicalTableName="Dim - Date Fixed Assets Calendar"
 flattenedFlexVORoleAlias="Dim_CalendarDayPVO_DatePlacedInService">
<!-- Use the below tag to indicate the presentation subject area and table where the DFF
attributes from the role-playig logical table need to be exposed -->
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Financial Information Real Time"
   table="Date Placed in Service" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Dim - Transaction Date Fixed Asset Calendar"
 roleMasterLogicalTableName="Dim - Date Fixed Assets Calendar" flattenedFlexVORoleAlias="Dim_
 CalendarDayPVO_TransactionDate">
<!-- Use the below tag to indicate the presentation subject area and table where the DFF
attributes from the role-playig logical table need to be exposed -->
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Assignments Real Time"
   table="Transaction Date" />
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Financial Information Real Time"
   table="Transaction Date" />
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Retirements and Reinstatements Real
   Time" table="Transaction Date" />
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Transactions Real Time"
   table="Transaction Date" />
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Transfer Real Time"
   table="Transaction Date" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
...
</MappingRules>
...
<!-- Rule file for AP_TERMS_B -->
<MappingRules appName="FscmTopModelAM.PaymentTermHeaderDffBIAM" businessModelName="Core"
flattenedFlexLogicalTable="">
 <LabelToDimensionMappings logicalDimTableName="Dim - AP Terms">
  <TableSpecificOTBISubjectAreas>
```

```
   <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" table=
   "- Header Information" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
</MappingRules>
<!-- Rule file for AP_PAYMENT_SCHEDULES -->
<MappingRules appName="FscmTopModelAM.InstallmentsDffBIAM" businessModelName="Core"
flattenedFlexLogicalTable="">
 <LabelToDimensionMappings logicalDimTableName="Dim - AP Payment Schedule Details">
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Payables Invoices - Installments Real Time"
   table="Invoices Installment Details" />
  </TableSpecificOTBISubjectAreas>
  <DFFBaseVOAliases>
   <AliasTable aliasTableName="Fact_InvoicePaymentSchedulePVO_Disbursement" />
  </DFFBaseVOAliases>
 </LabelToDimensionMappings>
</MappingRules>
...
<!-- Rule file for DFF AP_HOLDS -->
<MappingRules appName="FscmTopModelAM.HoldsDffBIAM" businessModelName="Core"
flattenedFlexLogicalTable="">
 <LabelToDimensionMappings logicalDimTableName="Dim - AP Hold Details">
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Payables Invoices - Holds Real Time" table="Invoices
   Hold Details" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
</MappingRules>
<!-- End Finance -->

