Oracle BI 11*g* R1: Build Repositories

Volume I - Student Guide

D63514GC10
Edition 1.0
October 2010
D69303

ORACLE

Author

Jim Sarokin

Technical Contributors and Reviewers

Marla Azriel
Roger Bolsius
Bob Ertl
Alan Lee
Monica Moore
Kasturi Shekhar
Aneel Shenker
Scott Silbernick
Nick Tuson
Krishnan Viswanathan

Graphic Designer

Rajiv Chandrabhanu

Editors

Aju Kumar
Daniel Milne
Richard Wallis

Publishers

Syed Ali
Giri Venugopal

Contents

1

Course Introduction

Lesson Agenda

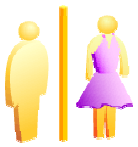
This lesson provides an introduction to the following:

- Instructor and class participants
- Training site information
- Course:
  - Audience
  - Prerequisites
  - Goal
  - Objectives
  - Methodology
  - Materials
  - Agenda

Instructor and Class Participants

- Who are you?
  - Name
  - Company
  - Role
- What is your prior experience?
  - Business intelligence
  - Data warehouse design
  - Database design
  - Oracle BI applications
- How do you expect to benefit from this course?

Training Site Information

- Restrooms 
- Telephones 
- Fire exits 
- Class duration and breaks 
- Meals and refreshments 
- Questions 

Course Audience

This course is intended for:

- Application developers
- Business intelligence developers
- Business analysts
- Data modelers
- Database designers
- Data warehouse analysts
- Technical consultants

Course Prerequisites

Recommended Oracle University courses:

- *Oracle BI 11g R1: Create Analyses and Dashboards*

Recommended experience:

- Business intelligence
- Data warehouse design
- Data modeling
- Database design
- Basic SQL

Course Goal

To enable students to build and deploy an Oracle Business Intelligence repository

Course Objectives

- Build the Physical, Business Model and Mapping, and Presentation layers of a repository.
- Set up query logging for testing and debugging.
- Build and run analyses to test and validate a repository.
- Manage multiple logical table sources in a repository.
- Build simple and calculated measures for a fact table.
- Create logical dimensions with level-based hierarchies and level-based measures.
- Create logical dimensions with parent-child hierarchies.
- Model aggregate tables to speed up processing.
- Model partitions and fragments to improve application performance and usability.

ORACLE

Course Objectives

- Use variables to streamline administrative tasks and dynamically modify metadata content.
- Use time series functions to support historical time comparison analyses.
- Use bridge tables to resolve many-to-many relationships between dimension and fact tables.
- Configure Oracle BI to support multilingual environments.
- Use implicit fact columns to select fact tables for dimension-only queries.
- Create a repository by using a multidimensional data source.

Course Objectives

- Set up security to authenticate users and assign appropriate permissions and privileges.
- Apply cache management techniques to maintain and enhance query performance.
- Enable usage tracking to track queries and database usage and to improve query performance.
- Set up and use a multiuser development environment.
- Set up and configure “write back” to enable users to modify data in analyses.
- Perform a repository patch merge in a development-to-production scenario.
- Define user-preferred currency options.

Course Objectives

- Use Administration Tool utilities and wizards to perform administrative tasks.
- Identify best practices for optimizing end-user query performance when implementing Oracle BI.
- Identify and apply repository design principles and best practices.

Course Methodology

The subject matter is delivered through:

- Lectures and slide presentations
- Software demonstrations
- Class discussions
- Hands-on practices

Course Materials

Student Guide

- All slides presented during lectures

Activity Guide

- Hands-on practices

Course Agenda

Day One

- Lesson 1: Course Introduction
- Lesson 2: Repository Basics
- Lesson 3: Building the Physical Layer of a Repository
- Lesson 4: Building the Business Model and Mapping Layer of a Repository
- Lesson 5: Building the Presentation Layer of a Repository
- Lesson 6: Testing and Validating a Repository

ORACLE

Course Agenda

Day Two

- Lesson 7: Managing Logical Table Sources
- Lesson 8: Adding Calculations to a Fact
- Lesson 9: Working with Logical Dimensions
- Lesson 10: Using Aggregates
- Lesson 11: Using Partitions and Fragments

Course Agenda

Day Three

- Lesson 12: Using Repository Variables
- Lesson 13: Modeling Time Series Data
- Lesson 14: Modeling Many-to-Many Relationships
- Lesson 15: Localizing Oracle BI Metadata
- Lesson 16: Localizing Oracle BI Data

Course Agenda

Day Four

- Lesson 17: Setting an Implicit Fact Column
- Lesson 18: Importing Metadata from Multidimensional Data Sources
- Lesson 19: Security
- Lesson 20: Cache Management
- Lesson 21: Managing Usage Tracking

Course Agenda

Day Five

- Lesson 22: Setting Up and Using the Multiuser Development Environment
- Lesson 23: Configuring Write Back in Analyses
- Lesson 24: Performing a Patch Merge
- Lesson 25: Configuring Logical Columns for Multicurrency Support
- Lesson 26: Using Administration Tool Utilities

Summary

This lesson provided an introduction to the following:

- Instructor and class participants
- Training site information
- Course:
  - Audience
  - Prerequisites
  - Goal
  - Objectives
  - Methodology
  - Materials
  - Agenda

2

Repository Basics

Objectives

After completing this lesson, you should be able to:

- Identify and describe the major components of the Oracle Business Intelligence (BI) architecture
- Identify and describe the three layers of an Oracle BI repository and their relationship to each other
- Use the Oracle BI Administration Tool to explore repository objects
- Publish a repository
- Start and stop Oracle BI components

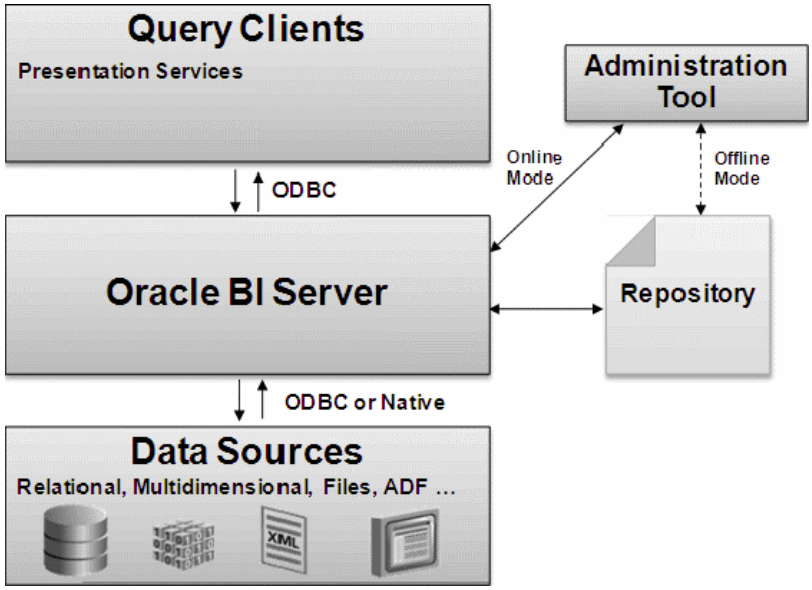
ORACLE

Objectives

This lesson provides foundational information to set the context for the remainder of this course. An understanding of architecture components, the basics of the Administration Tool, and repository structure should enable you to build a working Oracle BI repository.

Oracle BI Server Architecture

This lesson provides a high-level overview of Oracle BI Server architecture and related components.



Oracle BI Server Architecture

This lesson provides a high-level overview of Oracle BI Server architecture and related BI components. Oracle BI Server and related BI components are part of Oracle BI Foundation Suite, which provides a complete, open, and integrated solution for all enterprise BI needs. An understanding of these components and their relationships helps you successfully complete this course. Each component is discussed in more detail in the slides that follow.

Note that this lesson does not provide an exhaustive discussion of Oracle BI architecture. It discusses only a subset of the components that are critical to understanding the subject matter in this course. For more extensive information, see the following documentation:

- *Oracle Fusion Middleware Metadata Repository Builder's Guide for Oracle Business Intelligence Enterprise Edition*
- *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition*
- *Oracle Fusion Middleware User's Guide for Oracle Business Intelligence Enterprise Edition*
- *Oracle Fusion Middleware Security Guide for Oracle Business Intelligence Enterprise Edition*
- *Oracle Fusion Middleware Integrator's Guide for Oracle Business Intelligence Enterprise Edition*
- *Oracle Fusion Middleware Developer's Guide for Oracle Business Intelligence Enterprise Edition*
- *Oracle Fusion Middleware Installation Guide for Oracle Business Intelligence*
- *Oracle Fusion Middleware Upgrade Guide for Oracle Business Intelligence*

Oracle BI Presentation Services

- Provides the framework and interface for the presentation of business intelligence data to Web clients
- Is a set of query clients that includes the following components:
  - Analysis Editor
    - Set of graphical tools that enable users to build, view, and modify analyses that provide analytic information
  - Dashboards
    - Display the results of analyses and other items

Oracle BI Presentation Services

Oracle BI Presentation Services provides the framework and interface for the presentation of business intelligence data to Web clients. It is a stand-alone process and communicates with Oracle BI Server using Open Database Connectivity (ODBC) over TCP/IP. It consists of a set of query clients that includes the Analysis Editor and Interactive Dashboards.

The Analysis Editor enables users with the appropriate permissions to build and modify analyses that provide end users with the ability to explore and interact with information. An analysis can present information in various formats, such as tables, pivot tables, graphs, maps, and gauges; the results of an analysis can be enhanced by adding calculated items and drilling. Prebuilt analyses can be used out of the box or modified to suit your business's information needs. Analyses are saved in the Oracle BI Presentation Catalog and integrated into any Oracle Business Intelligence dashboard. The recipient of an analysis can format the analysis' results, output the results to another data format, save the results, and share the analysis' results with other users. An analysis can be configured to refresh results in real time. You use the Analysis Editor throughout this course to create and run analyses that test the business models you build in the Oracle BI repository.

Dashboards provide an end user with access to analytics information, including the results of analyses that are built in to the Analysis Editor. Users with appropriate permissions can place results from analyses into dashboards for use by end users. Your company might also have purchased preconfigured dashboards that contain prebuilt analyses that are specific to your industry.

Oracle BI Server

- Is the core server behind Oracle Business Intelligence
- Provides efficient processing to access physical data sources and structure information intelligently:
  - Uses metadata to direct processing
  - Generates dynamic SQL to query data in the physical data sources
  - Connects natively or through ODBC to a data source
  - Structures results to satisfy requests
  - Provides BI data to Oracle BI Presentation Services

Oracle BI Server

Oracle BI Server is the core server behind Oracle Business Intelligence. It is an optimized query engine that receives analytical requests, intelligently accesses multiple physical data sources, generates SQL to query data in the data sources, and then structures the results to satisfy the requests. It also handles requests from a variety of front ends, including Oracle BI applications as well as third-party tools. Oracle BI Server allows a single information request to query multiple data sources, providing information access to members of the enterprise and (in Web-based applications) to suppliers, customers, prospects, or any authorized user with Web access.

Oracle BI Server serves as a portal to structured data that resides in one or more data sources: multiple data marts, an enterprise data warehouse, an operational data store, transaction system databases, personal databases, and so on. Transparent to both end users and query tools, Oracle BI Server functions as the integrating component of a complex decision support system by acting as a layer of abstraction and unification over the underlying databases. This offers users a simplified query environment in which they can ask business questions that span information sources across the enterprise and beyond.

Data Sources

- Contain the business data that users want to analyze
- Are accessed by Oracle BI Server
- Can be in multiple formats, such as:
  - Relational databases
  - Online analytical processing (OLAP) databases
  - Flat files
  - Multidimensional data sources

Data Sources

Data sources are the physical sources where business data is stored. They can be in any format, including transactional databases, OLAP databases, text files, multidimensional data sources such as Essbase, spreadsheets, and so on. A connection to the data source is created and then used by Oracle BI Server. The data source connection can be defined to use native drivers or Open Database Connectivity (ODBC). You learn more about this when you create the Physical layer of a repository in the lesson titled “Building the Physical Layer of a Repository.”

SQL is generated by Oracle BI Server against the data sources by using the data source connection, information from the repository, and database-specific features and parameters. As a result, Oracle BI Server is not just a SQL generator; it determines the best source and the optimal way to access data. In some cases, Oracle BI Server performs operations more efficiently than the physical data sources do.

Oracle BI Repository

- Stores the metadata used by Oracle BI Server
- Is accessed and configured by using the Oracle BI Administration Tool, which you use to:
  - Import metadata from databases and other data sources
  - Simplify and reorganize the metadata into business models
  - Structure the business model for presentation to users who request information

ORACLE

Oracle BI Repository

Oracle BI Server stores metadata in repositories. The Oracle BI Administration Tool has a graphical user interface that allows server administrators to set up these repositories.

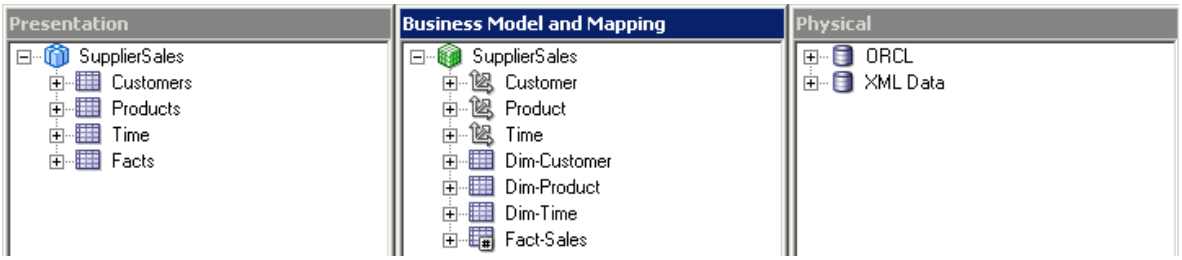
In this course, you learn how to use this Administration Tool to:

- Import metadata from databases and other data sources
- Simplify and reorganize the imported metadata into business models
- Structure the business model for presentation to users who request business intelligence information via Oracle BI end-user tools

Oracle BI Administration Tool

Exposes the Oracle BI repository as three separate panes called *layers*:

- Physical
- Business Model and Mapping (BMM)
- Presentation

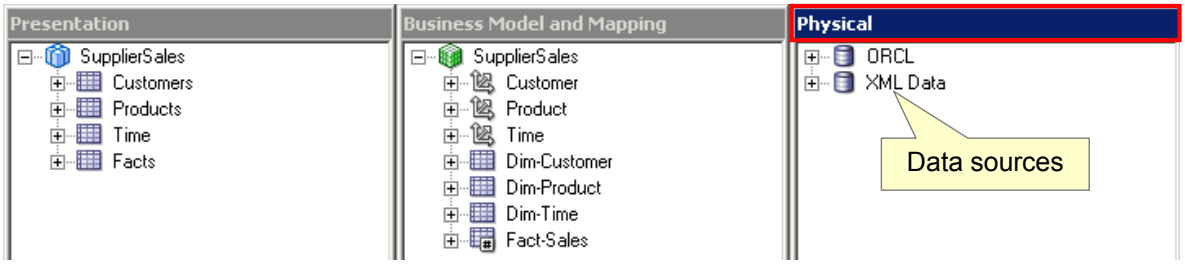


Oracle BI Administration Tool

You develop a repository by using the Oracle BI Administration Tool. The main window of the Administration Tool provides a graphical representation of the three layers of a repository. In this course, you learn how to use the Administration Tool to build, edit, manage, and administer Oracle BI repositories. Each layer of the repository is explained in more detail on the slides that follow.

Physical Layer

- Contains objects representing the physical data sources to which Oracle BI Server submits queries
- May contain multiple data sources
- Is typically the first layer built in the repository



Physical Layer

The Physical layer contains information about the physical data sources to which Oracle BI Server submits queries. The most common way to populate the Physical layer is by importing metadata from databases and other data sources. The data sources can be of the same or different varieties.

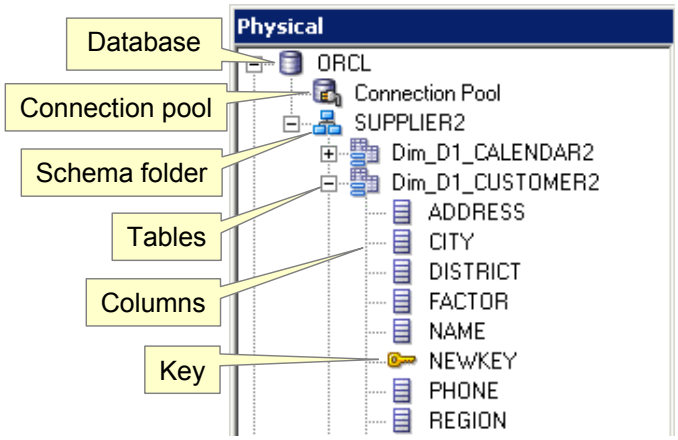
When you import metadata, many of the properties of the data sources are configured automatically based on the information gathered during the import process. After import, you can also define other attributes of the physical data source (such as join relationships) that might not exist in the data source metadata.

There can be one or more data sources in the Physical layer, including databases, spreadsheets, and XML documents. In this example, there are two data sources in the Physical layer: `ORCL` and `XML Data`.

Objects in the Physical Layer

Are objects in the Physical layer, such as connection pools, folders, tables, columns, and keys

- Expand a database object to display the objects that it contains:



Objects in the Physical Layer

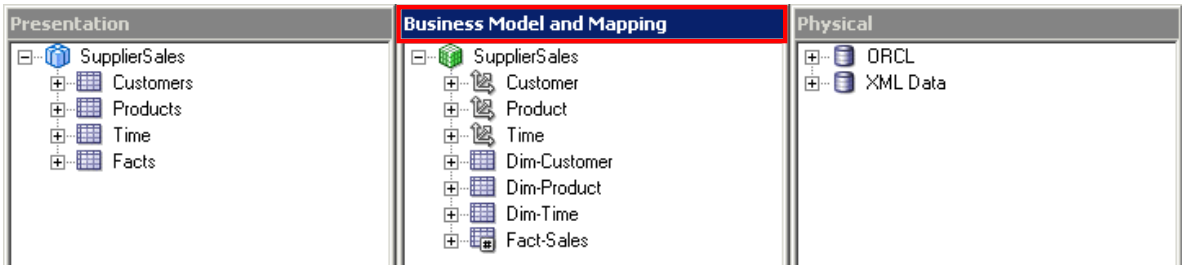
The database object is the highest object in the Physical layer. Each database object in the Physical layer contains a connection pool. A connection pool contains information about the connection between Oracle BI Server and the data source. For example, the connection pool contains data source name information that is used to connect to a data source, username and password, the number of connections allowed, timeout information, and other connectivity-related administrative details. There is at least one connection pool for each data source.

You can organize the Physical layer into folders. The folders shown in the slide are created by default when a schema is first imported. Additional folders are optional. A schema folder (`SUPPLIER2` in this example) contains tables, columns, and keys for a physical schema. The table, column, and key objects provide the metadata that is necessary for Oracle BI Server to access the actual physical sources with SQL requests.

Each object in the Physical layer is associated with a set of properties. To view the properties, double-click the object's icon to open the Properties dialog box (not shown here).

Business Model and Mapping Layer

Is where physical schemas are simplified and reorganized to form the basis of the user's view of the data



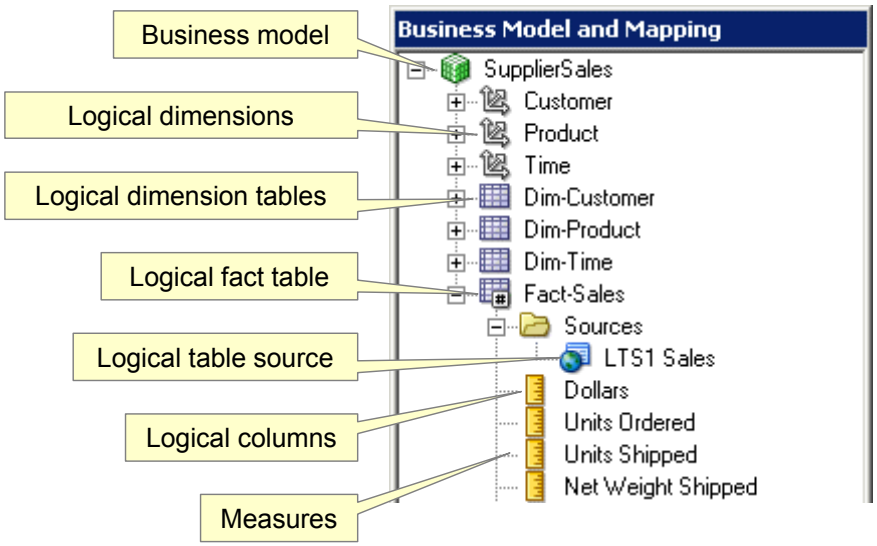
Business Model and Mapping Layer

The Business Model and Mapping layer of the Administration Tool defines the business (or logical) models of the data and specifies the mapping between the business models and the Physical layer schemas. This is where the physical schemas are simplified to form the basis for the users' view of the data.

The Business Model and Mapping layer of the Administration Tool contains one or more business model objects. A business model object contains the business model definitions and the mappings from logical to physical tables for the business model.

Objects in the Business Model and Mapping Layer

The Business Model and Mapping layer contains business model objects.



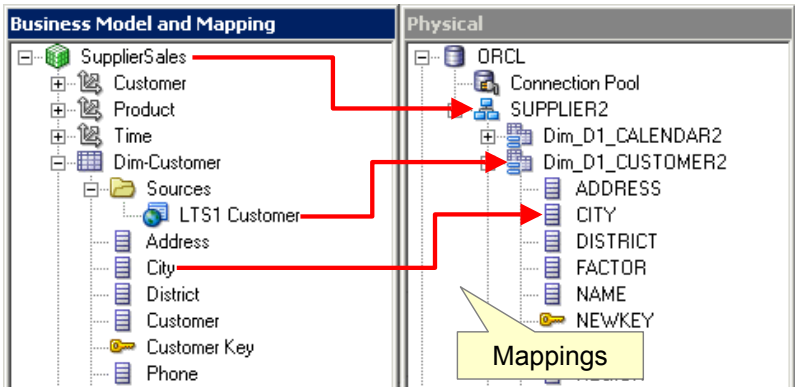
Objects in the Business Model and Mapping Layer

The Business Model and Mapping Layer contains business model objects. There can be multiple business models in this layer, although there is only one in the example in the slide. `SupplierSales` , the business model object in this example, is the highest-level object and contains logical tables: `Dim-Customer`, `Dim-Product`, `Dim-Time`, and `Fact-Sales`. Logical tables contain logical columns. Logical tables also have a `Sources` folder that contains one or more logical table sources. These logical table sources define the mappings between the logical table and the physical tables where the data is actually stored. In this example, the logical table source for the `Fact-Sales` logical table is named `LTS1 Sales` and is mapped to a single physical table in the Physical layer. You can create business model objects by dragging objects from the Physical layer.

In the Business Model and Mapping layer, data is separated and simplified into facts and dimensions. Recall that facts contain the business measures and dimensions contain descriptive attributes that qualify the facts. Each business model contains one or more dimension tables and fact tables. A fact table icon appears with a # sign. The business model also contains logical dimensions (in this example, `Customer`, `Product`, and `Time`). Logical dimensions introduce formal hierarchies into a business model, establish levels for data groupings and calculations, and provide drill-down paths in end-user tools.

Business Model Mappings

- Business Model and Mapping layer objects map to source data objects in the Physical layer.
- Mappings are typically not one-to-one:
  - Business models may map to multiple data sources.
  - Logical tables may map to multiple physical tables.
  - Logical columns may map to multiple physical columns.