<!-- Rule file for JobAM EFF -->
</MappingRules>
<MappingRules appName="HcmTopModelAnalyticsGlobalAM.JobAM.PER_JOBS_LEG_EFF"
businessModelName="Core" flattenedFlexLogicalTable="">
 <LabelToDimensionMappings logicalDimTableName="Dim - Job">
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Workforce Management - Worker Assignment Real Time"
table="Job" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
</MappingRules>
...
</document>
```

mapping_rules.xsd XML Schema File

The XML schema file mapping_rules.xsd contains mapping rules that define the physical column to logical column mapping. This section provides an annotated version of the mapping_rules.xsd XML schema file.

Example H–3 mapping_rules.xsd

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- This XML validation schema defines data extensibility rules that determine
how objects imported at the Physical layer are mapped to the Business
Model and Mapping layer. -->

 <xs:simpleType name="yes_no_t">
  <xs:restriction base="xs:string">
   <xs:enumeration value="yes" />
   <xs:enumeration value="no" />
```

```
  </xs:restriction>
 </xs:simpleType>

<!-- Description of one mapping -->

 <xs:complexType name="mapping_t">

<!-- The source attribute specifies the physical ADF view object column -->

  <xs:attribute name="source" type="xs:string" use="required" />

<!-- The target attribute specifies the target logical column suffix or complete
name -->

  <xs:attribute name="target" type="xs:string" use="required" />

<!-- The addprefix attribute specifies whether or not the prefix specified in the
"rule" must be prefixed with the target to get the logical table column name
before performing the mapping -->

  <xs:attribute name="addprefix" type="yes_no_t" default="no" use="optional" />
 </xs:complexType>
 <xs:complexType name="rule_t">
  <xs:sequence>

<!-- List of mappings that define the rule -->

   <xs:element name="mapping" type="mapping_t" minOccurs="1" maxOccurs="unbounded"
   />
  </xs:sequence>

<!-- The name attribute specifies the rule name. This name is used by the instance
to specify which rule to apply. -->

  <xs:attribute name="name" type="xs:string" use="required" />
 </xs:complexType>
 <xs:complexType name="rules_t">
  <xs:sequence>

<!-- Each rule defines the source (physical) to target (logical) mapping. It
is used by the Import Metadata Wizard during the Physical to Logical Mapping
phase. Each mapping in a rule can specify whether the default prefix for an
instance should be added. -->

   <xs:element name="rule" type="rule_t" minOccurs="1" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="apply_t">

<!-- The rule attribute specifies which rule to apply for the logical table
mentioned in the instance. -->

  <xs:attribute name="rule" type="xs:string" use="required" />

<!-- The prefix attribute specifies the string that might or might not be
prepended when performing the mapping between the ADF view object column and the
logical table column. -->

  <xs:attribute name="prefix" type="xs:string" default="" use="optional" />
 </xs:complexType>
```

```
 <xs:complexType name="instance_t">
  <xs:sequence>

<!-- apply specifies what rule to apply to the logical table -->

   <xs:element name="apply" type="apply_t" minOccurs="1" maxOccurs="unbounded" />
  </xs:sequence>

<!-- The logicaltable attribute specifies the logical table to which the rule is
applied -->

  <xs:attribute name="logicaltable" type="xs:string" use="required" />
 </xs:complexType>
 <xs:complexType name="instances_t">
  <xs:sequence>

<!-- Each instance is associated with a single logical table. It applies any
number of rules, with an optional prefix. The rules are applied in order. Note
that the prefix includes a space at the end. It merely concatenates with the
target logical column to determine the right logical column to map to. -->

   <xs:element name="instance" type="instance_t" minOccurs="1"
maxOccurs="unbounded" />
  </xs:sequence>

<!-- appName is the ADF Module Name from which ADF view objects are imported -->

   <xs:attribute name="appName" type="xs:string" />
 </xs:complexType>