Business Model Mappings

The Business Model and Mapping layer of the Administration Tool can contain one or more business model objects. A business model object contains other business model object definitions and the mappings from logical to physical tables for the business model. `SupplierSales` is the business model object in this example. Business model objects map to the source data objects in the Physical layer: business models map to physical schemas, logical tables map to physical tables, logical columns map to physical columns, and so on. The arrows convey the mappings.

Typically there is not a one-to-one mapping between business model objects and physical objects. A business model can have multiple data sources, a logical table can have multiple physical tables as sources, and a logical column can have multiple physical columns as sources. In the Administration Tool, the mappings are specified in the logical Business Model and Mapping layer, not the Physical layer. The mappings can also include transformations and formulas. Thus, you use the Business Model and Mapping layer to define the meaning and content of each physical source in business-model terms. Oracle BI Server then uses these business model definitions to pick the appropriate sources for each data request.

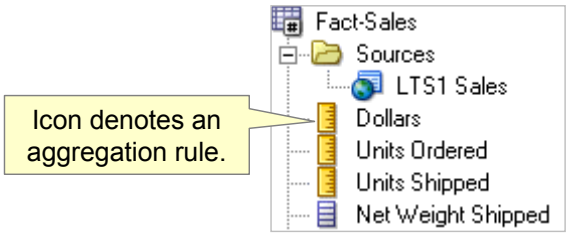
Business Model Mappings (continued)

For more information about business model mappings, see the lesson titled “Building the Business Model and Mapping Layer of a Repository.”

Also note that, in this example, the business model objects have been renamed. For example, the name of the logical table source that maps to the `Dim_D1_CUSTOMER2` physical table has been changed to `LTS1 Customer`. The logical table name has been changed to `Dim-Customer`. You can change the names of business model objects independently from corresponding physical model object names and properties. Likewise, you can change the names of physical model objects and properties independently from the names of business model objects. The tool keeps track of the changes. Techniques for making these changes are covered in Lesson 4: Building the Business Model and Mapping Layer of a Repository.

Measures

- Are the facts that a business uses to evaluate its performance
- Can be calculations that define measurable quantities
- Are created on logical columns in the fact table
- Typically have a defined aggregation rule



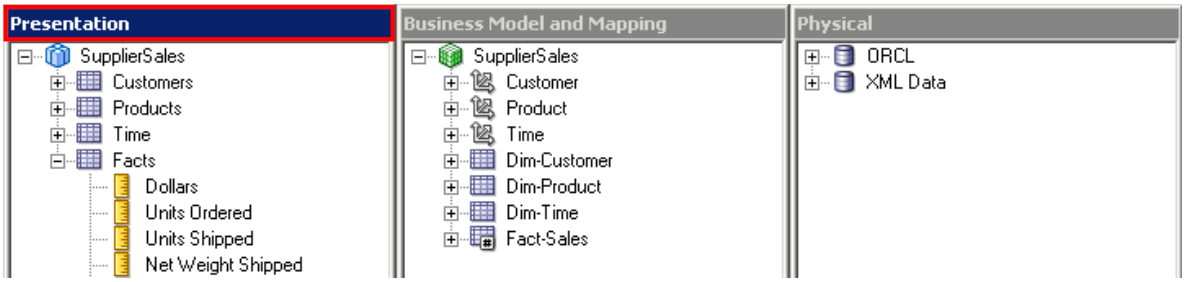
Measures

In a business model, some logical columns serve as measures. Measures are the facts that a business uses to evaluate its performance. Measures need to be computed correctly from the base physical data when Oracle BI Server receives a query. By default, a column has no aggregation rule defined. When you define an aggregation rule on a column, the icon changes to reflect this. After you define an aggregation rule on a column, the aggregation is the default in end-user tools.

In the example in the slide, the sum aggregation rule has been defined for the `Dollars`, `Units Ordered`, and `Units Shipped` columns. The icons have changed to reflect this. The calculations are sent to the database or processed by Oracle BI Server. Measures may not require any calculations or aggregation. For example, a logical column may simply return data directly from a source.

Presentation Layer

- Contains presentation objects that provide a customized view of a business model to users
- Simplifies and organizes the business model to make it easy for users to understand and query
- Exposes only the data that is meaningful to the users

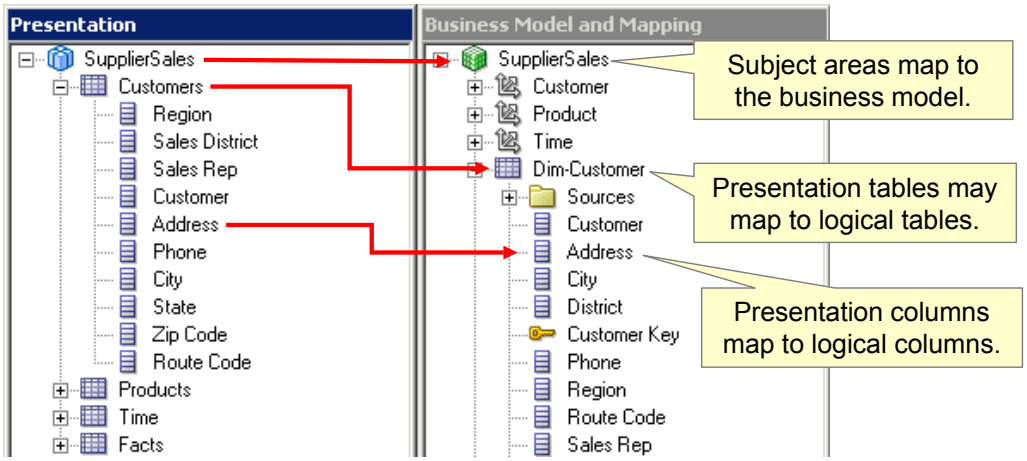


Presentation Layer

The Presentation layer provides a means to further simplify or customize the business model for end users. Simplifying the view of the data for users makes it easier for users to craft queries based on their business needs. For example, you can organize columns differently than the table structure in the Business Model and Mapping layer, show fewer columns, and rename columns so that they are more understandable to users.

Presentation Layer Mappings

Presentation layer objects map to objects in the Business Model and Mapping layer.



Presentation Layer Mappings

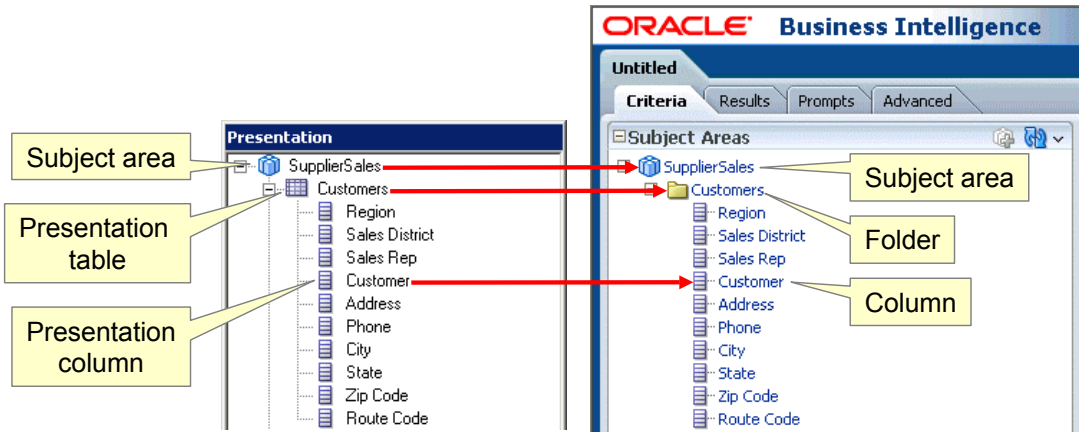
Presentation layer objects map to objects in the Business Model and Mapping layer. Subject areas map to business models, presentation tables may map to logical tables, and presentation columns map to logical columns. A subject area can map to only a single business model. However, multiple subject areas can reference the same business model, and a presentation table can contain columns from one or more logical tables. As a result, you can use subject areas and tables to organize columns into categories that users can understand.

Presentation tables do not necessarily have a one-to-one mapping to a logical table. Presentation tables are folders that are used for organizing columns and can be organized and structured differently from logical tables in the Business Model and Mapping layer.

You can create Presentation layer objects by dragging objects from the Business Model and Mapping layer. Corresponding objects are automatically created in the Presentation layer. The presentation table and column names are, by default, identical to the logical names in the Business Model and Mapping layer, but they can be renamed. Thus, the names and object properties of the presentation tables are independent of the logical table properties. It is also possible to nest presentation tables so they appear as nested folders in the user interface.

Presentation Layer Defines the User Interface

Presentation layer objects define the interface that users see to query data from data sources.



Presentation Layer Defines the User Interface

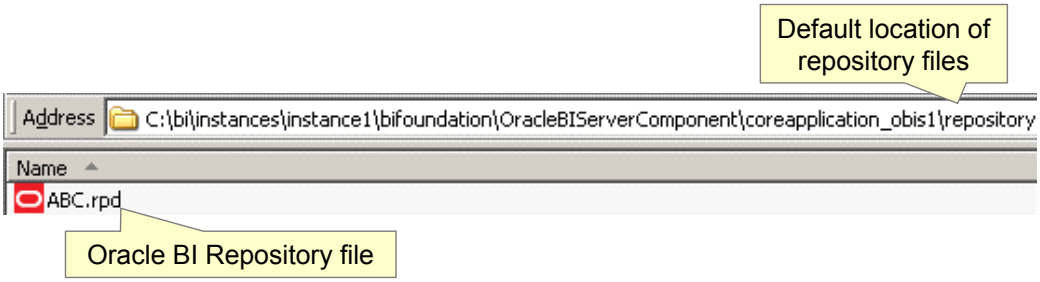
Presentation layer objects define the interface that users see to query the data from data sources:

- Subject areas in the Presentation layer are displayed as subject areas.
- Presentation tables in the Presentation layer are displayed as folders.
- Presentation columns are displayed as columns within the folders.

Repository Directory

By default, repositories are stored in the `repository` subdirectory where Oracle BI software is installed:

ORACLE_INSTANCE\bifoundation\
OracleBIServerComponent\
*bi_instance_name_obisn***repository**



Repository Directory

By default, new repositories are stored in the repository subdirectory, which is located at *ORACLE_INSTANCE*\bifoundation\OracleBIServerComponent*bi_instance_name_obisn*\repository (where the Oracle BI software is installed). ORACLE_INSTANCE is the path to the Oracle Fusion Middleware Oracle instance where data related to the Oracle BI system components is stored. For more information, refer to the *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition.*

From this directory, repository files are loaded by Oracle BI Server or they can be opened for editing. You use the Administration Tool to open repository files for editing. Repository files can be opened for editing in two modes (which are discussed in the next slide). You use Fusion Middleware Control Enterprise Manager to load repositories (see the slide after next).

Repository Modes

Repository files can be opened for editing in offline mode or online mode.

- Offline mode
  - Repository is not loaded into Oracle BI Server memory.
- Online mode
  - Repository is loaded into Oracle BI Server memory.
  - Administrators can perform tasks that are not available in offline mode:
    - Manage scheduled jobs.
    - Manage user sessions.
    - Manage the query cache.
    - Manage clustered servers.

ORACLE

Repository Modes

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

Offline Mode

Use offline mode to view and modify a repository while it is not loaded into Oracle BI Server. If you attempt to open a repository in offline mode while it is loaded into 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.

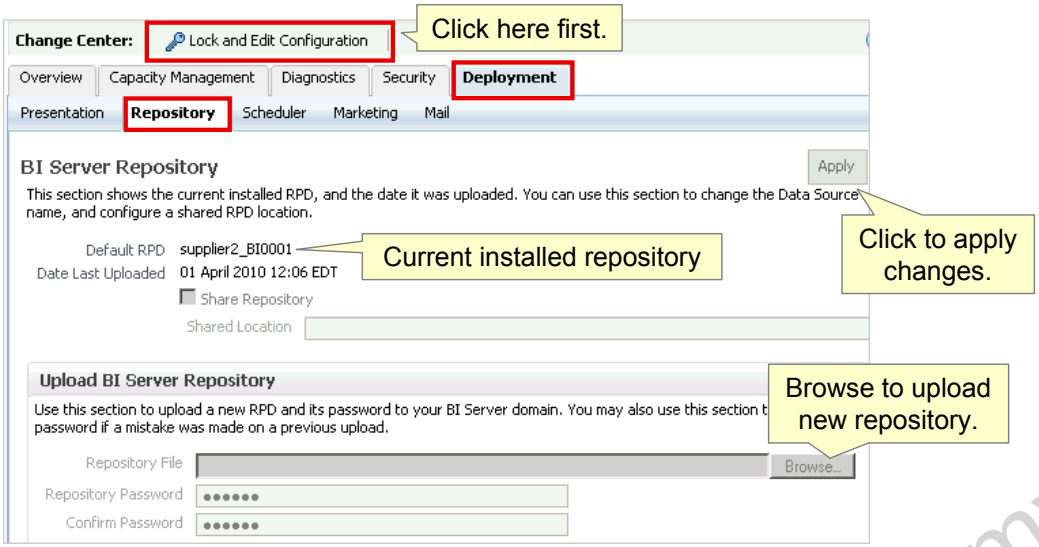
Online Mode

Use online mode to view and modify a repository while it is loaded into Oracle BI Server, which must be running to open a repository. 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 scheduled jobs.
- Manage user sessions.
- Manage the query cache.
- Manage clustered servers.

Publishing a Repository

Use Fusion Middleware Control to publish a repository and make it available for queries.



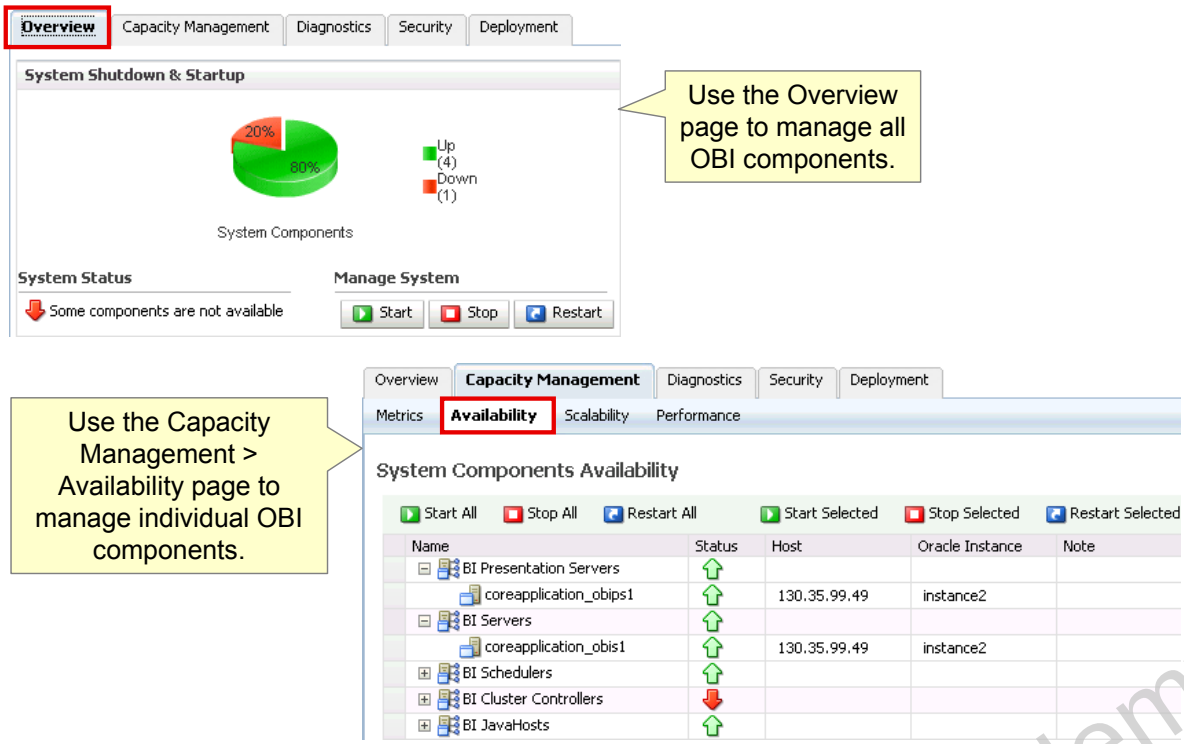
Publishing a Repository

After you build or modify a repository in offline mode, you use Fusion Middleware (FMW) Control Enterprise Manager to publish the repository. FMW Control is a Web-based centralized administration console that you can use to start, stop, and manage BI processes and perform many administration tasks. You learn how to access Enterprise Manager and publish repositories in the lesson titled "Testing and Validating a Repository."

Publishing a repository enables Oracle BI Server to load the repository into memory at startup and makes the repository available for queries by end users. When you upload a repository file for publishing, you provide the name and location of the repository that you want to upload, as well as the current repository password.

To use Enterprise Manager to upload a repository and set repository parameters, display the Repository tab of the Deployment page. On this tab, you can view the current default repository and the date on which it was last uploaded. To upload a new repository, click "Lock and Edit Configuration." In the Upload BI Server Repository section, click Browse. The Browse dialog box should be set to the default repository directory. If not, go to `ORACLE_INSTANCE\bifoundation\OracleBIServerComponent\coreapplication_obis1\repository`, select the desired repository, and click Open. Click "Apply and Activate Changes" (not shown here). You then need to restart the Oracle BI Server component, which is discussed in the next slide.

Using FMW Control to Manage OBI Components



Using FMW Control to Manage OBI Components

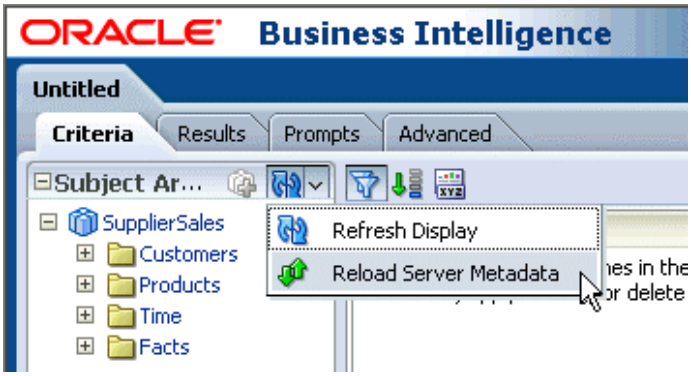
After you make changes to a repository in offline mode and publish the repository, you must restart the Oracle BI Server component before those changes can take effect.

To use FMW Control to start, stop, and restart all Oracle Business Intelligence system components, go to the Business Intelligence Overview page. Use the buttons in the System Shutdown & Startup region to start, stop, or restart all Oracle Business Intelligence system components

To start, stop, or restart individual Oracle Business Intelligence system components, display the Availability tab within the Capacity Management tab, select a process for a selected server, and use the appropriate button to start, stop, or restart individual processes.

Reloading Server Metadata

Click the Reload Server Metadata link to refresh the view of Oracle BI metadata.

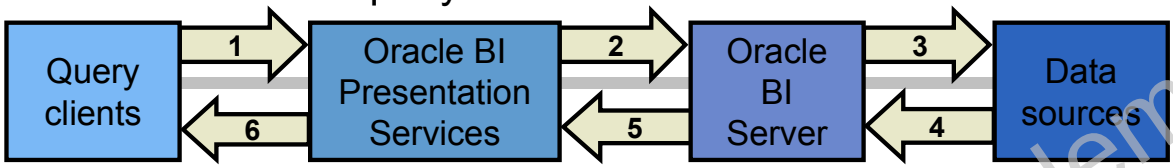


Reloading Server Metadata

When changes have been made to saved content or to the Oracle BI Server metadata in online mode, you must refresh the display to access the most current information. To refresh the view of the Oracle BI Server metadata for subject areas, click the Reload Server Metadata link. To refresh the information for saved requests, filters, briefing books, and dashboard content, click the Refresh Display link. You must have the appropriate permissions to perform these tasks.

Sample Analysis Processing

1. A user views a dashboard or submits an analysis.
2. Oracle BI Presentation Services makes a request to Oracle BI Server to retrieve the requested data.
3. Using the repository file, Oracle BI Server optimizes functions to request the data from the data sources.
4. Oracle BI Server receives the data from the data sources and processes as necessary.
5. Oracle BI Server passes the data to Oracle BI Presentation Services.
6. Oracle BI Presentation Services formats the data and sends it to the query client.



Sample Analysis Processing

Shown in the slide is a simplified example of how an Oracle BI analysis is processed. A user accesses a dashboard or requests results for an analysis. The analysis request is received by Oracle BI Presentation Services, which routes the request to Oracle BI Server. Oracle BI Server uses the repository to determine the best way to access the requested data. Then it sends the SQL or other requests to the sources and combines the results or provides further processing. Oracle BI Server then sends the data back to Oracle BI Presentation Services, which formats the data as appropriate and sends it to the query client for display.

Summary

In this lesson, you should have learned how to:

- Identify and describe the major components of the Oracle BI architecture
- Identify and describe the three layers of an Oracle BI repository and their relationship to each other
- Use the Oracle BI Administration Tool to explore repository objects
- Publish a repository
- Start and stop Oracle BI components

Practice 2-1 Overview: Exploring an Oracle BI Repository

This practice covers the following topics:

- Exploring the Physical layer of a repository
- Exploring the Business Model and Mapping layer of a repository
- Exploring the Presentation layer of a repository
- Loading a repository into memory
- Submitting an analysis

Practice 2-1 Overview: Exploring an Oracle BI Repository

This practice familiarizes you with the Administration Tool. It also helps you understand the three layers of an Oracle BI repository and the relationships among them.

3

Building the Physical Layer of a Repository

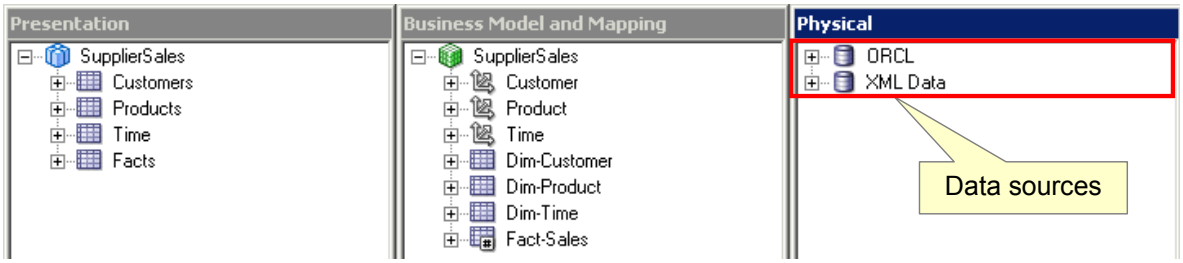
Objectives

After completing this lesson, you should be able to:

- Identify and describe the objects in the Physical layer of a repository
- Create the Physical layer of a repository

Physical Layer

- Contains objects representing the physical data sources to which Oracle BI Server submits queries
- May contain multiple data sources
- Is typically the first layer built in a repository



Physical Layer

The Physical layer defines the data sources to which Oracle BI Server submits queries. It also defines the relationships between physical databases and other data sources that are used to process multiple data source queries.

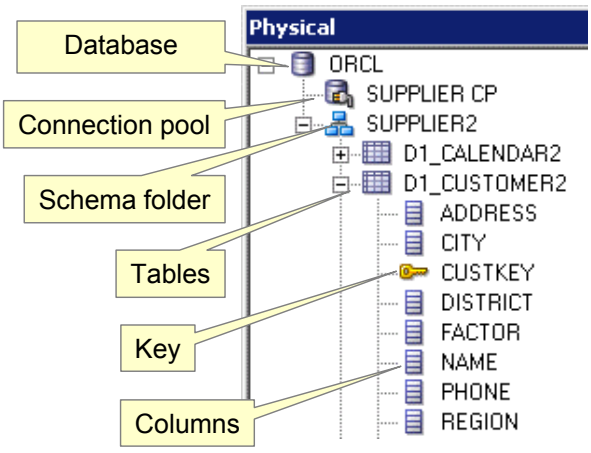
The most common way to populate the Physical layer is by importing metadata from databases and other data sources. The data sources can be of the same or different varieties. You can import schemas or portions of schemas from existing data sources. Additionally, you can create the Physical layer manually; however, the recommended approach is to use the import wizard.

When you import metadata, many of the properties of the data sources are configured automatically based on the information gathered during the import process. After import, you can also define other attributes of the physical data source, such as join relationships, that might not exist in the data source metadata. There can be one or more data sources in the Physical layer, including databases, spreadsheets, and Extensible Markup Language (XML) documents. In the example in the slide, there are two data sources in the Physical layer: `ORCL` and `XML Data`.

Physical Layer Objects

Are objects in the Physical layer, such as connection pools, folders, tables, columns, and keys

- Expand a database object to display the objects that it contains:



Physical Layer Objects

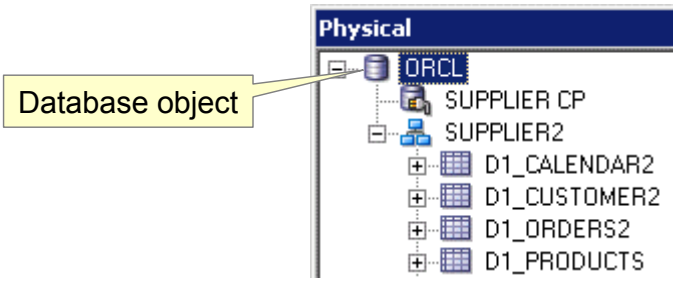
The database object is the highest object in the Physical layer. Each database object in the Physical layer contains a connection pool (`SUPPLIER CP` in this example). A connection pool contains information about the connection between Oracle BI Server and the data source.

You can organize the Physical layer into folders. The folders shown here are created by default when the schema is first imported. Additional folders are optional. A schema folder, `SUPPLIER2` in this example, contains tables, columns, and keys for a physical schema. The table, column, and key objects provide the metadata necessary for Oracle BI Server to access the actual physical sources with SQL requests.

Each object in the Physical layer is associated with a set of properties. To view an object's properties, double-click its icon to open the properties dialog box. Each of the objects shown here is discussed in more detail in the following slides.

Database Object

- Is the highest-level object in the Physical layer
- Defines the data source to which Oracle BI Server submits queries



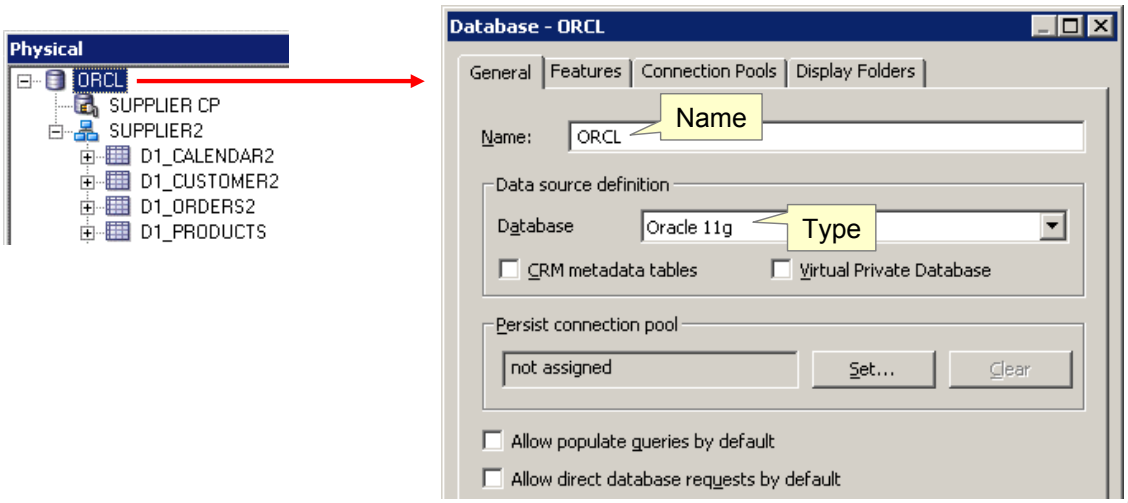
Database Object

The database object is the highest-level object in the Physical layer. It defines the data source to which Oracle BI Server submits queries. Importing a schema automatically creates a database object for the schema in the Physical layer.

In this example the database object is named `ORCL`, which is the name inherited from the `tnsnames.ora` connection information during import.

Database Object: General Properties

Click the General tab to view and set general properties for a database object.



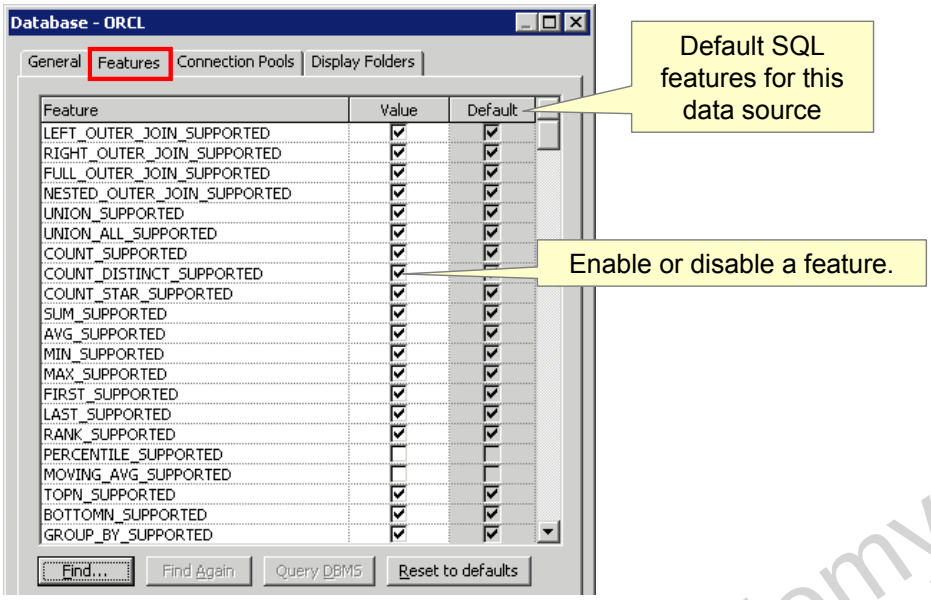
Database Object: General Properties

Importing a schema automatically creates a database object for it, but sometimes you need to set up the database object properties manually. Double-click the database object, or right-click and select Properties to open the Database Object Properties dialog box.

- If you import the physical schema into the Physical layer, the database name and type are usually assigned automatically. However, if the server cannot determine the database type, an approximate database type is assigned to the database object. If necessary, replace the database type with the closest matching entry from the database drop-down menu.
- "Persist connection pool" is a database property that is typically used to support queries in Marketing applications. For more information, see the Oracle BI Enterprise Edition documentation.
- When the "Allow populate queries by default" check box is selected, you can 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 by using the Security Manager. You learn more about the Security Manager in the lesson titled "Security."
- When the "Allow direct database requests by default" check box is selected, you can execute physical queries directly against the database from the Oracle BI user interface. 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.

Database Object: Features

Click the Features tab to view and set the SQL features that Oracle BI Server uses with this data source.



Database Object: Features

When you import the schema or specify a database type on the General tab of the Database dialog box, the Features table is automatically populated with default values appropriate for the database type. These are the SQL features that Oracle BI Server uses with this data source.

You can customize the default query features for a database. For example, if a data source supports left outer join queries, but you want to prohibit the server from sending such queries to a particular database, you can change this default setting in the Feature table. But be careful when modifying the Features table. If you enable SQL features that the database does not support, your query may return errors and unexpected results. If you disable supported SQL features, the server may issue less-efficient SQL to the database.

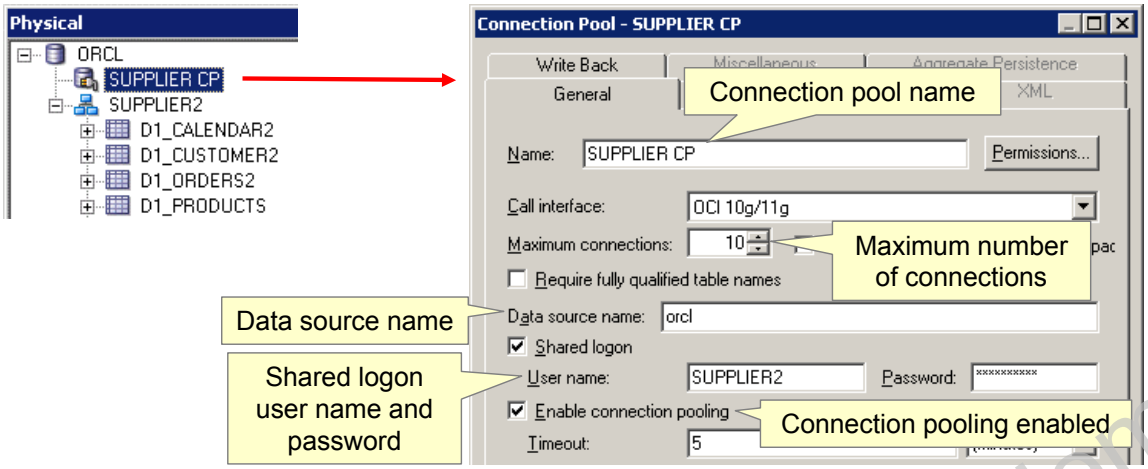
- The Default check box identifies the default SQL features for this data source.
- The Value check box allows you to enable or disable a query type. It is strongly recommended that you do not disable the default SQL features.
- The Query DBMS button is used only if you are installing and querying a database that has no Features table. It allows you to query a database for the SQL features that it supports. Be careful when using Query DBMS. The query results are not always an accurate reflection of the SQL features that are actually supported by the data source. You should use this functionality only with the assistance of Oracle Technical Support.

Database Object: Features (continued)

- The “Reset to defaults” button restores the default features values.
- The Find button enables you to enter a string to help you locate a feature in the list.

Connection Pool

- Defines how Oracle BI Server connects to a data source
- Specifies the ODBC or native data source name
- Allows multiple users to share a pool of data source connections



Connection Pool

For each data source, there is at least one corresponding connection pool. The connection pool contains information about the connection between Oracle BI Server and a data source. Typically, the database object and connection pool are created automatically when you import the physical schema. It is strongly recommended that you use Oracle Call Interface (OCI) for connecting to an Oracle data source.

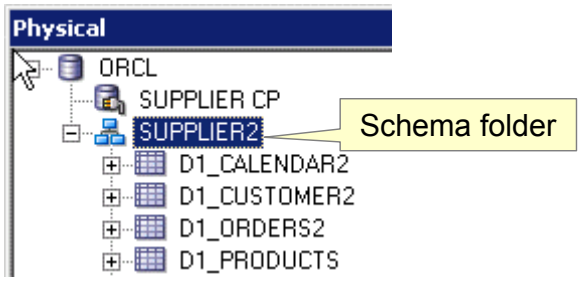
The connection pool contains data source name (DSN) information used to connect to a data source, the number of connections allowed, timeout information, and other connectivity-related administrative details. Connection pools allow multiple concurrent data source requests (queries) to share a single database connection, reducing the overhead of connecting to a database. You can create multiple connection pools to improve the performance for groups of users.

Selecting the "Enable connection pooling" check box allows multiple concurrent query requests to share a single database connection. This reduces the overhead of connecting to a database because it does not open and close a new connection for every query. If you do not select this option, each query sent to the database opens a new connection.

Schema Folder

Is an optional display folder that contains tables and columns for a physical schema

- To create a schema folder, right-click a database object and select New Object > Physical Schema:



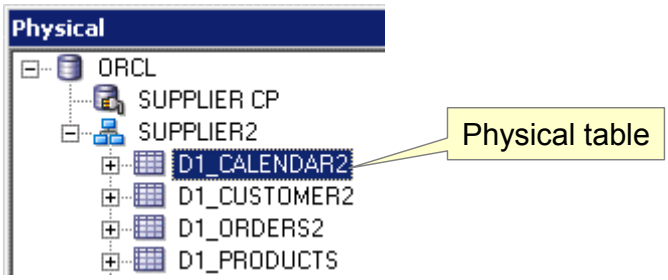
Schema Folder

The schema folder contains tables and columns for a physical schema. Schema folder objects are optional in the Physical layer of the Administration Tool. You must create a database object before you create a schema object. A schema folder is created by default when you first import from a data source.

To create a new schema object, right-click the database object and select New Object > Physical Schema.

Physical Table

- Is an object that corresponds to a table in a physical data source
- Is typically imported from a database or other data source
- Provides the metadata necessary for Oracle BI Server to access the tables with SQL requests



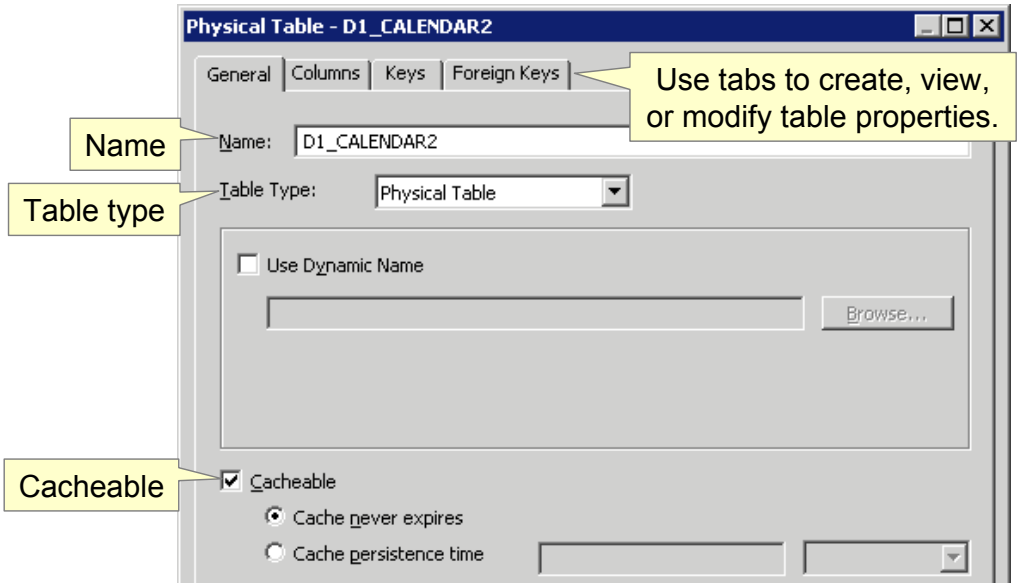
Physical Table

A physical table is an object in the Physical layer of the Administration Tool that corresponds to a table in a physical database. Physical tables are usually imported from a database or another data source, and they provide the metadata necessary for Oracle BI Server to access the tables with SQL requests.

When data source definitions are imported, no actual data is moved. Data remains in the physical data source.

Physical Table Properties

Double-click a physical table object to view its properties:



Physical Table Properties

Click the General tab to view and set general properties for a physical table object. By default, the name corresponds to the table name that is defined during import, but you can rename it.

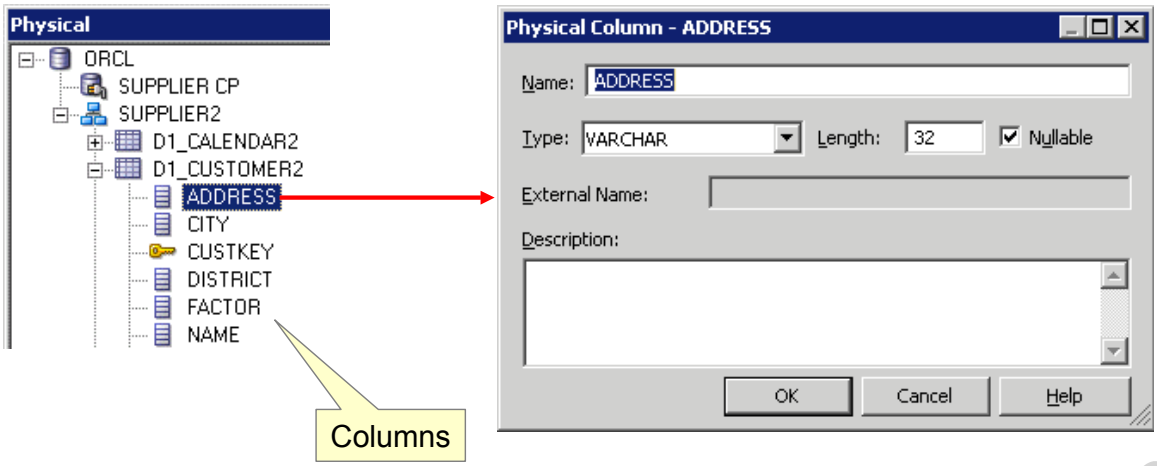
- The Table Type drop-down list allows you to specify the physical table object type. Physical Table is the default. You can also define the physical table as a stored procedure or a `SELECT` statement. When you select either of these options, a text pane below the Table Type drop-down list becomes active, allowing you to enter the stored procedure or the `SELECT` statement. Requests against this table then execute the stored procedure or the `SELECT` statement.
- You can use a dynamic name to specify the name of a table object. You must define a session variable before you can set a dynamic name. You learn more about variables in the lesson titled “Using Repository Variables.”
- Select the Cacheable check box to include the table in the Oracle BI Server query cache. This is selected by default. Select “Cache never expires” or set “Cache persistence time” to define how long table entries should persist in the cache. You learn more about caching in the lesson titled “Cache Management.”

Physical Table Properties (continued)

- Database hints are instructions placed within a SQL statement that tell the database query optimizer what the most efficient way to execute the statement is. 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.
- The Columns, Keys, and Foreign Keys tabs allow you to view, create new, and edit existing columns, keys, and foreign keys that are associated with the table.

Physical Column

Is an object that corresponds to a column in a physical database



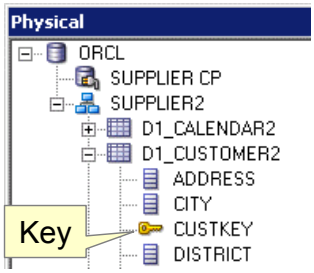
Physical Column

A physical column is an object in the Physical layer of the Administration Tool that corresponds to a column in a physical database. Each physical table object in the Physical layer has one or more physical column objects. The column properties are set automatically when the column is imported. Double-click a column to view or modify its properties.

Key Column

Defines relationships between tables

- Primary key
  - Uniquely identifies a single row of data
  - Consists of a column or set of columns
  - Is identified by a key icon
- Foreign key
  - Refers to the primary key columns in another table
  - Is composed of a column or set of columns



Key Column

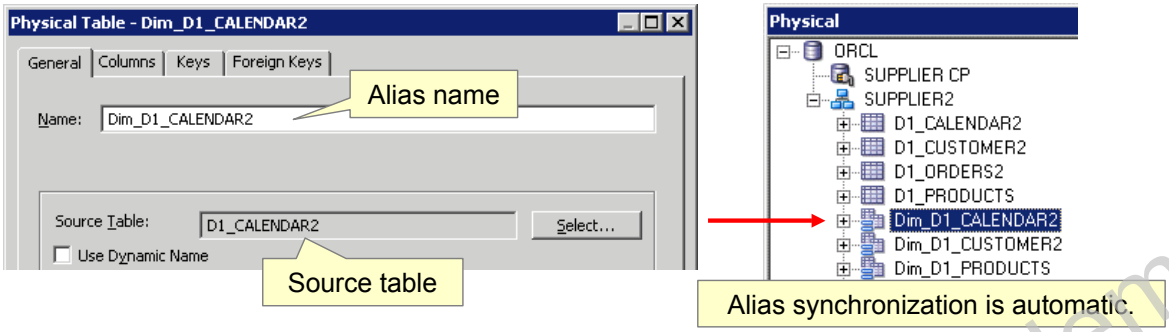
A primary key and foreign key relationship defines a one-to-many relationship between two tables. A foreign key is a column or a set of columns in one table that references the primary key columns in another table. The primary key is defined as a column or set of columns where each value is unique and identifies a single row of the table. Keys and joins help Oracle BI Server determine the fact–dimension relationships between tables.

The Physical layer typically uses foreign key joins to define relationships, whereas the Business Model and Mapping layer uses logical joins to define relationships. You learn more about logical joins in the lesson titled “Building the Business Model and Mapping Layer of a Repository.” Typically, keys are defined for dimension tables only and not for fact tables.

Physical Table: Alias

Is a virtual physical table object that points to a physical table object

1. Right-click a physical table and select New Object > Alias.
2. Provide a name for the alias table.
   The alias table appears with an alias icon in the Physical layer.



Physical Table: Alias

An alias table is a virtual physical table object that points to a physical table object. An alias table in the Physical layer of the repository is just like any alias used in standard SQL notation. That is, you want to reference the same table twice in a given SQL statement, but you use a different name to avoid circularity in the references. A common use for aliases is role-playing dimensions, where a single dimension simultaneously appears several times in the same fact table. For example, an order date and a shipping date in a fact table may both point to the same column in a physical dimension table, but each role is presented as a separately labeled alias.

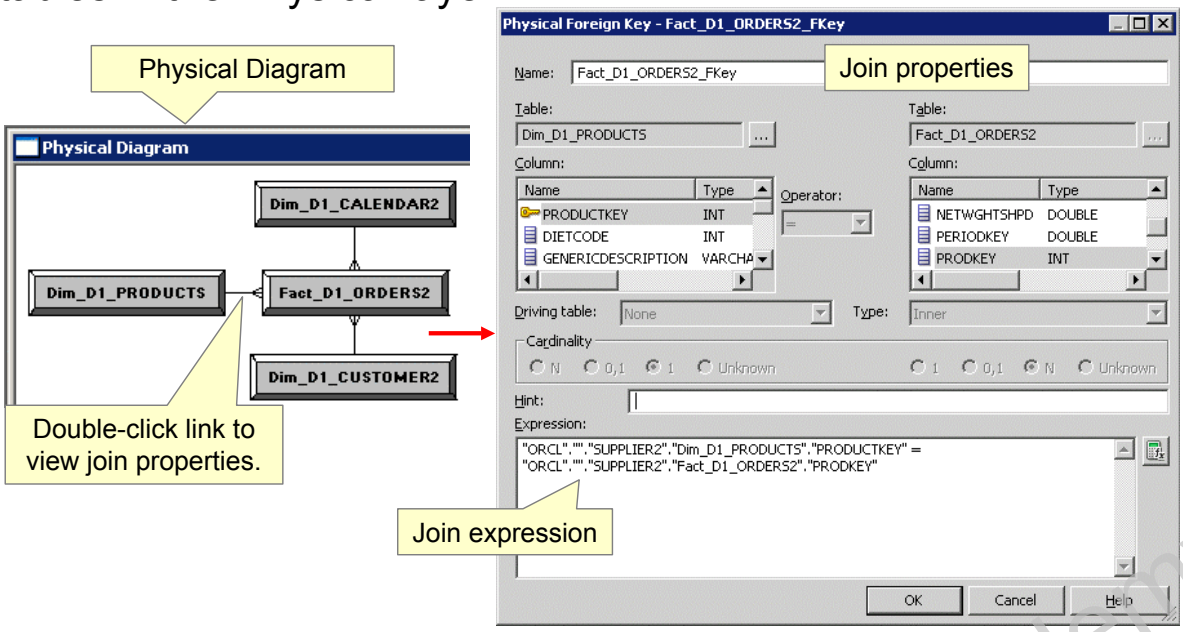
Alias synchronization is the act of ensuring that the source table and related alias tables have the same column definitions. An alias table always inherits all the column definitions from the source table, and synchronization happens automatically. Thus, columns in an alias table cannot be modified. All columns opened from an alias table become read-only. Creating a source column creates the same column on all its alias tables. Deleting a source column automatically deletes corresponding alias columns. Modification of a source column forces the same changes to be reflected in the alias columns.