<!-- The document element is the element declaration for the entire document -->

 <xs:element name="document">
  <xs:complexType>
   <xs:sequence>

<!-- The rules element specifies the set of rules -->

    <xs:element name="rules" type="rules_t" minOccurs="1" maxOccurs="1" />

<!-- The instances element specifies the set of instances related to a single ADF
module. -->

    <xs:element name="instances" type="instances_t" minOccurs="1" maxOccurs="1" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

mapping_rules_*.xml Example

This section provides an example of a mapping_rules_*.xml file, based on the validation rules contained in mapping_rules.xsd. Note that the actual file has been shortened.

Example H–4 mapping_rules_*.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<document>
<rules>
```

```
<!-- Each Rule defines the source (physical) to target (logical) mapping. It is
used by the Import Metadata Wizard during the Physical to Logical Mapping phase.
Each mapping in a rule can specify whether it should add the default prefix for an
instance. -->

 <rule name="GL Segment Tree Rule">
  <mapping source="TreeCode" target="Tree Code" addprefix="no" />
  <mapping source="TreeVersionId" target="Tree Version ID" addprefix="no" />
  <mapping source="VersionName" target="Tree Version Name" addprefix="no" />
  <mapping source="Dep0Pk2Value" target="Segment Value Set Code" addprefix="yes" />
  <mapping source="Dep0Value" target="Segment Code" addprefix="yes" />
  <mapping source="Dep0Description" target="Segment Description" addprefix="yes" />
  <mapping source="Dep1Value" target="Level 1 Code" addprefix="yes" />
  <mapping source="Dep1Description" target="Level 1 Description" addprefix="yes" />
  <mapping source="Dep2Value" target="Level 2 Code" addprefix="yes" />
  <mapping source="Dep2Description" target="Level 2 Description" addprefix="yes" />
  <mapping source="Dep3Value" target="Level 3 Code" addprefix="yes" />
  <mapping source="Dep3Description" target="Level 3 Description" addprefix="yes" />
...
  <mapping source="Distance" target="Fixed Hierarchy Level" addprefix="no" />
  <mapping source="VersionEffectiveStartDate" target="Effective From" addprefix="no" />
  <mapping source="VersionEffectiveEndDate" target="Effective To" addprefix="no" />
 </rule>
 <rule name="GL Segment Non-Tree Rule">
  <mapping source="Value" target="Segment Code" addprefix="yes" />
  <mapping source="ValueSetCode" target="Segment Value Set Code" addprefix="yes" />
  <mapping source="Description" target="Segment Description" addprefix="yes" />
 </rule>
 <rule name="Qualified Segment Tree Rule">
  <mapping source="TreeCode" target="Tree Code" addprefix="no" />
  <mapping source="TreeVersionId" target="Tree Version ID" addprefix="no" />
  <mapping source="VersionName" target="Tree Version Name" addprefix="no" />
  <mapping source="Dep0Pk2Value" target="Value Set Code" addprefix="yes" />
  <mapping source="Dep0Value" target="Code" addprefix="yes" />
  <mapping source="Dep0Description" target="Description" addprefix="yes" />
  <mapping source="Dep1Value" target="Level 1 Code" addprefix="yes" />
  <mapping source="Dep1Description" target="Level 1 Description" addprefix="yes" />
  <mapping source="Dep2Value" target="Level 2 Code" addprefix="yes" />
  <mapping source="Dep2Description" target="Level 2 Description" addprefix="yes" />
  <mapping source="Dep3Value" target="Level 3 Code" addprefix="yes" />
  <mapping source="Dep3Description" target="Level 3 Description" addprefix="yes" />
...
  <mapping source="Distance" target="Fixed Hierarchy Level" addprefix="no" />
  <mapping source="VersionEffectiveStartDate" target="Effective From" addprefix="no" />
  <mapping source="VersionEffectiveEndDate" target="Effective To" addprefix="no" />
  <mapping source="Dep0FlexValueAttribute6" target="Group Account Number" addprefix="no" />
 </rule>
 <rule name="Qualified Segment Non-Tree Rule">
  <mapping source="Value" target="Code" addprefix="yes" />
  <mapping source="ValueSetCode" target="Value Set Code" addprefix="yes" />
  <mapping source="Description" target="Description" addprefix="yes" />
  <mapping source="FlexValueAttribute6" target="Group Account Number" addprefix="no" />
 </rule>
 <rule name="Role Playing KFF Tree Rule 1">
  <mapping source="TreeCode" target="Tree Code" addprefix="no" />
  <mapping source="TreeVersionId" target="Tree Version ID" addprefix="no" />
  <mapping source="Dep0Description" target="Description" addprefix="yes" />
  <mapping source="Dep0Pk2Value" target="Value Set Code" addprefix="yes" />
  <mapping source="Dep0Value" target="Code" addprefix="yes" />
  <mapping source="Value" target="Code" addprefix="yes" />
  <mapping source="ValueSetCode" target="Value Set Code" addprefix="yes" />
  <mapping source="Description" target="Description" addprefix="yes" />
 </rule>
 <rule name="Role Playing KFF Tree Rule 2">
  <mapping source="TreeCode" target="Tree Code" addprefix="no" />
```

```
  <mapping source="TreeVersionId" target="Tree Version ID" addprefix="no" />
  <mapping source="Dep0Description" target="Desscription" addprefix="yes" />
  <mapping source="Dep0Pk2Value" target="Value Set Code" addprefix="yes" />
  <mapping source="Dep0Value" target="Code" addprefix="yes" />
  <mapping source="Value" target="Code" addprefix="yes" />
  <mapping source="ValueSetCode" target="Value Set Code" addprefix="yes" />
  <mapping source="Description" target="Desscription" addprefix="yes" />
 </rule>
...
</rules>
<instances appName="FscmTopModelAM.AccountBIAM">

<!-- Each instance is associated with the a single logical table. It applies any number of
rules, with an optional prefix. The rules are applied in order. Note that the prefix includes
a space at the end. It merely concatenates with the target logical column to determine the
right logical column to map to. -->

 <instance logicaltable="Dim - Balancing Segment">
  <apply rule="Qualified Segment Tree Rule" prefix="Balancing Segment" />
  <apply rule="Qualified Segment Non-Tree Rule" prefix="Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - Cost Center">
  <apply rule="Qualified Segment Tree Rule" prefix="Cost Center" />
  <apply rule="Qualified Segment Non-Tree Rule" prefix="Cost Center" />
 </instance>
 <instance logicaltable="Dim - Natural Account Segment">
  <apply rule="Qualified Segment Tree Rule" prefix="Account" />
  <apply rule="Qualified Segment Non-Tree Rule" prefix="Account" />
 </instance>
 <instance logicaltable="Dim - GL Segment1">
  <apply rule="GL Segment Tree Rule" />
  <apply rule="GL Segment Non-Tree Rule" />
 </instance>
 <instance logicaltable="Dim - GL Segment2">
  <apply rule="GL Segment Tree Rule" />
  <apply rule="GL Segment Non-Tree Rule" />
 </instance>
 <instance logicaltable="Dim - GL Segment3">
  <apply rule="GL Segment Tree Rule" />
  <apply rule="GL Segment Non-Tree Rule" />
 </instance>
...
 <instance logicaltable="Dim - AP Account Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 1" prefix="Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - AP Asset Account Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 1" prefix="Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - AP Pay Account Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 1" prefix="Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - Asset From Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 3" prefix="Asset From Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - Asset To Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 3" prefix="Asset To Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - Asset Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 3" prefix="Asset Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - Distribution Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 3" prefix="Distribution Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - SLA Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 3" prefix="Balancing Segment" />
 </instance>
```

```
 <instance logicaltable="Dim - CE Asset Balancing Segment Value">
  <apply rule="Role Playing KFF Tree Rule 1" prefix="Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - CE Offset Balancing Segment Value">
  <apply rule="Role Playing KFF Tree Rule 1" prefix="Balancing Segment" />
 </instance>
...
</instances>
</document>
```