To create an alias:

1. Right-click a physical table and select New Object > Alias.
2. Give the alias table a name. The Table Properties dialog box displays the source table. The alias table appears with an alias icon in the Physical layer.

Joins

Represent the primary key–foreign key relationships between tables in the Physical layer



Joins

Joins represent the relationships between tables in the Physical layer. Joins that already exist in the physical data sources are imported automatically into the Physical layer but can be modified. When you import keys in a physical schema, the primary key–foreign key joins are automatically defined when you select keys and foreign keys in the Import Metadata Wizard. You learn more about the Import Metadata Wizard later in this lesson.

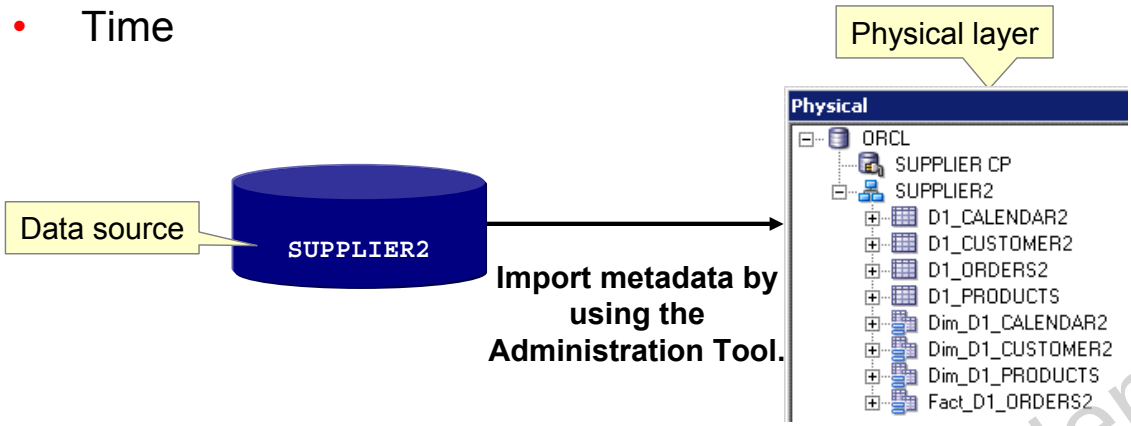
All other joins within each database have to be defined explicitly. You need to define joins that express relationships between tables in the Physical layer of the repository. You can use the Physical Diagram feature to create joins between physical table objects. Note that complex joins can be used in this layer to express the relationships that do not involve a primary key–foreign key relationship.

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 do not specify expressions. You learn more about logical joins in the lesson titled “Building the Business Model and Mapping Layer of a Repository.”

ABC Scenario

The SUPPLIER2 schema in an Oracle database contains tables with the following ABC data:

- Invoice
- Customer
- Product
- Time



ABC Scenario

Throughout this course, you build a repository based on the business requirements of ABC, a fictitious company that sells restaurant supplies. In the first practice for this lesson, you read a document that explains in detail the ABC business scenario. Because you have not yet read this document, this slide briefly introduces the ABC business requirements.

ABC's data resides in the SUPPLIER2 schema in an Oracle relational database on your student machine. The schema contains invoice fact data and customer, product, and time dimension data. The first tasks are to use the Administration Tool to import metadata into the Physical layer of the repository, create alias tables, and build physical joins. The remaining slides in this lesson describe the steps for completing these tasks.

Implementation Steps

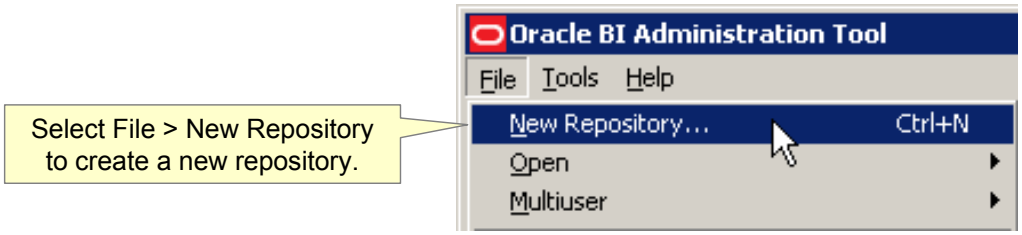
1. Create a new repository file.
2. Provide repository information.
3. Select the data source.
4. Select metadata types for import.
5. Select metadata objects for import.
6. Verify and edit connection pool properties.
7. Verify objects for import
8. Verify import in the Physical layer.
9. Verify connectivity
10. Create alias tables.
11. Define keys and joins.

Implementation Steps

This is an overview of the necessary steps for building the Physical layer. Subsequent slides cover each step in detail.

Create a New Repository File

Use the Administration Tool to create a new repository file:



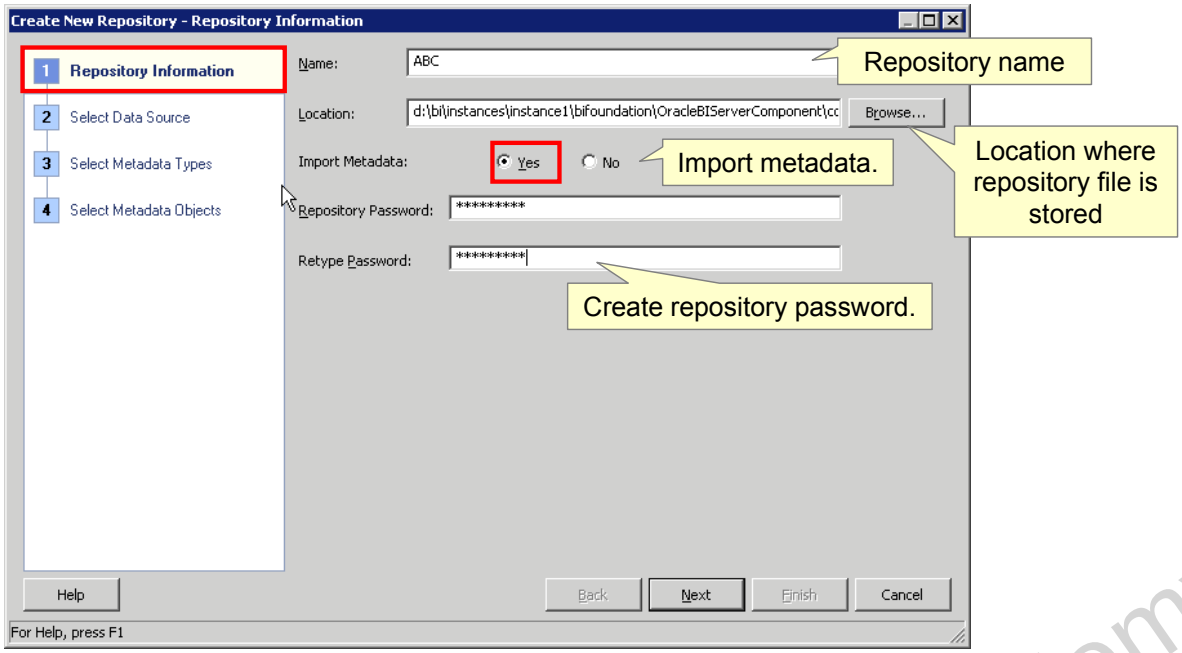
Create a New Repository File

You use the Create New Repository Wizard in the Administration Tool to create a new repository and import tables into the Physical layer of the repository.

Select File > New Repository to open the Create New Repository Wizard, which guides you through the steps to create a new repository and import metadata.

Provide Repository Information

Provide general information about the repository:



Provide Repository Information

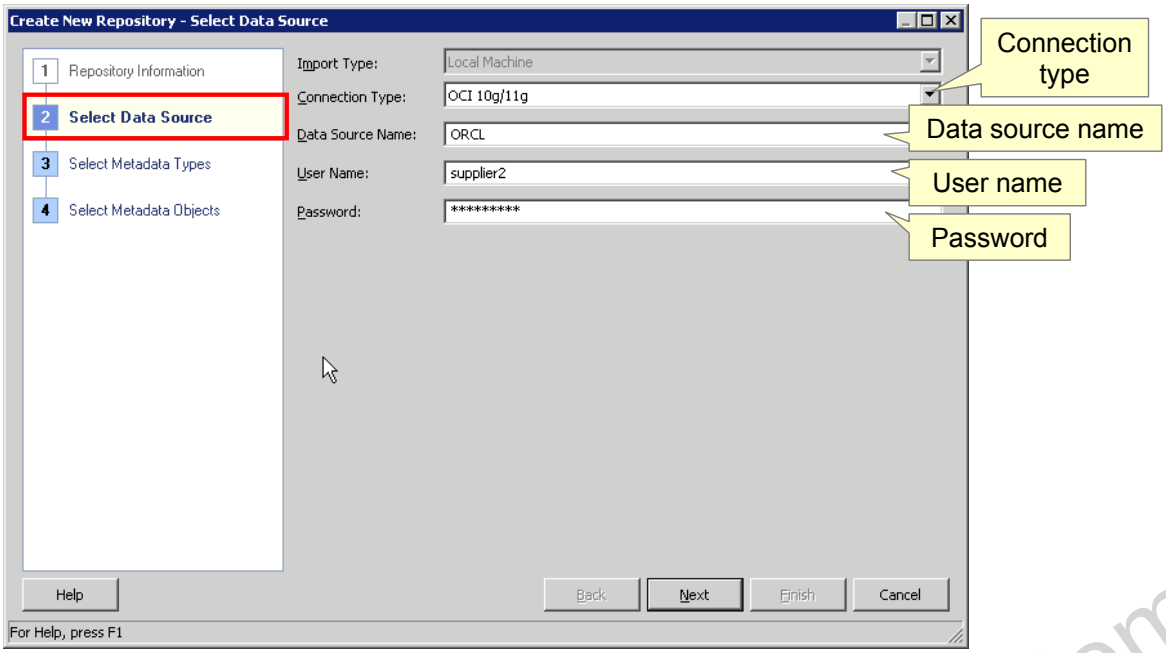
On the first screen of the Create New Repository Wizard, enter general information about the repository. Provide a name for the repository and the location where the repository is to be stored. By default, the location field points to `ORACLE_INSTANCE\bifoundation\OracleBIServerComponent\coreapplication_obis1\repository`.

When creating a new repository, you typically accept the default selection `Yes` for Import Metadata. When `Yes` is selected, the Create New Repository Wizard continues with screens for importing metadata. When `No` is selected, an empty repository is saved to the selected location.

Enter a repository password and confirm the password. Remember that you are *creating* a repository password in this step rather than entering an existing value. The repository password is used to encrypt and decrypt the content of the repository and is used every time the repository is opened or uploaded. It must be longer than five characters and cannot be empty.

Select the Data Source

Provide connection information to select the data source:



Select the Data Source

On the Select Data Source screen, provide the information for the data source from which you are importing the metadata.

Begin by selecting a connection type from the Connection Type drop-down list. The screen displays connection fields based on the connection type you select. Connection types include Oracle Call Interface (OCI), XML, ODBC, Essbase, XMLA, and so forth.

If you use an ODBC connection type, you must have already used ODBC Administrator to define the system data source name for the data source you want to import. This example illustrates using the OCI native database gateway to import a physical schema.

Enter the data source name. In this example, the name `ORCL` corresponds to an entry in the `tnsnames.ora` file.

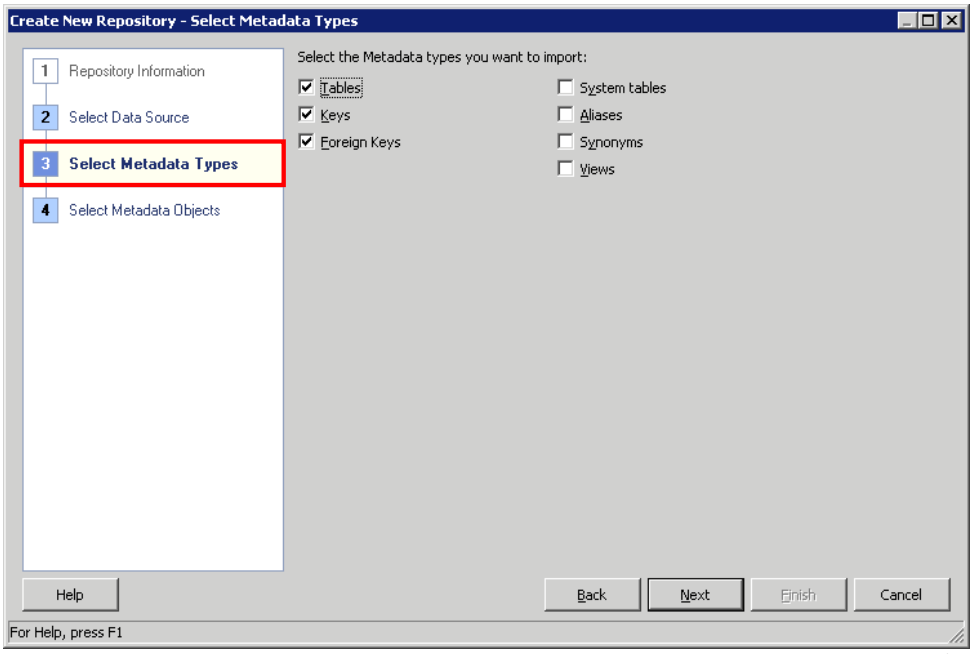
Note: You must put a copy of `tnsnames.ora` in `ORACLE_HOME\network\admin`. This is a new requirement for Oracle BI 11*g*.

Enter a valid user name and password to connect to the data source.

The most common way to populate the Physical layer is by importing from databases and other data sources. In fact, it is recommended that you always import the physical schema rather than build the metadata manually in the Physical layer.

Select Metadata Types for Import

Provide information about which metadata types to import:



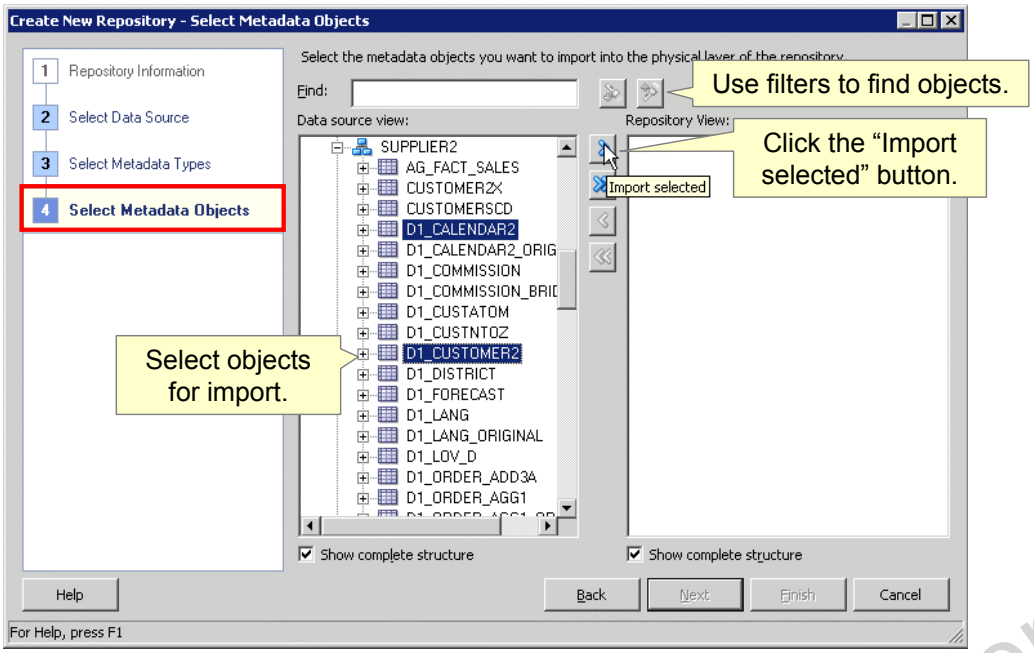
Select Metadata Types for Import

On the Select Metadata Types screen, the check boxes enable you to select the information to import. You can import tables, keys, foreign keys, database views, aliases, synonyms, and system tables.

As a general rule, import only those objects that are needed to support your business model.

Select Metadata Objects for Import

Provide information about which metadata objects to import:



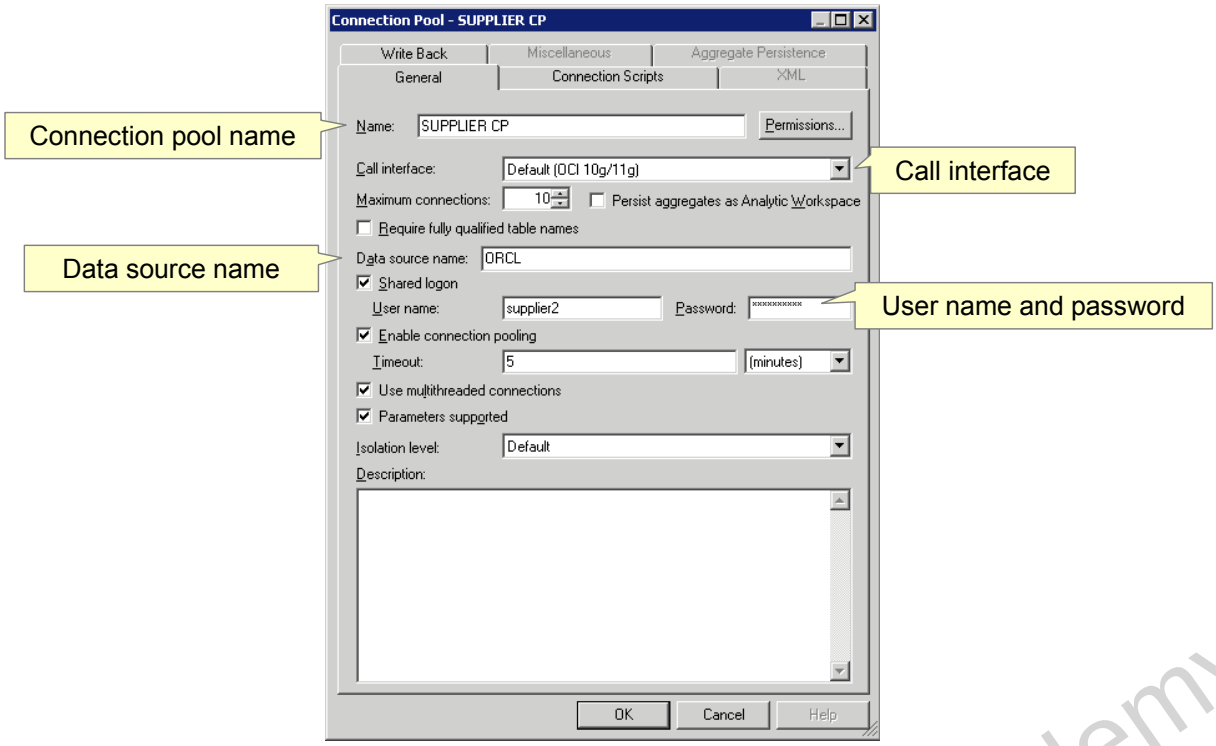
Select Metadata Objects for Import

At this point, you can import schemas or portions of schemas from existing data sources. When you import the physical tables, it is good practice to limit the import to only those tables that contain data that you are likely to use in the business models you create. If you do import extra objects at this point, you can delete them later if you determine that they do not support your business model. You can also import additional table objects later.

You can use the Find field to limit the number of tables that appear in the import list. This makes it easier to locate and select the tables that you want to import.

After selecting the desired objects in the data source view, click the "Import selected" button to move the objects to the Repository View pane.

Verify and Edit Connection Pool Properties



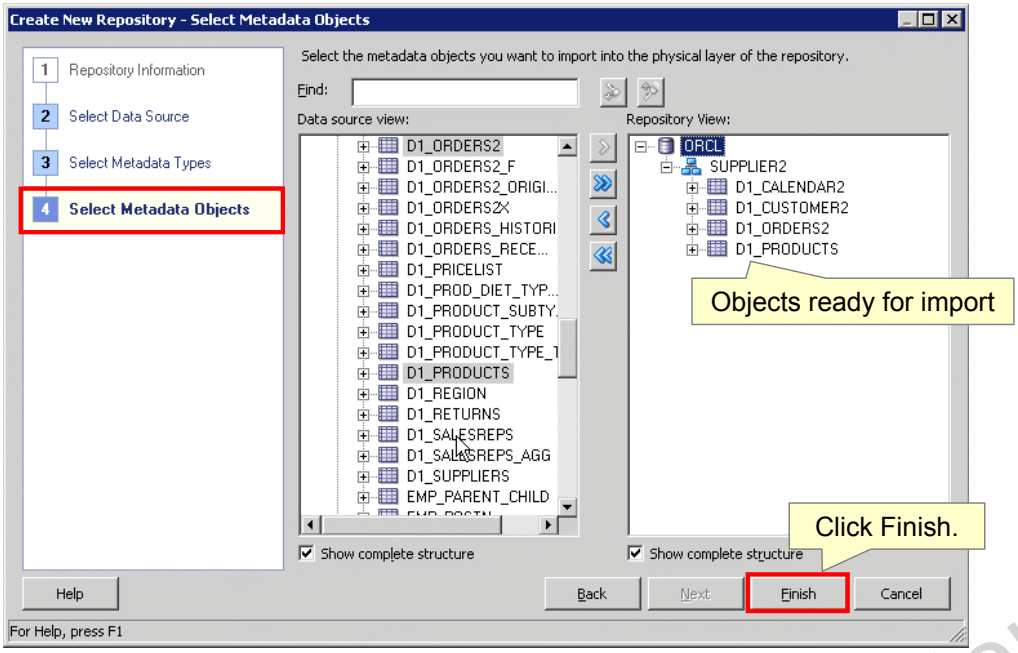
Verify and Edit Connection Pool Properties

When you click the “Import selected” button, the Connection Pool dialog box opens. Recall that the connection pool contains information about the connection between Oracle BI Server and that data source (for example, the connection pool name, the data source name that connects to the data source, user name and password for the data source, maximum number of connections, and so forth).

Connection Pool parameters are configured automatically as part of the import process, so in most cases you can accept the defaults and click OK. In this example, the connection pool name has been changed from `Connection Pool` to `SUPPLIER CP`.