Administration Tool Keyboard Shortcuts

This appendix provides keyboard shortcut information for the Oracle BI Administration Tool, including menu items and their corresponding keyboard shortcuts, keyboard shortcuts for navigating dialogs, and Physical Diagram and Business Model Diagram keyboard shortcuts.

This appendix contains the following topics:

Menu Keyboard Shortcuts

The following sections describe keyboard shortcuts for Administration Tool menu options.

File Menu Shortcuts

- The shortcut for New is Ctrl + N.
- The shortcut for Open, and then Offline is Ctrl + F.
- The shortcut for Open, and then Online is Ctrl + L.
- The shortcut for Save is Ctrl + S.
- The shortcut for Check Global Consistency is Ctrl + K.

Edit Menu Shortcuts

- The shortcut for Cut is Ctrl + X.
- The shortcut for Copy is Ctrl + C.
- The shortcut for Paste is Ctrl + V.
- The shortcut for Delete is Delete.

View Menu Shortcut

- The shortcut for Refresh is F5.

Tools Menu Shortcuts

- The shortcut for Show Consistency Checker is Ctrl + E.
- The shortcut for Query Repository is Ctrl + Q.

General Menu Shortcuts

Table I–1 lists the general keyboard shortcuts available in the Administration Tool menus. Note that you can use the Window menu options to change the focus from the menus to the navigation panes.

Table I–1 Menu Keyboard Shortcuts

Action	Keyboard Shortcut
Quit the application	Alt+ F4
Move cursor to the menu option	Alt + Underlined letter
Open application's control menu	Alt+ Spacebar
View the shortcut menu for the selected item	Shift + F10
Move through the menu bar	Left arrow key Right arrow key
Open a menu option	Down arrow key
Move through a menu list	Up arrow Down arrow
Close the current menu	Esc
Select or deselect items in a check box or list	Spacebar
Make noncontinguous selections	Ctrl + Up arrow + Spacebar

Dialog Keyboard Shortcuts

Table I–2 lists the keyboard shortcuts available in Administration Tool dialogs.

Table I–2 Dialog Keyboard Shortcuts

Action	Keyboard Shortcut
Move forward through options	Tab
Move backward through options	Shift + Tab
Select or deselect an item in a list	Shift + Up arrow Shift + Down arrow
Close the current dialog	Esc
Go to the top of a list	Home
Go to the bottom of a list	End
Refresh	F5
For dialogs with up arrow buttons: Move selected item up in the list	Alt + Up arrow **Note:** Select a list item before using this shortcut.

Table I–2 (Cont.) Dialog Keyboard Shortcuts

Action	Keyboard Shortcut
For dialogs with down arrow buttons: Move selected item down in the list	Alt + Down arrow **Note:** Select a list item before using this shortcut.
For dialogs with plus (add) buttons: Insert item from list	Alt + Insert
For dialogs with x (delete) buttons: Delete item from list	Alt + Delete
For dialogs with pencil (edit) buttons: Edit item from list	Alt + Enter
Browse dialog: Move focus between trees located in left pane	F5, F6, Shift + Tab, Tab
When a table row has a child row (grid): Expand the child row from the cell displaying the plus icon This situation occurs in the Define Merge Strategy page of the Merge Repository Wizard.	Spacebar **Note:** Move the focus to the cell displaying the plus icon before using this shortcut.
When a table row has a check box: Select or deselect the check box This situation occurs in the Define Merge Strategy page of the Merge Repository Wizard.	Spacebar **Note:** Move the focus to the cell displaying the check box before using this shortcut.

Physical Diagram and Business Model Diagram Keyboard Shortcuts

Table I–3 lists the keyboard shortcuts available in the Physical and Business Model Diagrams. Note that the Physical and Business Model Diagram toolbar options are also available from the Diagram menu.

Table I–3 Diagram Keyboard Shortcuts

Action	Keyboard Shortcut
Pan around the diagram when no diagram objects are selected	Arrow keys
Select a diagram object: use the arrow keys to move an object under the pointer, then press the spacebar to select the object	Spacebar + Arrow keys
Open the property dialog for a selected diagram object	Enter
Cancel current operation	Esc

Table I–3 (Cont.) Diagram Keyboard Shortcuts

Action	Keyboard Shortcut
Resume default mode (Select) after using Pan or Marquee Zoom	Esc
Deselect an object	Use one of the following methods: Esc Press the spacebar when the mouse cursor is not over an object
Zoom in	+
Zoom out	-
Select the pan tool	P Note that you can also use the arrow keys to pan around the diagram.
Revert to auto-layout	S
Create a new join	J This shortcut selects the New Join option.
Create a new table	N This shortcut selects the New Table option. After using this shortcut, you can use the arrow keys and spacebar to pan around the diagram and open the Logical Table dialog for the new table.
Select the Marquee Zoom tool	Z This shortcut selects the Marquee Zoom tool.
Zoom to fit all objects in the current view	F
Show all tables in Expanded View, with columns visible	1
Show all tables in Collapsed View, with columns hidden and only the table name displayed	2

J

Administration Tool User Interface Components

This appendix serves as an icon reference and provides a set of lists containing icons used in the Administration Tool. It also provides references to the topics that describe how the elements corresponding to the icons are used for metadata repository development.

This appendix contains the following topics:

About This Icon Reference

In this icon reference, the icons are grouped in general according to the Administration Tool layer or window in which they appear.

Icons in Multiple Windows

Some icons appear in more than one Administration Tool window, such as the Expression Builder icon:

Other icons that appear in more than one window might represent different stages or usages of an object - such as the physical, logical, and presentation versions of a table.

Overlay Symbols

Table J–1 describes Administration Tool symbols that overlay other icons. These can appear in different Administration Tool windows.

Table J–1 Overlay Symbols

Symbol	Description
Reference Shortcut 	Represents a reference shortcut to an existing repository object in the Administration Tool.
Checked Out Object 	Registers that an object has been checked out in a repository open in online mode.

Icons Listed Elsewhere

Not all the Administration Tool icons are described in this icon reference. The main exceptions are as follows:

- Certain Administration Tool icons are described elsewhere in this manual, such as the following:
  - Table 2–5, " Toolbar Options for the Physical and Business Model Diagrams"
  - Table 4–1, " Options for Managing Environment Variables in the SCM Configuration Editor"
  - Table 17–4, " Elements in the Define Merge Strategies Screen"
- This icon reference only identifies icons that relate to metadata repository development. It does not cover icons for objects and functions covered in other guides, such as icons related to jobs, sessions, cache entries, clusters, and icons related to Oracle Marketing Segmentation. The guides for these objects and functions appear in the options described in Manage Menu.
- You can change the icons associated with many repository objects, as described in Changing Icons for Repository Objects. This icon reference lists the icon-object associations as released with Oracle Business Intelligence, and before any change.

Icons for the Physical Layer

This section lists icons for the Physical Layer.

Table J–2 Icons for the Physical Layer

Icon	Reference
Analytic Workspace 	Working with Oracle OLAP Analytic Workspace (AW) Objects
Physical Column 	Creating and Managing Columns and Keys for Relational and Cube Tables
Connection Pool 	Creating or Changing Connection Pools
Cube Table 	Creating and Managing Physical Tables and Physical Cube Tables

Table J–2 (Cont.) Icons for the Physical Layer

Icon	Reference
Cube Column 	Creating and Managing Columns and Keys for Relational and Cube Tables
Cube Table View 	Deploying Opaque Views
Database 	Setting Up Database Objects
Database with cubes 	Working with Multidimensional Sources in the Physical Layer
Oracle OLAP database 	Working with Oracle OLAP Data Sources
Physical Dimension 	About Working with Physical Dimensions and Physical Hierarchies
Physical Display Folder (Closed) 	Setting Up Display Folders in the Physical Layer
Physical Display Folder (Open) 	Setting Up Display Folders in the Physical Layer
Physical Hierarchy 	About Working with Physical Dimensions and Physical Hierarchies
Key 	Creating and Managing Columns and Keys for Relational and Cube Tables
Multidimensional Key 	Creating and Managing Columns and Keys for Relational and Cube Tables
Physical Catalog 	Creating Physical Layer Catalogs and Schemas
Physical Level 	Working with Physical Hierarchy Objects

Table J–2 (Cont.) Icons for the Physical Layer

Icon	Reference
Physical Schema 	Creating Physical Layer Catalogs and Schemas
Physical Table 	Working with Physical Tables
Physical Alias Table 	About Physical Alias Tables
Parent-Child Relationship Table 	Defining Parent-Child Relationship Tables
Opaque View 	Deploying Opaque View Objects
Stored Procedure 	About Tables in the Physical Layer

Icons for the Business Model and Mapping Layer

This section lists icons for the Business Model and Mapping Layer.