Verify Objects for Import

Verify which objects will be imported into the Physical layer:



Verify Objects for Import

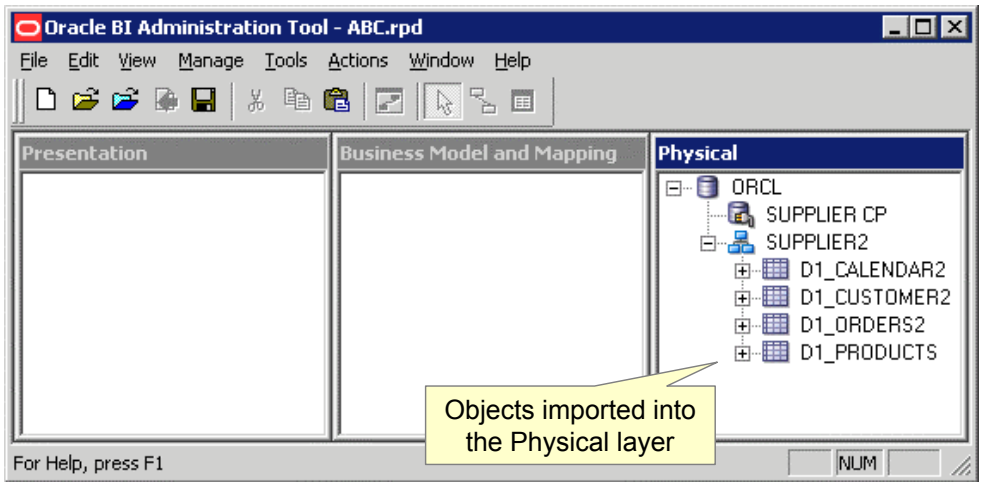
After closing the Connection Pool dialog box, you can see which objects will be imported into the Physical layer of the repository. At any time in the import process, you can click the Back button to return to an earlier screen.

You can choose to show the complete structure. You can also show only those objects in the data source view that have not been imported, or only those objects in the Repository view that are being imported.

Click Finish when you are ready to import into the Physical layer of the repository.

Verify Import in the Physical Layer

Confirm that objects are imported into the Physical layer:

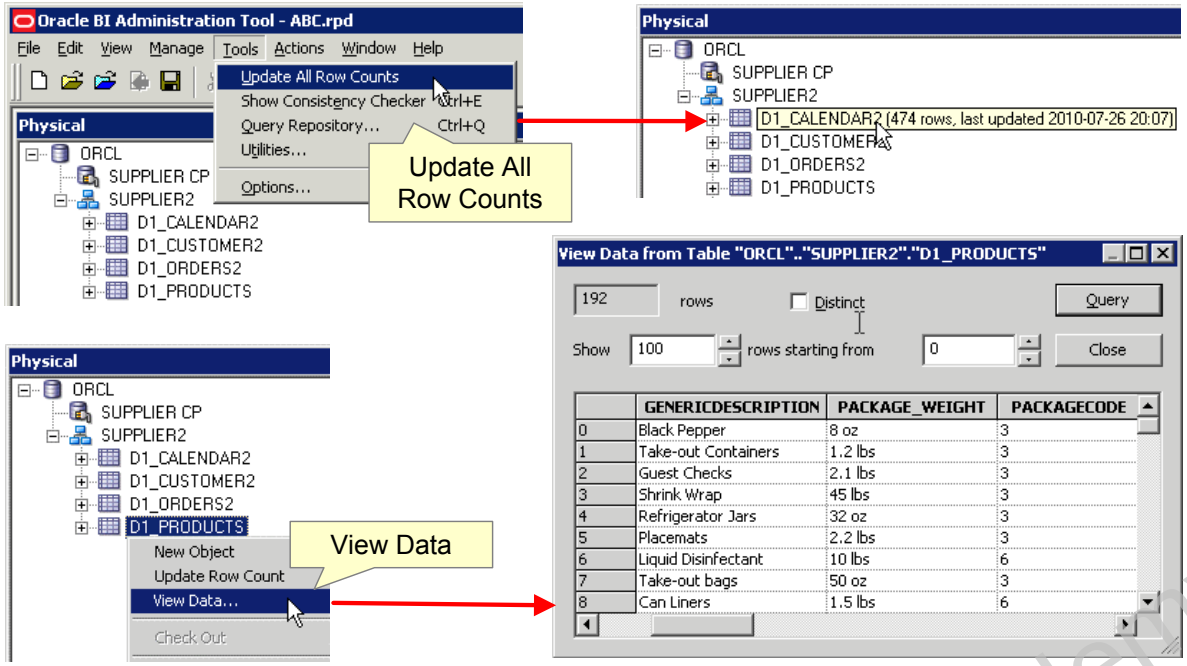


Verify Import in the Physical Layer

After import completes, confirm that the expected schemas, tables, columns, keys, and so forth exist in the Physical layer.

Verify Connectivity

Update row counts and view data to confirm connectivity:



Verify Connectivity

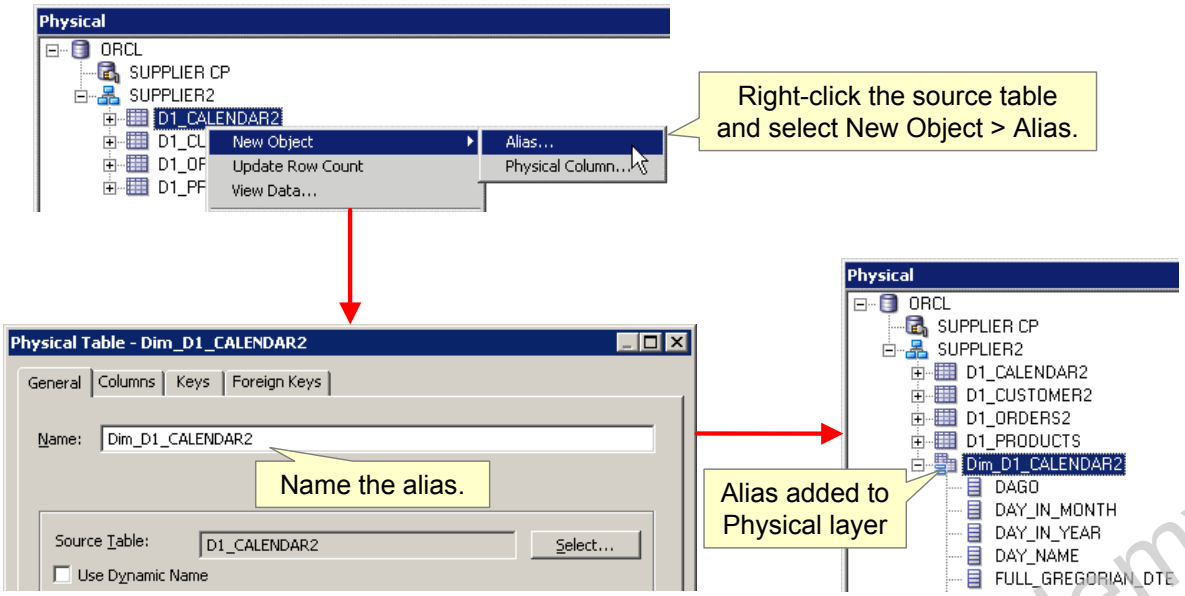
It is good practice to update row counts and view data to ensure that the connection is configured correctly.

Select Tools > Update All Row Counts to update row counts for all rows. Alternatively, you can right-click an object and select Update Row Count.

To view data for an object, right-click the object and select View Data.

Create Alias Tables

Assign aliases to physical tables before mapping them to the business model layer:



Create Alias Tables

Creating alias tables is not a required step. However, it is recommended that you use table aliases frequently in the Physical layer to eliminate extraneous joins and to include best-practice naming conventions for physical table names.

To create an alias, right-click the desired source table and select New Object > Alias. Provide a name for the alias. The alias is automatically added to the Physical layer and shares all of the attributes of its physical source table.

Define Keys and Joins

The Administration Tool enables you to define physical keys and joins that were not imported automatically.

- Define keys by using the Physical Table properties dialog box.
- Define joins and keys by using the Physical Diagram.
- Define joins and keys by using the Joins Manager.

Define Keys and Joins

You must define valid keys and joins in the Physical layer of the repository. The Administration Tool allows you to define keys and joins by using several different methods. If the tables imported from the data source already have primary key–foreign key join relationships, and if the primary keys and foreign keys are imported into the repository during the import process, the join conditions are set up automatically in the Physical layer of the repository.

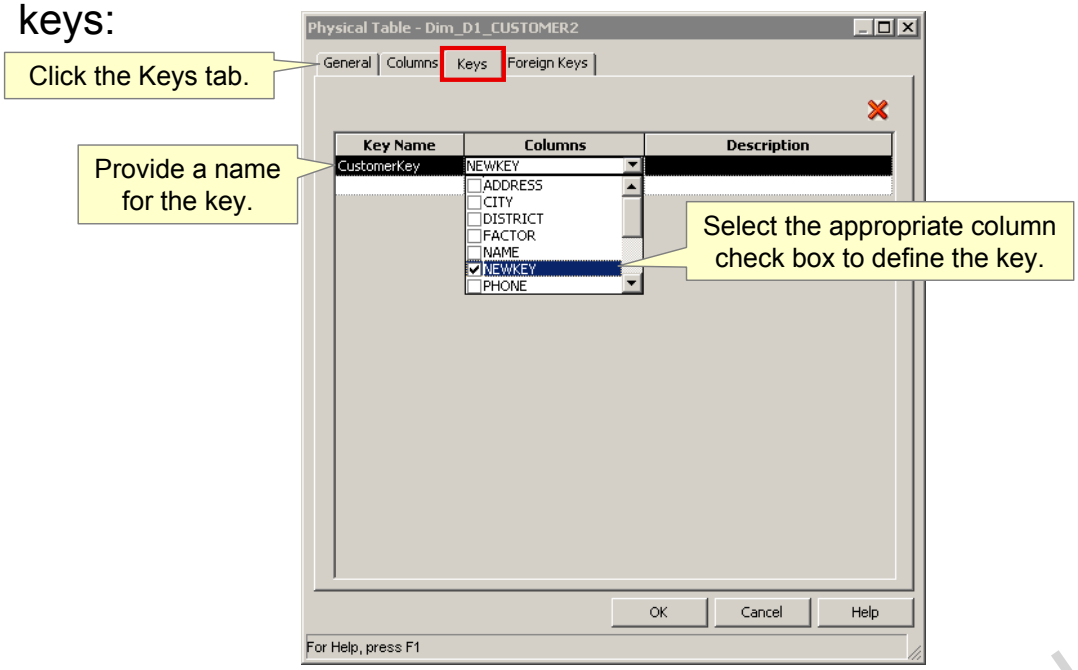
If you need to define keys and joins manually, use any of the following methods:

- Define keys using the Keys tab in the Physical dialog box.
- Define keys while creating joins by using the Physical Diagram.
- Define joins in the Physical Diagram.
- Define joins by using the Joins Manager.

Each method is discussed in detail in the following slides.

Defining Keys by Using the Table Properties Dialog Box

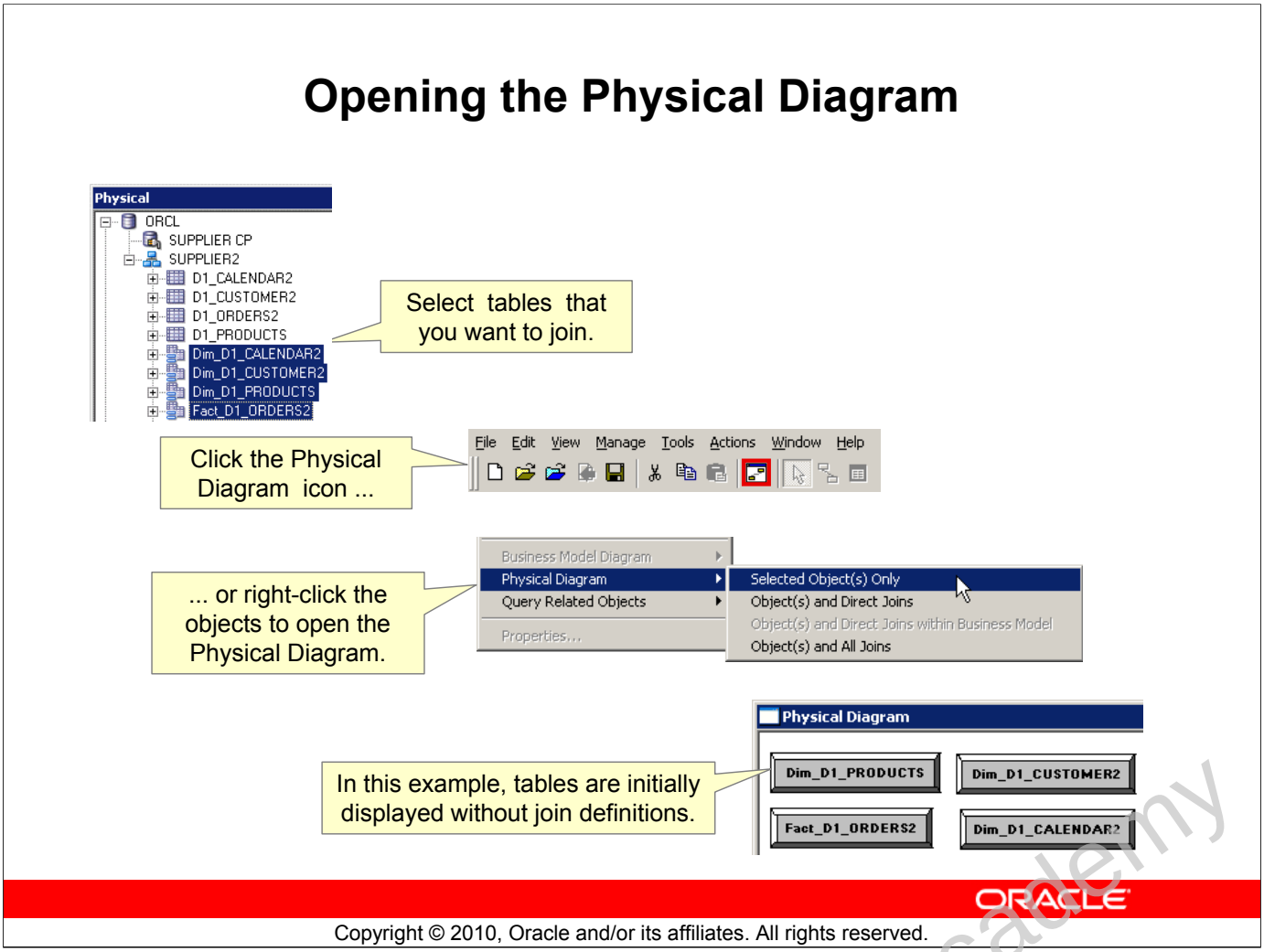
Open the Physical Table properties dialog box to view or define keys:



Defining Keys by Using the Table Properties Dialog Box

Right-click a table object and select Properties, or double-click the table object to open the Physical Table properties dialog box, and then click the Keys tab. Provide a name for the key and select the column or columns that define the key. The primary key is defined as a column or set of columns in which each value is unique and identifies a single row of the table.

You also can view or manually define foreign keys on the Foreign Keys tab (not shown here).



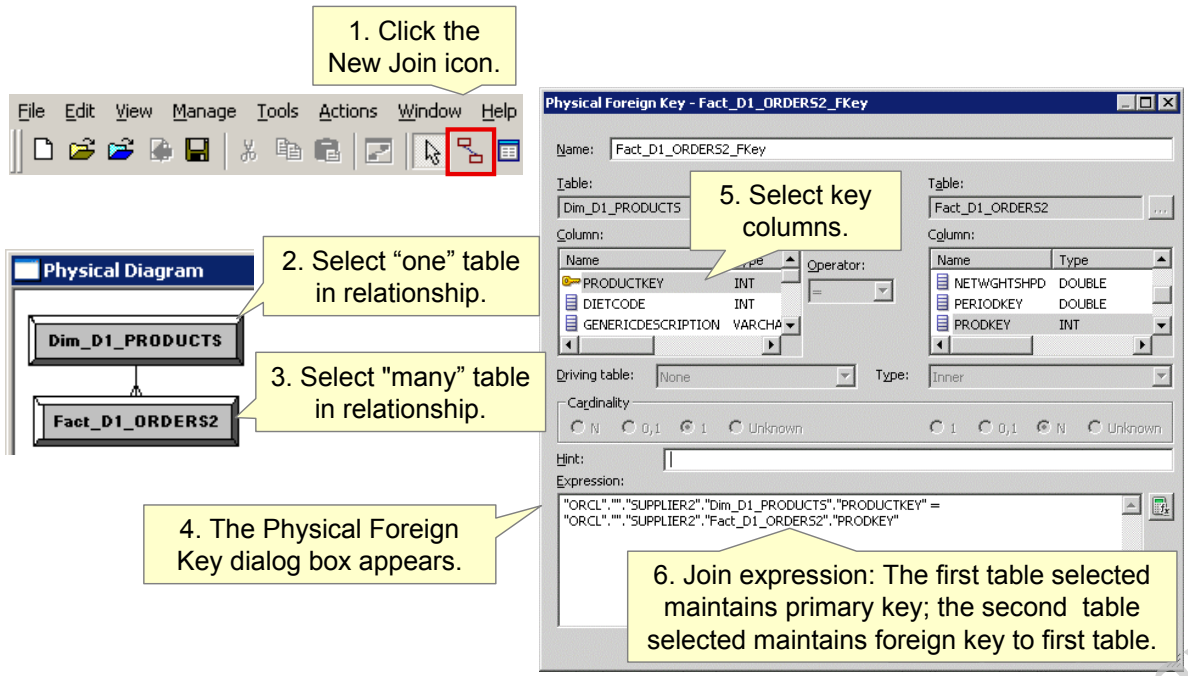
Opening the Physical Diagram

The Physical Diagram graphically shows the physical tables and any defined joins between them. You can use the Physical Diagram to define primary keys, foreign keys, and joins between tables. As discussed earlier, all valid physical joins must be configured in the Physical layer of the Administration Tool.

To open the Physical Diagram window, select one or more tables in the Physical layer and click the Physical Diagram icon on the toolbar, or right-click one of the selected tables and then select one of the Physical Diagram options from the shortcut menu. The options are shown in the slide: Selected Object(s) Only, Object(s) and Direct Joins, and Object(s) and All Joins.

In the ABC scenario, the tables are initially displayed without join definitions. You can then create new joins by using the Join icons on the Physical Diagram toolbar. The steps are shown in the next slide.

Defining Foreign Key Joins



Defining Foreign Key Joins

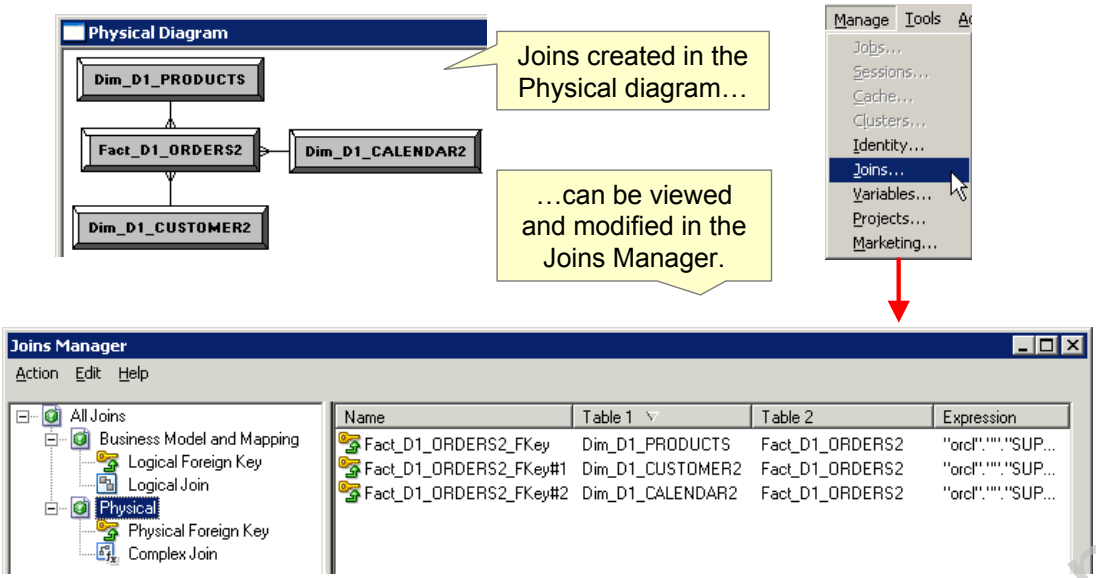
This slide describes the process for building a foreign key join in the Physical Diagram. To define a foreign key join:

1. Click the New Join icon on the Administration Tool toolbar.
2. Move to the Physical Diagram and click the first table in the join (the table representing “one” in the one-to-many join) to select it.
3. Place the cursor over the table to which you want to join (the table representing “many” in the one-to-many join) and click the second table to select it. The order is important. The first table you click contains the primary key. The second table contains the foreign key that points to the primary key in the first table.
4. After you click the second table, the Physical Foreign Key dialog box appears.
5. Join the key column in the first table to a column that is the foreign key in the second table by selecting the appropriate columns. By default, the left side is the “one” side and the right side is the “many.”
6. The SQL join condition appears in the Expression pane of the window.

You can view or modify joins that have already been created by double-clicking a join in the Physical Diagram to open the Physical Foreign Key dialog box.

Using the Joins Manager

The Joins Manager enables you to examine, edit, and delete all physical and logical repository joins:



Using the Joins Manager

Select Manage > Joins to open the Joins Manager. The joins displayed in the right pane vary depending on the leaf that you select in the left pane. You can view all joins in the repository, in a particular business model, in the Business Model and Mapping layer, in the Physical layer, in the Business Model and Mapping layer for a particular business model, and in the Physical layer for a particular business model. Joins are further divided into logical foreign key, logical, physical foreign key, and physical complex. In this example, Physical is selected in the left pane and all existing physical joins are displayed in the right pane.

You can double-click any of the foreign key expressions to open the Physical Foreign Key dialog box and display the join information (not shown here). Alternatively, you can right-click any join in the list and select Properties to open the Physical Foreign Key dialog box. You can view or edit the join properties by using this dialog box.

Note that you can create new joins by using the Joins Manager. However, most users create joins with the Physical Diagram (as you do in this course). 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.

Design Tips for the Physical Layer

- Import metadata from databases and other data sources.
- Eliminate all outer joins in the data warehouse.
- Use table aliases frequently.
- Use a naming standard in the Physical layer.
- Use an opaque view (a Physical layer table that consists of a `SELECT` statement) only if there is no other solution.

Summary

In this lesson, you should have learned how to:

- Identify and describe the objects in the Physical layer of a repository
- Create the Physical layer of a repository

Practice 3-1 Overview: ABC Business Scenario

This practice covers the ABC business scenario.

Practice 3-1 Overview: ABC Business Scenario

In this practice, you read the ABC document to get acquainted with the business scenario for the fictitious company used throughout this course.

Practice 3-2 Overview: Gathering Information to Build an Initial Business Model

This practice covers the following topics:

- Gathering and analyzing the business requirements of the ABC company
- Determining the structure of the initial business model

Practice 3-2 Overview: Gathering Information to Build an Initial Business Model

Before you can begin to build the metadata, you need to gather and analyze the business requirements of the ABC company. In this practice, you use the information provided in the ABC document that you read in the previous practice to determine the structure of the initial business model.

Practice 3-3 Overview: Creating a Repository and Importing a Data Source

This practice covers the following topics:

- Creating a new repository
- Importing tables into the Physical layer of the repository

Practice 3-3 Overview: Creating a Repository and Importing a Data Source

In the previous practice, you checked the connection to ABC's `SUPPLIER2` database schema by testing your connectivity using the ODBC Client Utility. Now you are ready to create a new repository and import tables from the `SUPPLIER2` schema into the Physical layer of the repository by using the Administration Tool.

Practice 3-4 Overview: Creating Alias Tables

This practice covers assigning aliases to physical tables before mapping them to the business model.

Practice 3-4 Overview: Creating Alias Tables

You create aliases for the metadata objects that you imported into the Physical layer of the repository. It is recommended that you use table aliases frequently in the Physical layer to eliminate extraneous joins and to include best-practice naming conventions for physical table names.

Practice 3-5 Overview: Defining Keys and Joins

This practice covers using the Administration Tool to define the primary keys, foreign keys, and joins in the Physical layer of the repository.

Practice 3-5 Overview: Defining Keys and Joins

In the previous practices, you created a new repository and imported the initial tables from the `SUPPLIER2` schema into the Physical layer of the repository. Now you must define keys and joins that exist on the physical database in the Physical Layer of the repository. If the imported database joined over primary key–foreign key relationships and the primary keys and foreign keys were imported into the repository, the join conditions would be set up automatically. But that is not always what you want, because foreign key relationships are set in a database for only one purpose (referential integrity), which may not correspond to the purpose of the Administration Tool (knowing which joins to include in SQL).

In this database, primary keys, foreign keys, and joins have not been defined. Therefore, you need to define the keys and join conditions manually by using an object's Properties dialog box and the Physical Diagram feature of the Administration Tool.

4

Building the Business Model and Mapping Layer of a Repository

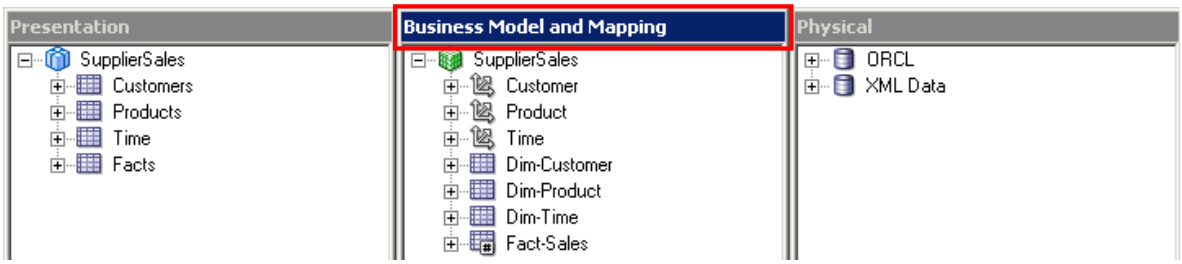
Objectives

After completing this lesson, you should be able to:

- Identify the objects in the Business Model and Mapping (BMM) layer of a repository
- Build a business model
- Create measures in a business model

Business Model and Mapping (BMM) Layer

The BMM layer is where physical schemas are simplified and reorganized to form the basis of the users' view of the data.



Business Model and Mapping Layer

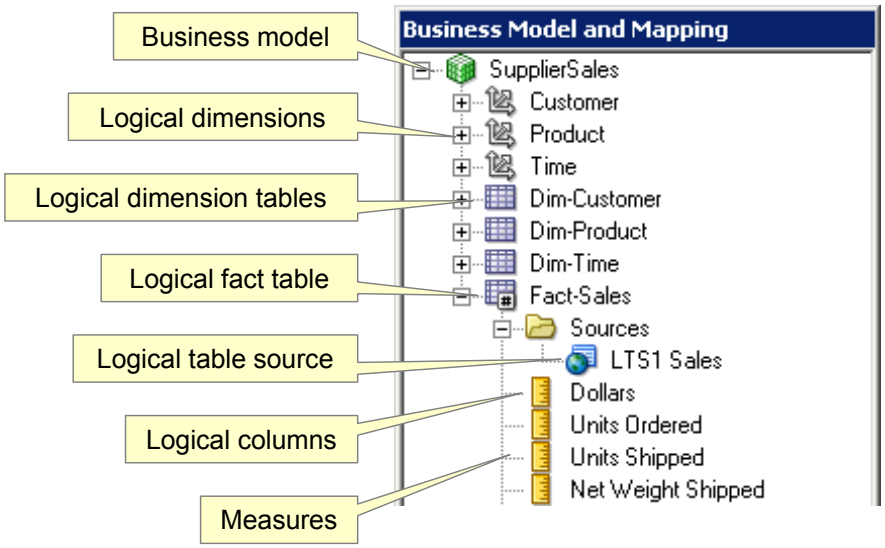
The Business Model and Mapping (BMM) layer of the Oracle BI repository defines the business—or logical—model of the data and specifies the mapping between the business model and the Physical layer schemas. Business models are always dimensional, unlike objects in the Physical layer, which reflect the organization of the data sources.

The BMM layer can contain one or more business models. Each business model contains logical tables, columns, and joins. Although similar terminology is used for logical table and physical table objects (such as the concept of *keys*) logical tables and joins in the BMM 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 BMM layer are typically created automatically when you drag objects from the Physical layer to a particular business model. After these objects have been created, you can perform tasks such as creating additional logical joins, performing calculations and transformations on columns, and adding and removing keys from dimension and fact tables.

Objects in the Business Model and Mapping Layer

The Business Model and Mapping layer contains business model objects:



Objects in the Business Model and Mapping Layer

The BMM layer contains business model objects. There can be multiple business models in this layer, although there is only one (`SupplierSales`) in this example. The business model object is the highest-level object, and in this example it contains the logical tables `Dim-Customer`, `Dim-Time`, `Dim-Product`, and `Fact-Sales`. Logical tables contain logical columns.

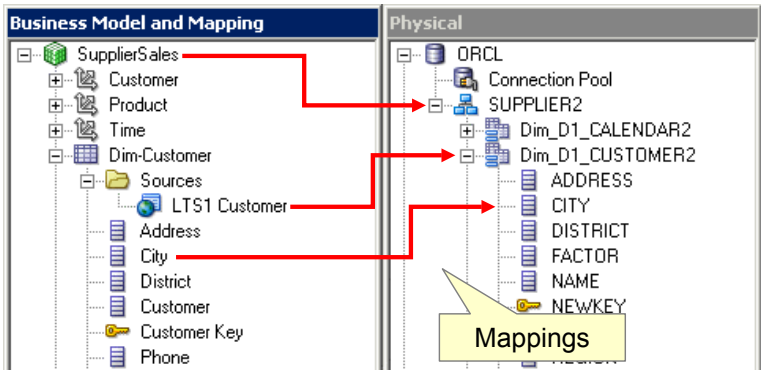
Logical tables have a `Sources` folder that contains one or more logical table sources. These logical table sources define the mappings between the logical tables in the business model and the physical tables in the Physical layer. In this example, the logical table source has been renamed to `LTS1 Sales` for the `Fact-Sales` logical table. `LTS1 Sales` maps to the `Fact_D1_ORDERS2` physical table (not shown here).

In the BMM layer, objects are separated and simplified into facts and dimensions. Recall that facts contain the business measures and dimensions contain descriptive attributes that qualify the facts. Each business model contains one or more dimension tables and fact tables. A business model must contain at least one logical fact table and one logical dimension table. A dimension table appears with a white table icon. A fact table icon includes a hash (#) sign.

The business model also contains logical dimensions. In this example, the logical dimensions are `Customer`, `Time`, and `Product`. Logical dimensions introduce formal hierarchies into a business model, establish levels for data groupings and calculations, and provide drill-down paths in end-user tools. Logical dimensions are discussed in more detail in the lesson titled "Creating Logical Dimensions and Level-Based Measures."

Business Model Mappings

- BMM layer objects map to source data objects in the Physical layer.
- Mappings are typically not one-to-one:
  - Business models may map to multiple data sources.
  - Logical tables may map to multiple physical tables.
  - Logical columns may map to multiple physical columns.



ORACLE

Business Model Mappings

Business model objects map to the source data objects in the Physical layer. Typically, business models map to physical schemas, logical tables map to physical tables, logical columns map to physical columns, and so on. The arrows convey the mappings.

To automatically map objects in the BMM layer to sources in the Physical layer, you can drag Physical layer objects to a particular business model in the logical layer. When you drag a physical table to the BMM 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 this happens only the first time the tables are dragged to a new business model.

Mappings between business model objects and physical objects do not have to be one-to-one. In fact, a business model can have multiple data sources, a logical table can have multiple physical tables as sources, and a logical column can have multiple physical columns as sources. In the Administration Tool, mappings are specified in the logical BMM layer and not in the Physical layer. The mappings can also include transformations and formulas.

Business Model Mappings (continued)

You use the BMM layer to define the meaning and content of each physical source in business model terms. Oracle BI Server then uses these business model definitions to choose the appropriate sources for each data request.

Note also that, in this example, the business model object names have been changed. For example, the name of the logical table that maps to the `Dim_D1_CUSTOMER2` physical table has been changed to `Dim-Customer`. The name of the logical table source has been changed to `LTS1 Customer`. You can change the names of business model objects independently from corresponding physical model object names and properties. Likewise, you can change the names of physical model objects and properties independently from the names of business model objects. The tool keeps track of the changes.

Objects in the Business Model and Mapping Layer

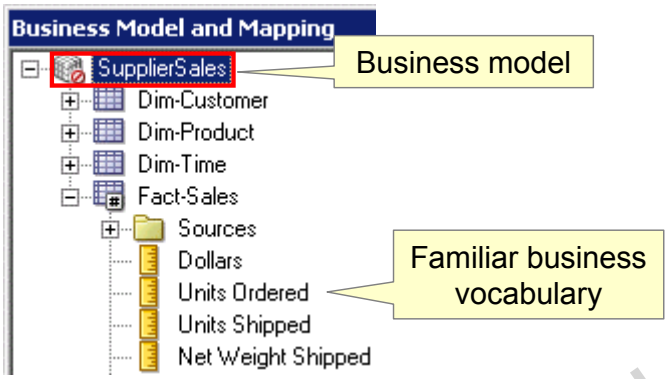
- Business model
- Logical tables
- Logical table sources
- Logical columns
- Logical primary keys
- Logical joins
- Measures

Objects in the Business Model and Mapping Layer

This slide identifies the BMM layer objects. Details about each type of object are discussed in the following slides.

Business Model Object

- Is the highest-level object in the BMM layer
- Contains the business model definitions and the mappings from logical to physical tables
- Simplifies the physical schema
- Captures how users think about their businesses by using their own vocabulary



Business Model Object

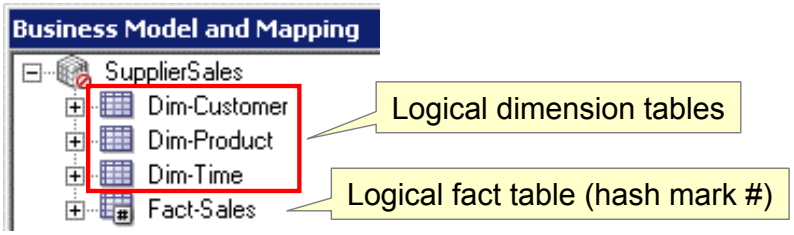
The business model object is the highest-level object in the BMM layer. The BMM layer of the Administration Tool must contain at least one business model. A business model object contains the business model definitions and the mappings from logical to physical tables for the business model. `SupplierSales` is the business model object in this example.

The main purpose of the business model is to capture how users think about their businesses by using their own vocabulary. The business model simplifies the physical schema and maps the users' business vocabulary to physical sources. Most of the vocabulary translates into logical columns in the business model.

You can create a business model automatically by dragging a schema from the Physical layer. You can create a business model manually by right-clicking in the white space in the BMM layer and selecting New Business Model.

Logical Tables

- Represent fact or dimension data
- Can be created:
  - Automatically by dragging tables from the Physical layer
  - Manually by right-clicking the business model and selecting New Object > Logical Table
- Can be modified without affecting Physical layer objects



Logical Tables

Logical tables exist in the BMM layer and represent fact or dimension data. Logical fact tables are indicated by a hash mark (#). The logical schema defined in each business model must contain at least two logical tables, and you must define relationships between them. In other words, there must be at least one logical fact table and one logical dimension table.

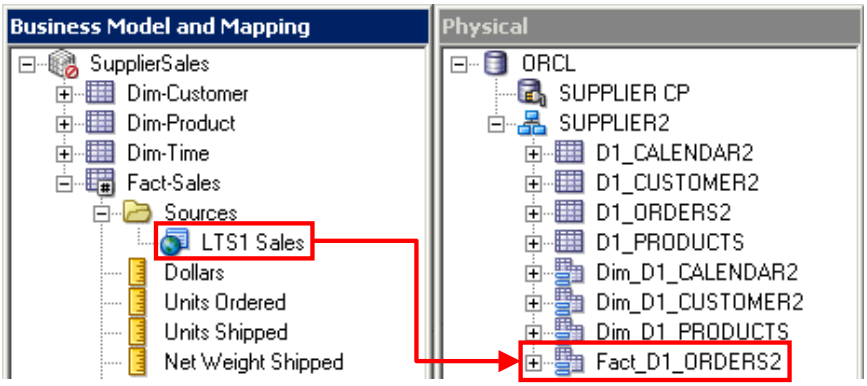
Typically, you create logical tables by dragging a physical table from the Physical layer to a business model in the BMM layer. If a table does not exist in your physical schema, you can create the logical table manually by right-clicking the business model object and selecting New Object > Logical Table. A drag operation is usually the fastest method for creating objects in the BMM layer. If you drag physical tables from the Physical layer to the BMM layer, the columns belonging to the table are also copied. After you drag objects to the BMM layer, you can modify them without affecting the objects in the Physical layer. For example, you can change the logical table name, reorder tables and columns, and so on.

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 BMM layer, the logical links between the new or newly dragged tables and the previously dragged tables must be created manually.

Logical Table Sources

Define the mappings from a logical table to a physical table:

- Logical tables may have multiple logical table sources.
- One logical table source may map to many physical sources.



Logical Table Sources

Logical table sources define the mappings from a logical table to a physical table. A logical table's `Sources` folder contains the logical table sources.

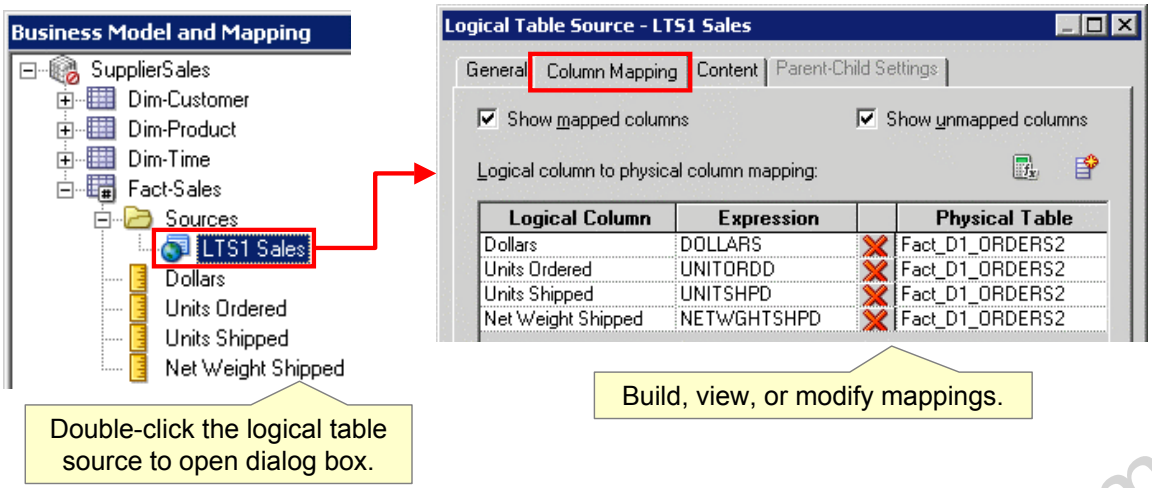
Logical tables can, and routinely do, have many physical table sources. For example, revenue information may come from multiple data sources in a business. You might have three different business units (each with its own order system) from which you get revenue information. As another example, you might periodically summarize revenue from an orders system or a financial system and use this aggregate table for high-level reporting.

In this example, `LTS1` is the only logical table source for the `Fact-Sales` table in the `SupplierSales` business model. It maps to the `Fact_D1_ORDERS2` alias table in the Physical layer.

When you create logical tables and columns by dragging from the Physical layer, the logical table sources are generated automatically and have the same name as the physical source. In this example, the logical table source was renamed from `Fact_D1_ORDERS2` to `LTS1 Sales`. To create a logical table source manually, right-click a logical table and select New Object > Logical Table Source.

Logical Table Source: Column Mappings

Click the Column Mapping tab in the Logical Table Source dialog box to build, view, or modify logical-to-physical column mappings:



Column Mappings

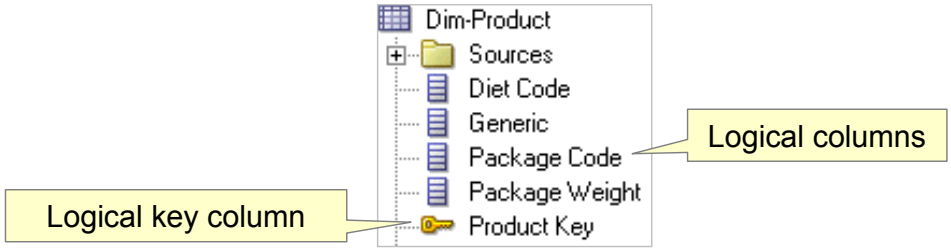
Double-click a logical table source to open the Logical Table Source dialog box. Click the Column Mapping tab to build, view, or modify logical-to-physical column mappings. A mapping can also be used to specify transformations that occur between the Physical layer and the BMM layer. The transformations can be simple, such as changing an integer data type to a character type, or more complex, such as applying a formula to find a percentage of sales per unit of population.

Within a single logical table source, a logical column can be mapped to only one physical column in one physical table. For example, the screenshot shows that the Units Shipped logical column maps to the `UNITSHPD` physical column in the `Fact_D1_ORDERS2` physical table in the Physical layer.

With multiple logical table sources, it is possible to map a logical column to multiple physical columns in multiple physical tables. You learn more about multiple mappings in the lesson titled “Managing Logical Table Sources.”

Logical Columns

- Represent the business view of the data
- May map to many columns in the Physical layer
- May be defined by other logical columns
- Can be created:
  - Automatically by dragging tables or columns from the Physical layer
  - Manually by right-clicking a logical table and selecting New Object > Logical Column



Logical Columns

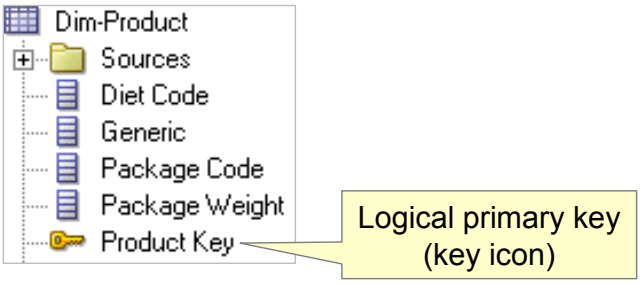
Logical columns are created automatically by dragging tables from the Physical layer to the BMM layer. You can manually create logical columns that involve calculations based on other logical columns. To create a logical column manually, right-click a logical table and select New Object > Logical Column.

A single logical column may map to many physical columns in the Physical layer. For example, a logical column might map to both a physical column in a detail table and a physical column in an aggregate table. Oracle BI Server determines which table and column to use based on the repository configuration and the query request. Logical columns may also be defined by other logical columns. For example, a logical column may use an existing logical column in an expression or calculation.

Logical columns are displayed in a tree structure expanded 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 the key icon. Logical columns can be renamed and reordered without affecting the columns in the Physical layer.

Logical Primary Keys

- Define unique identifiers for logical tables
- Define the lowest level of detail of a logical table
- Are required for each logical dimension table for a valid repository



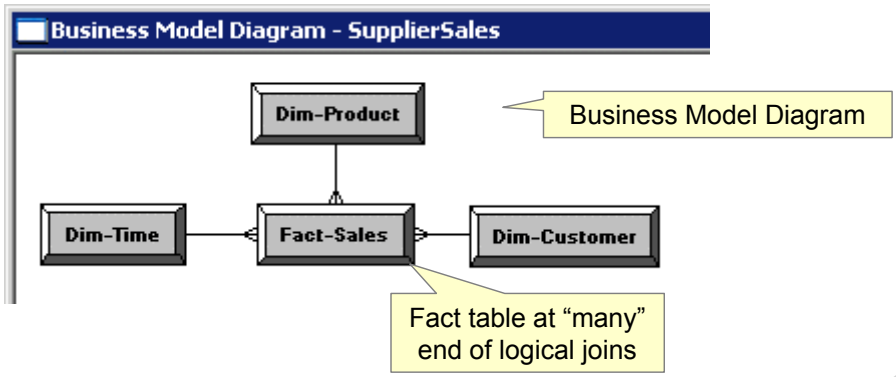
Logical Primary Keys

Logical primary keys define unique identifiers for logical tables. For a business model to be valid, each logical dimension table must have a logical primary key. Logical fact tables do not require keys for a business model to be valid. Logical primary keys can consist of one or more logical columns. The logical key defines the lowest level (the most detailed level) of information of any source in the logical table. After creating tables in the BMM layer, you can manually specify a primary key for each table by right-clicking the table and selecting New Object > Logical Key, or by double-clicking a table and clicking the Key tab.

You can also define logical keys during the process of defining logical joins, which is discussed in the next slide. Logical primary key columns are denoted by the key icon.

Logical Joins

- Define join relationships in the BMM layer
- Are required for a valid business model
- Help Oracle BI Server translate logical requests against the business model into SQL queries against the physical data sources



Logical Joins

Logical joins express the cardinality relationships between logical tables and are a requirement for a valid business model. Specifying the logical table joins is required so that Oracle BI Server has necessary metadata to translate logical requests against the business model into SQL queries against the physical data sources. Logical joins help Oracle BI Server understand the relationships between the various pieces of the business model.

When a query is sent to Oracle BI Server, the server determines how to construct physical queries by examining how the logical model is structured. Examining logical joins is an integral part of this process. 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.

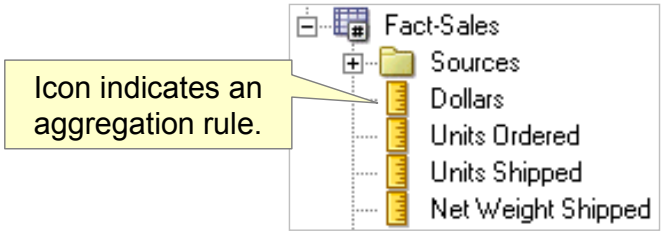
Typically, you use the Business Model Diagram to build logical joins. The method is similar to building physical joins by using the Physical Diagram in the Physical layer. It is also possible to construct joins by using the Joins Manager,.

Logical joins are automatically created if both of the following statements are true:

- You create the logical tables by simultaneously dragging all required physical tables to the BMM layer.
- The logical joins are the same as the joins in the Physical layer.

Measures

- Are the facts that a business uses to evaluate its performance
- Are calculations that define measurable quantities
- Are created on logical columns in the fact table
- Have a defined aggregation rule



Measures

Measures are the facts that a business uses to evaluate its performance. In a business model, some logical columns serve as measures. Measures need to be computed correctly from the base physical data when Oracle BI Server receives a logical query from an end-user tool.

You specify aggregation rules for mapped logical columns that are measures. Aggregation should be performed only on measure columns, with the possible exception of the aggregation rules `COUNT` and `COUNTDISTINCT`. Measure columns should exist only in logical fact tables. In this example, `Dollars`, `Units Shipped`, `Units Ordered`, and `Net Weight Shipped` are aggregated by summing.

By default, a column has no aggregation rule defined. When you define an aggregation rule on a column, the icon changes to reflect this. The aggregation rule calculations are sent to the database or processed by Oracle BI Server. The method for defining aggregation rules is described in more detail later in this lesson.