Table J–3 Icons for the Business Model and Mapping Layer

Icon	Reference
Aggregate 	Setting Default Levels of Aggregation for Measure Columns
Business Model 	Creating Business Models
Business Model (Disabled) 	Creating Business Models
Logical Column 	Creating Logical Columns
Logical Column (Derived) 	Creating Derived Columns

Table J–3 (Cont.) Icons for the Business Model and Mapping Layer

Icon	Reference
Logical Dimension 	Working with Logical Dimensions
Logical Display Folder (Closed) 	Setting Up Display Folders in the Business Model and Mapping Layer
Logical Display Folder (Open) 	Setting Up Display Folders in the Business Model and Mapping Layer
Invalid Object 	Indicates a serious problem with an object that requires user attention (for example, an object needs to be edited or deleted)
Logical Key 	Specifying a Primary Key in a Logical Table
Logical Level 	Creating Logical Levels in a Dimension
Logical Level Key 	Creating Logical Levels in a Dimension
Logical Lookup Table 	Creating Logical Tables
Logical Table 	Creating Logical Tables
Logical Fact Table 	Defining Logical Joins
Logical Table Source 	Creating and Managing Logical Table Sources

Icons for the Presentation Layer

This section lists icons for the Presentation Layer.

Table J–4 Icons for the Presentation Layer

Icon	Reference
Aggregate 	Creating and Managing Presentation Columns
Presentation Column 	Creating and Managing Presentation Columns
Presentation Column (Derived) 	Creating and Managing Presentation Columns
Presentation Hierarchy 	Creating and Managing Presentation Hierarchies
Key 	Creating and Managing Presentation Columns
Presentation Level 	Creating and Managing Presentation Levels
Subject Area 	About Creating Subject Areas
Presentation Table 	Creating and Managing Presentation Tables

Icons for the Identity Manager

This section lists icons for the Identity Manager.

Table J–5 Icons for the Identity Manager

Icon	Reference
Identity Manager folder 	About Data Access Security
BI Repository folder 	About Data Access Security
Adaptors folder 	*Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*

Table J–5 (Cont.) Icons for the Identity Manager

Icon	Reference
Directory Servers 	*Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*
Custom Authenticators 	*Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*
Users 	Setting Up Object Permissions
Application Roles 	Setting Up Object Permissions

Icons for the Joins Manager

This section lists icons for the Joins Manager.

Table J–6 Icons for the Joins Manager

Icon	Reference
General Joins folder 	Defining Logical Joins with the Joins Manager Defining Physical Joins with the Joins Manager
Foreign Key (Logical and Physical) 	Creating Logical Foreign Key Joins with the Joins Manager Defining Physical Joins with the Joins Manager
Logical Join 	Defining Logical Joins with the Joins Manager
Complex Join 	Defining Logical Joins with the Joins Manager Defining Physical Joins with the Joins Manager
Expression Builder 	Using Expression Builder

Icons for the Variable Manager

This section lists icons for the Variable Manager.

Table J–7 Icons for the Variable Manager

Icon	Reference
Repository folder 	Working with Repository Variables
Repository Initialization Blocks 	Creating Initialization Blocks
Repository Variables and Static Variables folders 	About Repository Variables About Static Repository Variables
Dynamic Variables 	About Dynamic Repository Variables
Static Variables 	About Static Repository Variables
Session folder 	Working with Session Variables
Session Initialization Blocks 	Creating Initialization Blocks
Session Variables folder, Non-System Variables *(folder + occurrences)* 	About Session Variables Working with Multi-Source Session Variables About Nonsystem Session Variables
Session System Variables 	About System Session Variables
Session System Security Variables 	About System Session Variables

Icons for the Project Manager

This section lists icons for the Project Manager.

Table J–8 Icons for the Project Manager

Icon	Reference
Project 	Setting Up Projects
Layers and elements from which project objects can be created: 	About the Project Dialog