ABC Example

Create a dimensional model to represent ABC's business.

- `Time`, `Product`, and `Customer` are logical dimensions.
- `Fact-Sales` is the central fact table containing data for orders, dollars, units ordered, and units shipped.

Time	Product	Customer	Fact-Sales
Year	Type	Region	Dollars
Quarter	Subtype	District	Units Ordered
Month	Description	Sales Rep	Units Shipped
Day	Price	Customer	

ABC Example

In the previous lesson, you identified the source of the information needed to create the ABC business model and imported tables and columns into the Physical layer. Now you use the BMM layer of the repository to construct the business model for the information that ABC wants to analyze. This model should represent the logical-to-physical mapping of sales, time, product, and customer data to support the dimensional model.

In this lesson, you arrange ABC's data in a business model so that it makes sense for answering business questions about performance measurements, such as total sales dollars, units ordered, and units shipped. You use an iterative approach to building the business model by starting small with simple aggregation rules and enhancing the model later.

Implementation Steps

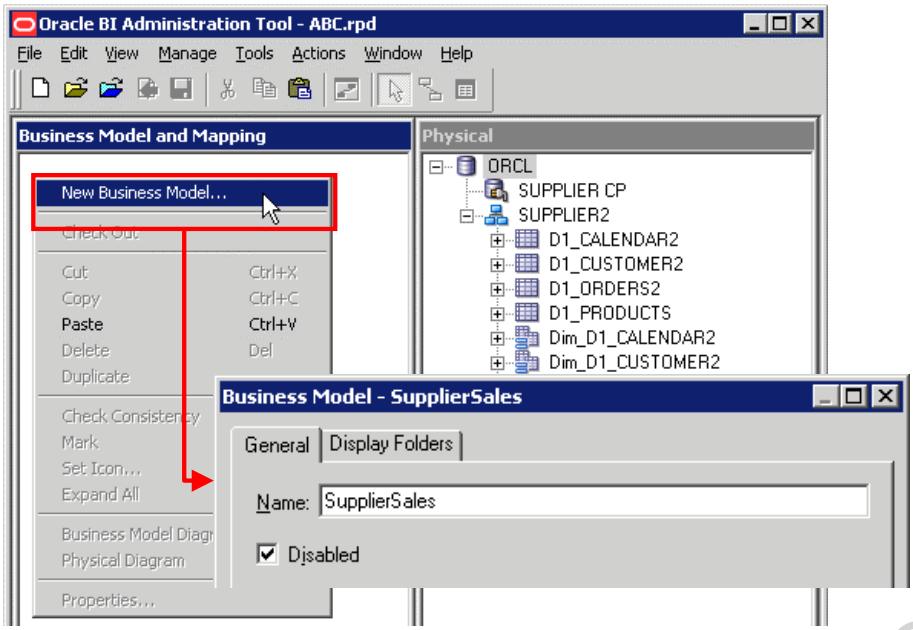
1. Create the logical business model.
2. Create the logical tables and columns.
3. Define the logical joins.
4. Modify the logical tables and columns.
5. Define the measures.

Implementation Steps

At a high level, these are the steps to build a business model in the BMM layer. Each step is covered in detail in the following slides.

1. Create the Logical Business Model

Right-click in the BMM layer and select New Business Model:



1. Create the Logical Business Model

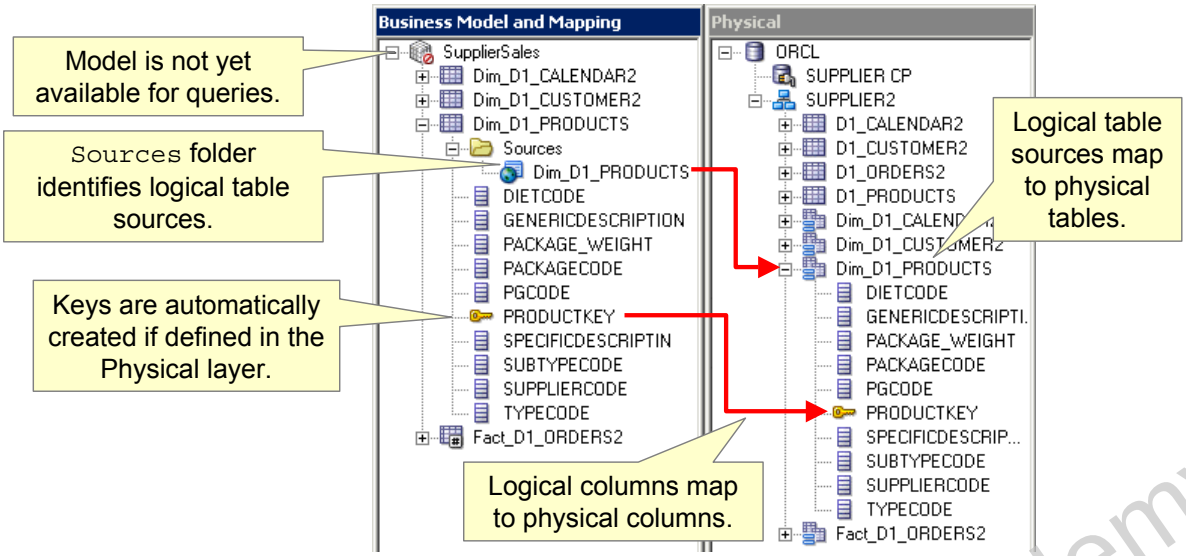
There are two ways to create the logical business model. You can create the business model manually, or you can drag a data source from the Physical layer.

This slide illustrates the manual method. Right-click in the white space of the BMM layer below any existing objects and select the New Business Model option. Specify a name for the business model.

The alternative method is to drag the `SUPPLIER2` schema from the Physical layer to the BMM layer and then rename the business model `SupplierSales` in the BMM layer.

2. Create the Logical Tables and Columns

Drag physical table objects from the Physical layer to the business model, or right-click the business model and select New Object > Logical Table.



2. Create the Logical Tables and Columns

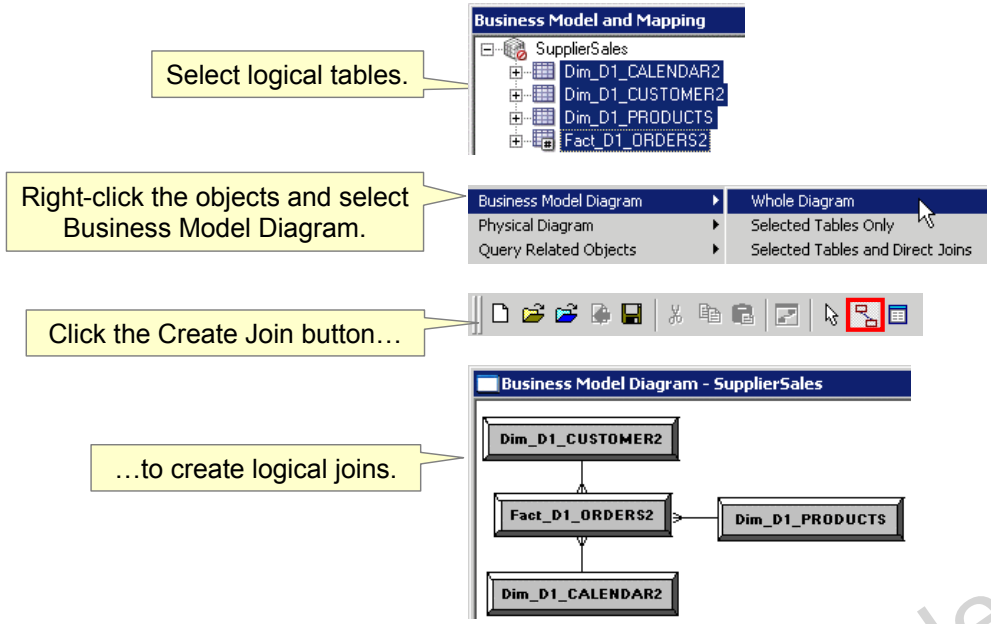
Typically, you create logical tables by dragging physical tables from the Physical layer to a business model in the BMM layer. In this example, you drag the alias tables. Logical tables and columns can also be created manually in the Business Model and Mapping layer by right-clicking the business model and selecting New Object > Logical Table. Drag operations are usually the fastest method for creating objects in the BMM layer. When you drag physical tables from the Physical layer to the BMM layer, the columns belonging to the table are also copied.

The red symbol on the business model folder indicates that it is not enabled for querying. By default, a newly created business model is disabled for querying. After a business model is valid and consistent, you can then enable it for querying. You learn more about checking business model consistency in the lesson titled “Testing and Validating a Repository.”

Note that this is an example of a simple business model with one-to-one mappings between BMM layer objects and Physical layer objects. Typically, however, you use the BMM layer to simplify the physical data model. For example, multiple dimensional tables that reference the same class of attributes could be collapsed into a single logical table source. Similarly, multiple fact tables that contain related measures could be collapsed into a single logical table source.

3. Define the Logical Joins

Define logical joins in the Business Model and Mapping layer by using techniques similar to those used in the Physical layer.



3. Define the Logical Joins

This slide illustrates the steps for defining logical joins in the BMM layer. The steps are similar to those used to define joins in the Physical layer. The differences are that you use the Business Model Diagram instead of the Physical Diagram and logical joins instead of foreign key joins.

Recall that logical joins are automatically created if both of the following statements are true:

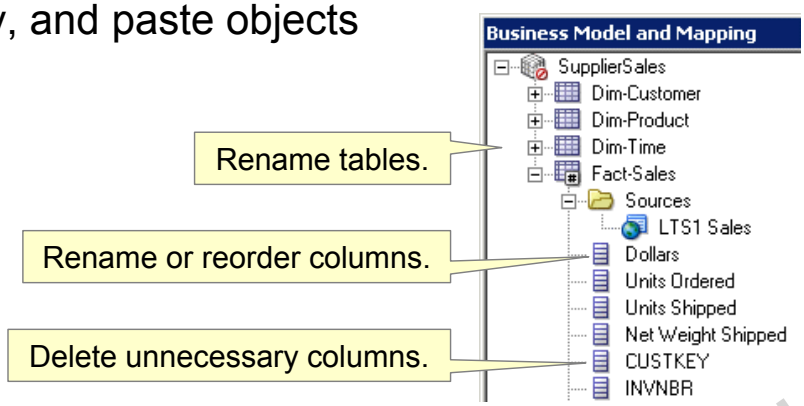
- You create the logical tables by simultaneously dragging all required physical tables to the BMM layer.
- The logical joins are the same as the joins in the Physical layer.

In all other cases, you must create logical joins manually by using the Business Model Diagram.

4. Modify the Logical Tables and Columns

Use an object's properties window or the user interface to:

- Rename tables and columns
- Reorder columns
- Add or delete tables and columns
- Add or delete sources
- Cut, copy, and paste objects



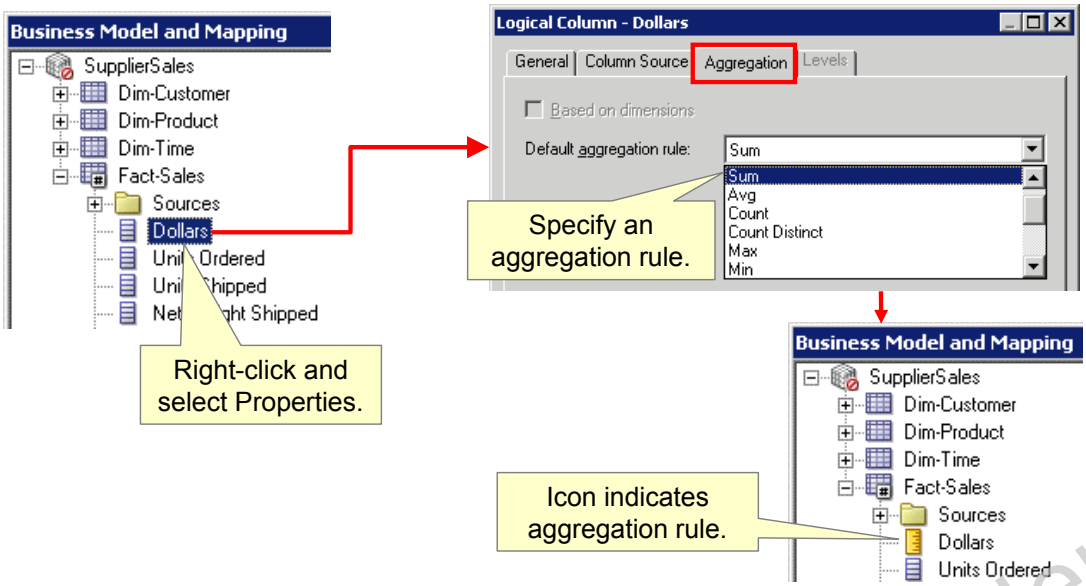
4. Modify the Logical Tables and Columns

After you create objects in the BMM layer, there are a number of ways to modify the objects. Double-click an object to open the Properties window for the object. Depending on the object that you are modifying, you can rename the object, reorder columns in the table, add or delete columns, add or delete physical sources, and add or delete primary and foreign keys. You can also perform many of the same tasks by right-clicking a table object. In addition, you can perform additional tasks such as copying and pasting objects, duplicating objects, and so on.

You can change the names of business model objects independently from corresponding physical model object names. Likewise, you can change the names of physical model objects independently from the names of business model objects. The tool tracks the changes.

5. Define the Measures

Right-click the logical column and select Properties > Aggregation:



5. Define the Measures

A critical step in setting up your business model is configuring your business measures. One method is to specify aggregation rules for mapped logical columns that are measures. Aggregation should be performed only on measure columns. Measure columns should exist only in logical fact tables.

To specify a default aggregation rule for a measure column, perform the following steps:

1. In the Business Model and Mapping layer, double-click a logical column, or right-click and select Properties to open the Properties window.
2. In the Logical Column dialog box, click the Aggregation tab.
3. On the Aggregation tab, select one of the aggregate functions from the "Default aggregation rule" drop-down list. The function you select is always applied when an end user or application requests the column in a query.

In this example, you are summing the `Dollars` column in the `Fact-Sales` table. If a user requests dollars by customers in a query, the results display the `SUM` (total dollars) for each customer based on records in the `Fact-Sales` table.

Note: It is possible to select multiple logical columns simultaneously and apply the same aggregation rule to them.

Design Tips

- Rename logical objects.
  - Use names acceptable to the organization and understood by users.
  - Use short names to minimize space on reports.
  - Create unique names for all columns.
  - Avoid using the same logical column name as a logical table or business model.
- Do not delete logical columns that define the content of logical table sources.

Design Tips

Rename objects in the business model by using names that are acceptable to the business organization and familiar and understood by business users.

Some columns can be deleted in the BMM layer (for example, logical fact table keys that were created by dragging a fact table from the Physical layer). However, be careful not to delete columns that are used to define the content of logical table sources.

Summary

In this lesson, you should have learned how to:

- Identify the objects of the Business Model and Mapping layer of a repository
- Build the business model
- Create simple measures

Practice 4-1 Overview: Creating the Business Model

This practice covers the following topics:

- Creating a business model
- Creating logical tables
- Creating logical keys and logical joins
- Renaming and reordering logical objects
- Deleting unnecessary logical objects

Practice 4-1 Overview: Creating the Business Model

In the previous set of practices, you created the physical layer of the repository. You are now ready to begin building the business model in the Business Model and Mapping layer of the repository. The main purpose of the business model is to capture how users think about their businesses by using their own vocabulary.

Practice 4-2 Overview: Creating Simple Measures

This practice covers the following topics:

- Examining logical-to-physical column mappings
- Creating simple measures

Practice 4-2 Overview: Creating Simple Measures

The `SupplierSales` business model is now defined in the Business Model and Mapping layer. In this practice, you review the logical-to-physical table and column mappings to better understand the relationships that exist between logical tables and their logical table sources. You then create measures by setting aggregation rules for logical columns.

5

Building the Presentation Layer of a Repository

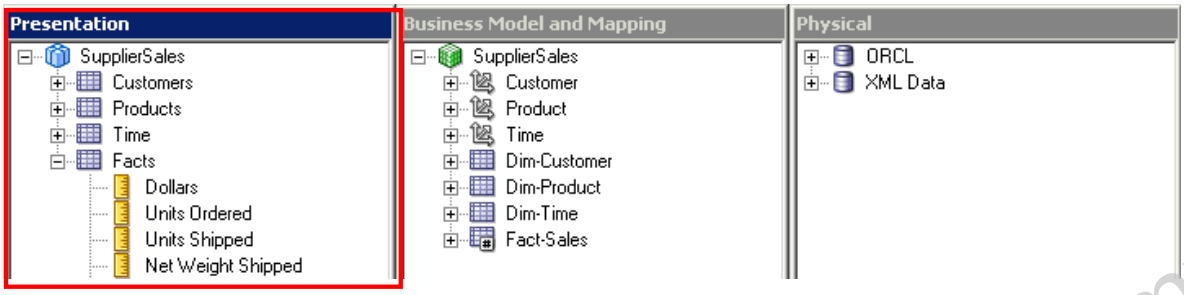
Objectives

After completing this lesson, you should be able to:

- Identify the objects in the Presentation layer of a repository
- Modify the properties of Presentation layer objects
- Build the Presentation layer of a repository

Presentation Layer

- Provides a way to present a customized view of a business model to users
- Exposes only the data that is meaningful to users
- Organizes the data in a way that aligns with the way users think about the data
- Renames data as necessary for the set of users



Presentation Layer

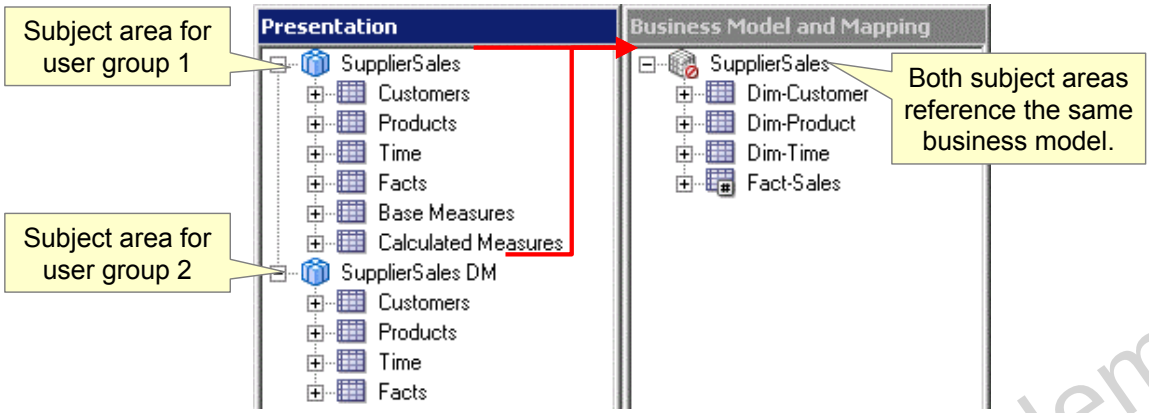
The Presentation layer is built after the Physical layer and the Business Model and Mapping layer and adds a level of abstraction over the Business Model and Mapping (BMM) layer. It is the view of the data seen by end users. The Presentation layer provides a means to further simplify or customize the BMM layer for users. For example, you can hide key columns or reorganize the data into catalogs and folders. Simplifying the view of the data for users makes it easier to craft queries based on their business needs.

You can create Presentation layer objects by dragging objects from the BMM layer. Corresponding objects are automatically created in the Presentation layer. You can also create Presentation layer objects manually.

Subject Areas

Organize and simplify the business model for a set of users:

- A single subject area must be populated with content from a single business model . It cannot span business models.
- Multiple subject areas can reference the same business model.



Subject Areas

Subject areas (formerly called *presentation catalogs*) allow you to show different views of a business model to different sets of users. Subject areas have to be populated with content from a single business model. They cannot span business models. However, multiple subject areas can refer to the same business model. The screenshot shows two different subject areas that refer to the same underlying business model. If there are different user bases, you can create different subject areas to meet their specific needs.

Creating a Subject Area

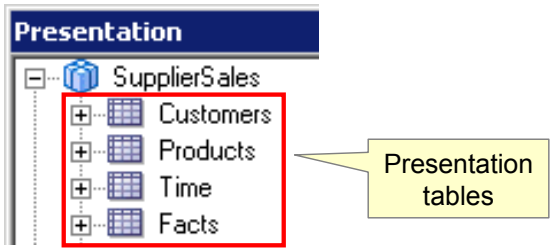
There are several ways to create a subject area in the Presentation layer. The recommended method is to drag a business model from the BMM layer to the Presentation layer. With this method, logical tables and columns automatically become presentation tables and columns. You can also right-click in the Presentation layer and select New Subject Area. You can then modify the Presentation layer based on how you want to organize the information for users. Presentation layer subject areas are seen as subject areas by end users when they create analyses in the Oracle BI user interface.

Modifying Presentation Tables in a Subject Area

Subject areas contain presentation tables. Click the Presentation Tables tab in the Subject Area properties dialog box to reorder, sort, or delete Presentation layer tables. You can also use this tab to access the Presentation Table dialog box, where you can create and edit tables. Changes to the Presentation layer do not impact corresponding objects in the BMM layer.

Presentation Tables

- Organize presentation columns into categories that make sense to users
- May contain columns from one or more logical tables
- Can be modified independently of logical tables
- Can be created:
  - Automatically by dragging logical tables from the BMM layer
  - Manually in the Presentation layer



Presentation Tables

Use presentation tables to organize columns into categories that make sense to users. Presentation tables contain presentation columns. 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.

Creating a Presentation Table

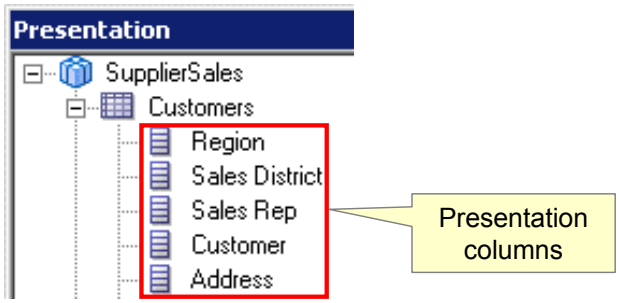
There are several ways to create a presentation table in the Presentation layer. Presentation tables are created automatically when you drag a business model to the Presentation layer to create a subject area. Another method is to drag a logical table from the BMM layer to an existing subject area in the Presentation layer. You can also right-click a subject area and select New Presentation Table. You can use presentation tables to modify the Presentation layer based on how you want to organize the information for users. Presentation tables in the Presentation layer are seen as folders by end users.

Modifying Presentation Columns in a Presentation Table

Presentation tables contain presentation columns. Use the Columns tab in the Presentation Table properties dialog box to reorder, sort, or delete Presentation layer columns. You can also use this tab to access the Presentation Column dialog box, where you can create and edit columns.

Presentation Columns

- Define the columns that are used to build analyses in the Oracle BI user interface
- Map to logical columns in the BMM layer
- Can be created:
  - Automatically by dragging logical tables or columns from the BMM layer
  - Manually in the Presentation layer



Presentation Columns

Presentation columns define the columns that are used to build analyses in the Oracle BI user interface. When presentation columns are dragged from the BMM layer, presentation column names are, by default, identical to the logical column names in the BMM layer. To provide a convenient organization for your end users, you can drag a column from a single logical table in the BMM layer to multiple presentation tables. This allows you to create categories that make sense to users. For example, you can create several presentation tables that contain different classes of measures—one containing volume measures, another containing share measures, yet another containing measures from a year ago, and so on.

Creating a Presentation Column

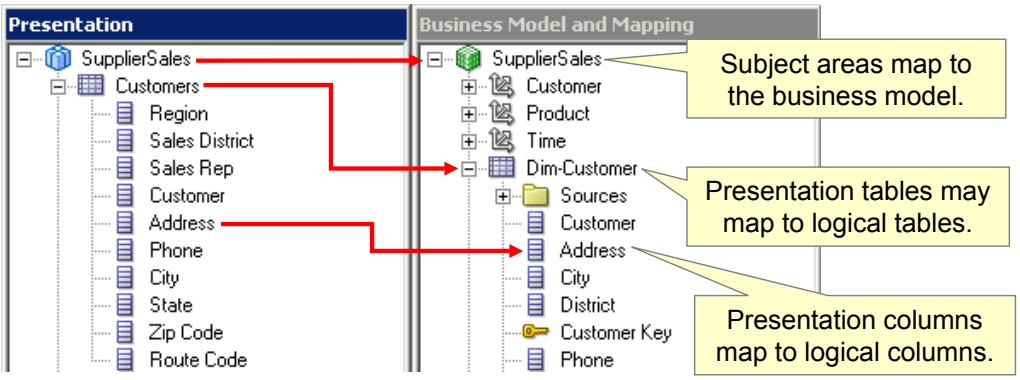
There are several ways to create a presentation column in the Presentation layer. Presentation columns are created automatically when you drag a logical table to the Presentation layer to create a presentation table. Another method is to drag a logical column from the BMM layer to a presentation table in the Presentation layer. You can also right-click a presentation table and select New Presentation Column. Presentation columns in the Presentation layer are seen as columns by end users.

Modifying Presentation Columns

Double-click a presentation column to open the Presentation Column dialog box.

Presentation Layer Mappings

Presentation layer objects map to objects in the Business Model and Mapping layer.



Presentation Layer Mappings

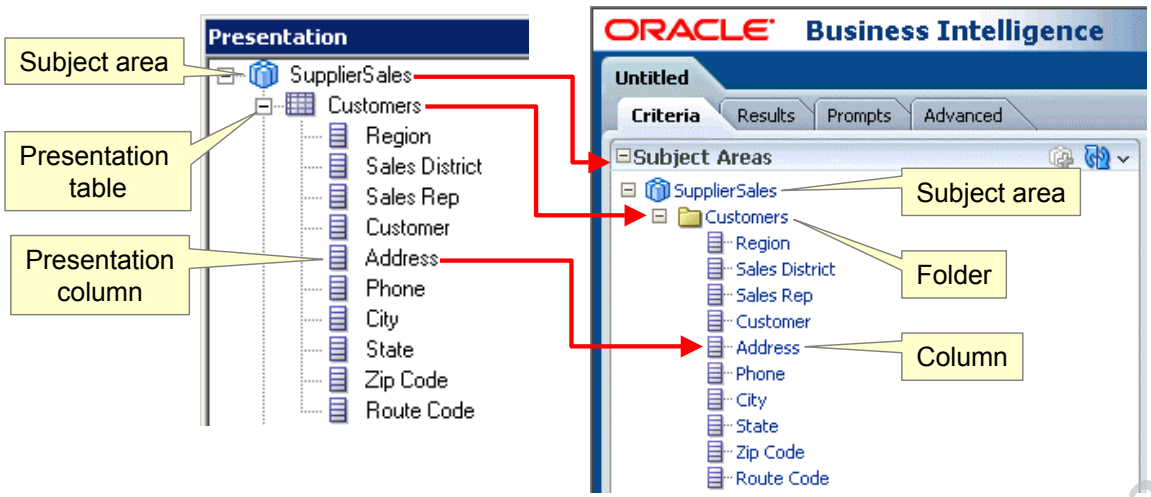
Presentation layer objects map to objects in the BMM layer. Subject areas map to business models, presentation tables typically map to logical tables, and presentation columns map to logical columns. A subject area can map to only a single business model and cannot span multiple business models. However, multiple subject areas can reference the business model and a presentation table can contain columns from one or more logical same tables. Thus, you can use subject areas and tables to organize columns into categories that make sense to the users.

A presentation table does not necessarily have a one-to-one mapping to a logical table. Presentation tables are folders used for organizing columns and can be organized and structured differently from logical tables in the BMM layer.

When objects are dragged from the BMM layer, the subject area, table, and column names are, by default, identical to the logical names in the BMM layer, but they can be renamed. The names and object properties of the presentation tables are independent of the logical table properties. Note that the screenshot provides simple examples and that presentation objects need not exactly mirror business model logical objects (as the example in the slide implies).

Defining the User Interface in the Presentation Layer

Presentation layer objects define the interface that users see to query data from the data sources.



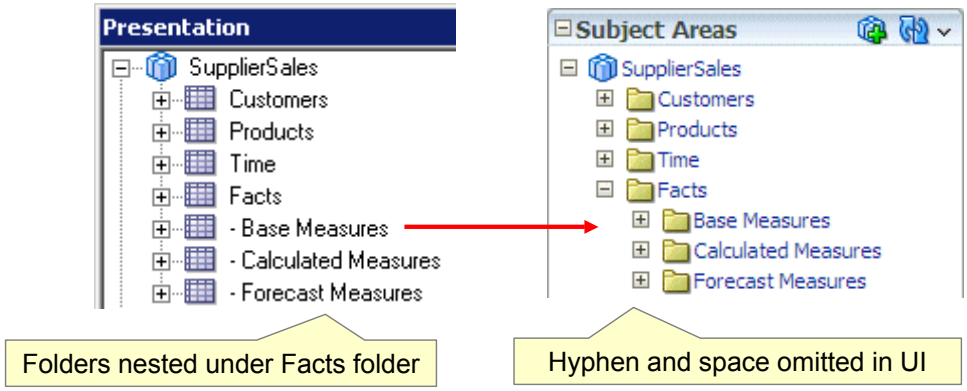
Defining the User Interface in the Presentation Layer

Presentation layer objects define the interface that users see to query data from the data sources. Subject areas in the Presentation layer appear as subject areas. Presentation tables in the Presentation layer appear as folders. Presentation columns appear as columns.

Nesting Presentation Tables

To give the appearance of nested folders:

1. Prefix the name of the presentation folder to be nested with a hyphen and a space.
2. Place it after the folder in which it nests.



Nesting Presentation Tables

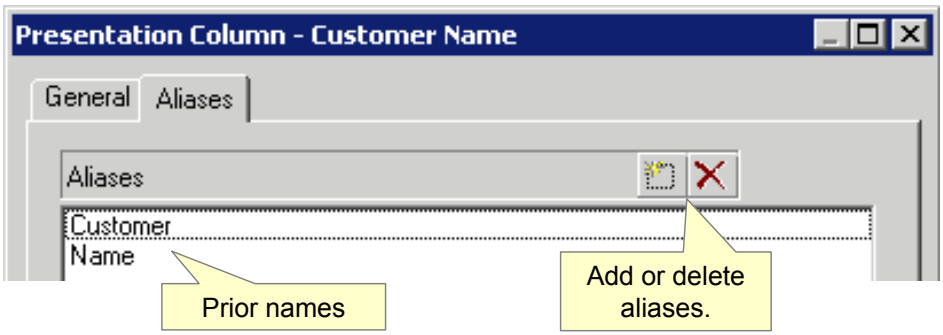
You can use the Administration Tool to update Presentation layer metadata to give the appearance of nested folders in the user interface. To do this, open the Properties dialog box for the presentation table that corresponds to the folder that you want to nest. Add `->` to the beginning of the Description field and then place the table after the presentation table in which it nests.

Alternatively, you can prefix the name of the presentation table to be nested with a hyphen and a space and then place the table after the presentation table in which it nests. For example, to nest the Base Measures folder under the Facts folder, place Base Measures directly after Facts and change its name to `- Base Measures`. When the user interface displays the folder name in the left pane, it omits the hyphen and space from the folder name. In this example, there are three folders nested under the Facts folder: Base Measures, Calculated Measures, and Forecast Measures.

Note that these techniques do not provide true folder nesting. If you move the parent presentation table, the “child” presentation tables do not move with it. These techniques provide only the appearance of nesting.

Aliases

Keep track of changes to Presentation layer objects.

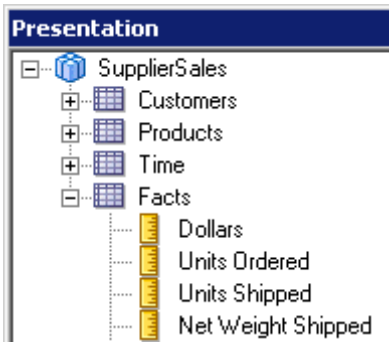


Aliases

An Aliases tab appears in the Subject Area, Presentation Table, and Presentation Column property dialog boxes. If you modify a Presentation layer object, this tab keeps track of any prior names. Aliases are used to maintain compatibility with previously written queries after an object has been modified. You also can use this tab to specify or delete an alias for the Presentation layer objects. In this example, the Customer Name presentation column had two prior names: Customer and Name.

ABC Example

Build the Presentation layer to present users with a customized view of ABC's data.



ABC Example

In the previous lesson, you built the business model for the information that ABC wants to analyze in the BMM layer of the repository. In this lesson, you arrange ABC's data in the Presentation layer to present a customized view of that data to users.

Implementation Steps

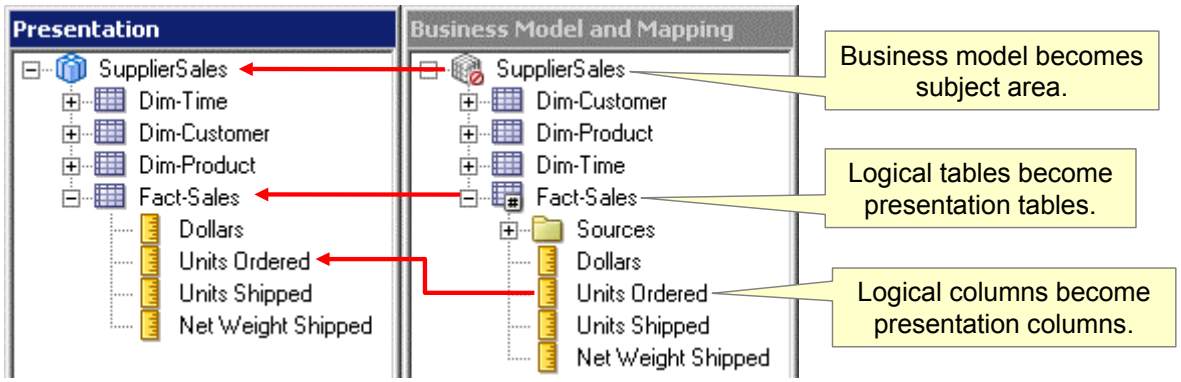
1. Create a new subject area.
2. Rename tables.
3. Reorder tables.
4. Delete columns.
5. Rename columns.
6. Reorder columns.

Implementation Steps

At a high level, these are the steps to build the Presentation layer. Each step is covered in detail in the following slides.

1. Create a New Subject Area

Drag the `SupplierSales` business model from the Business Model and Mapping layer to the Presentation layer to automatically create Presentation layer objects.



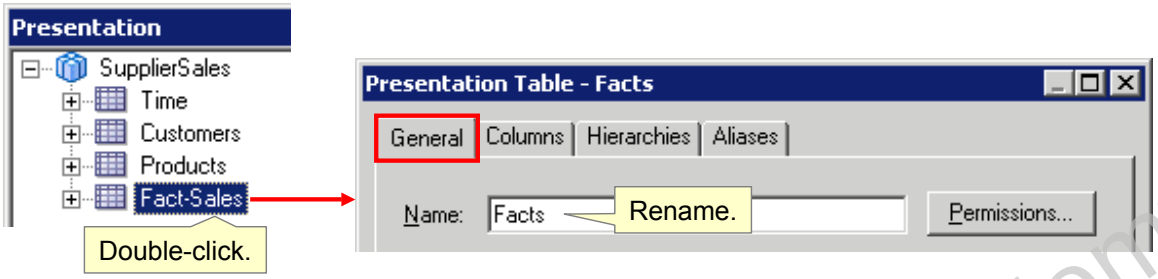
1. Create a Subject Area

Drag the `SupplierSales` business model from the BMM layer to the Presentation layer. Dragging automatically creates a new subject area with presentation tables and columns. You can also right-click the white space in the Presentation layer and select New Subject Area to build a subject area manually.

2. Rename Tables

To rename tables in the Presentation layer, use any of the following methods:

- Use the General tab in the properties dialog box.
- Right-click a table and select Rename.
- Highlight a table to make the name editable, and then enter a new name.
- Use the Rename Wizard.



2. Rename Tables

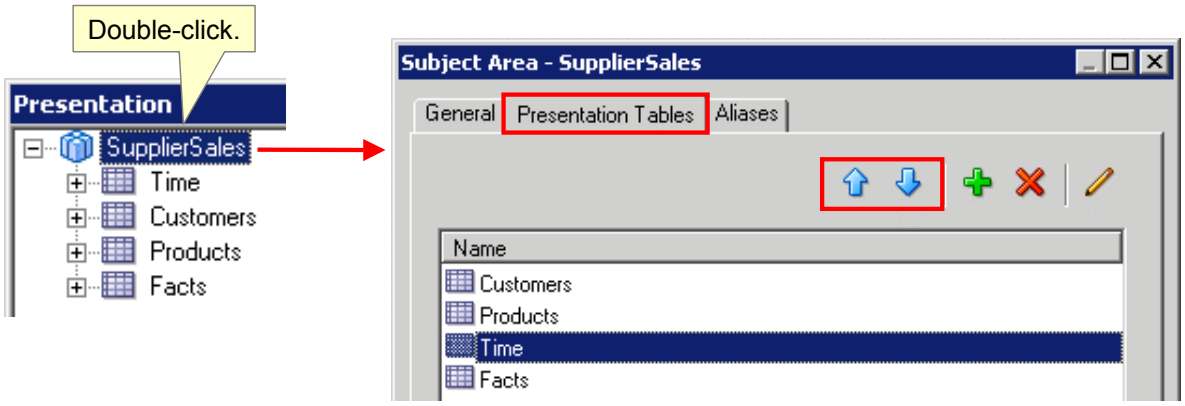
There are several methods for renaming tables in the Presentation layer:

- Double-click the table and click the General tab.
- Right-click the table in the Presentation layer and select Rename.
- Slowly double-click the table in the Presentation layer to make it editable and then enter the new name.
- Use the Rename Wizard

A best practice is to rename objects in the BMM layer and minimize renaming in the Presentation layer. However, if terminology is different between user groups who use the same business model, it may make sense to rename objects in the Presentation layer so that each user group is presented with familiar names.

3. Reorder Tables

To reorder tables, open the Subject Area properties box.
Then use the Up and Down buttons, or simply drag the tables.

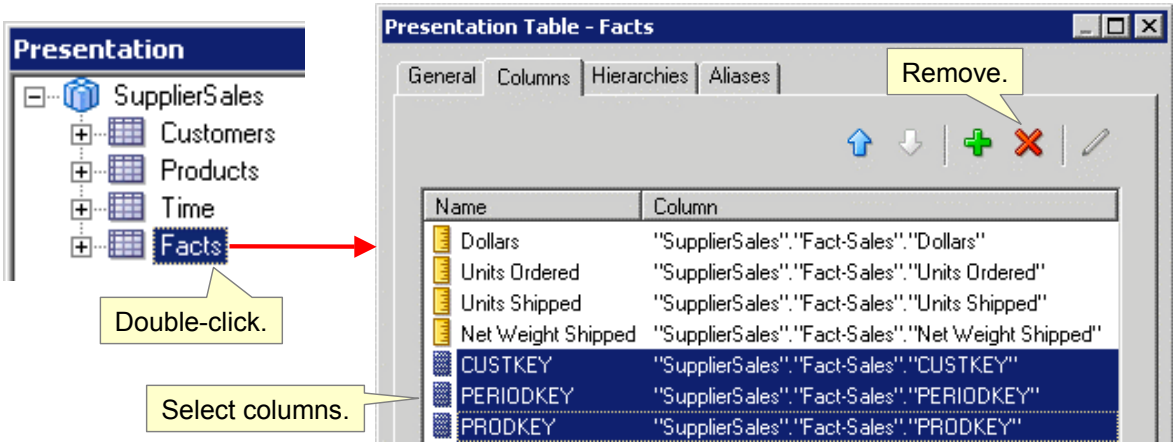


3. Reorder Tables

To reorder tables in the Presentation layer, double-click the subject area to open the Subject Area properties box. Then click the Presentation Tables tab. You can use the Up and Down arrows or drag to rearrange the tables.

4. Delete Columns

Delete unnecessary columns to simplify the content and make it more understandable to users.



4. Delete Columns

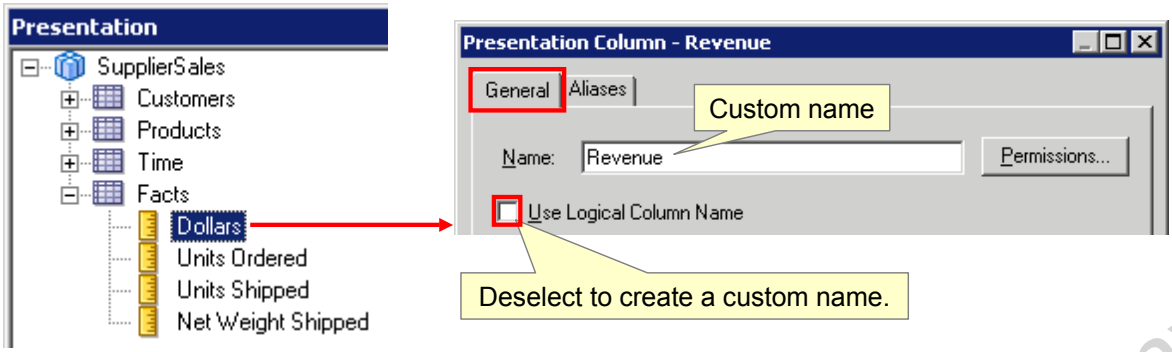
To delete columns in a presentation table, double-click the presentation table and click the Columns tab. Select the columns you want to delete and click the Remove button. You can also delete presentation columns by right-clicking the column in the Presentation layer and selecting Delete.

You delete columns because you may not want to expose all the logical columns of a business model in a presentation catalog. For example, key columns are often created just to handle load processing and other tasks; these columns are typically not useful and can be deleted unless they have an intrinsic meaning, such as a date key.

You can delete columns from the Presentation layer safely without affecting the existence of the logical column in the BMM layer.

5. Rename Columns

- Presentation columns use the logical column name by default.
- On the General tab in the column properties dialog box, deselect the Use Logical Column Name check box and enter a custom name.



ORACLE

5. Rename Columns

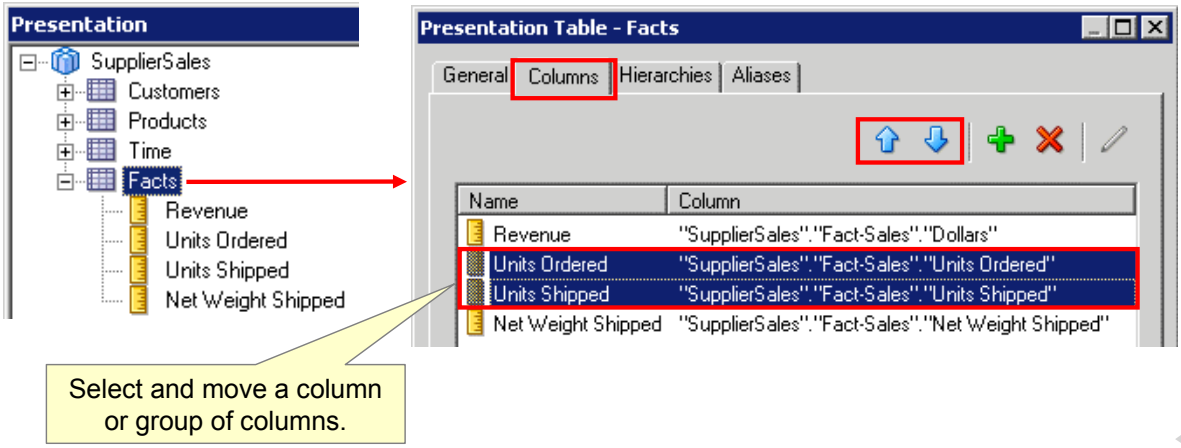
To rename a column in the Presentation layer, double-click the column and select the General tab. Deselect the Use Logical Column Name check box and enter the new name in the Name field.

Again, a best practice is to rename objects in the BMM layer and to minimize renaming in the Presentation layer. However, if terminology is different between user groups who use the same business model, it may make sense to rename objects in the Presentation layer so that each user group is presented with a familiar name.

You can rename a column in the Presentation layer safely without affecting the logical column name in the BMM layer. When the Use Logical Column Name check box is selected, Presentation columns are automatically renamed when the corresponding logical object is renamed.

6. Reorder Columns

To reorder columns, open the Presentation Table properties box. Then use the Up and Down buttons, or simply drag the columns.



6. Reorder Columns

To reorder columns in the Presentation layer, double-click the presentation table to open the Presentation Table properties box. Then click the Columns tab. Use the Up and Down arrows or drag to rearrange the columns.

Considerations

- Subject areas can map to only one business model.
- Multiple subject areas can map to the same business model.
- Presentation columns can come from multiple logical tables in a business model.
- When the Use Logical Column Name check box is selected, presentation columns are automatically renamed when the corresponding logical object is renamed.
- Presentation tables cannot have the same names as the subject areas.
- Presentation objects can be modified or deleted without affecting corresponding logical objects.

Considerations

By default, a presentation column uses the same name as its corresponding logical column in the BMM layer. If you rename a column in the BMM layer, corresponding presentation columns are automatically renamed wherever they appear in the Presentation layer.

But the reverse is not true. Renaming a presentation object does not affect the corresponding logical object in the BMM layer. However, the repository stores an alias for the object using the previous name.

Design Tips

- Use meaningful names.
- Names cannot contain single quotation marks (‘); the Administration Tool prevents it.
- Although permitted, avoid using double quotation marks (“).
- Keep presentation object names unique:
  - Giving presentation columns the same names as presentation tables can lead to inaccurate results.
  - Uniqueness allows SQL statements to be shorter because qualifiers are unnecessary.
- Group objects in meaningful ways.
- Eliminate unnecessary objects to reduce confusion.
- Use object description fields to convey information to users.
- Keep names short to save space on reports.

Summary

In this lesson, you should have learned how to:

- Identify the objects in the Presentation layer of a repository
- Modify the properties of Presentation layer objects
- Build the Presentation layer of a repository

Practice 5-1 Overview: Creating the Presentation Layer

This practice covers the following topics:

- Creating Presentation layer objects
- Customizing Presentation layer objects

Practice 5-1 Overview: Creating the Presentation Layer

You have already created the initial `SupplierSales` business model in the repository. You now create the Presentation layer of the repository, which enables you to expose the business model to users in the Oracle BI user interface so that they can build requests to analyze their data.

6

Testing and Validating a Repository

Objectives

After completing this lesson, you should be able to execute the steps to test and validate a repository.

Objectives

You can help ensure implementation success by testing and validating a repository before making it available to users.

Validating a Repository

The following steps validate whether a repository is constructed correctly and yields the expected query results:

1. Checking a repository for consistency
2. Enabling logging
3. Loading a repository
4. Checking a repository by running analyses
5. Inspecting the query log

Validating a Repository

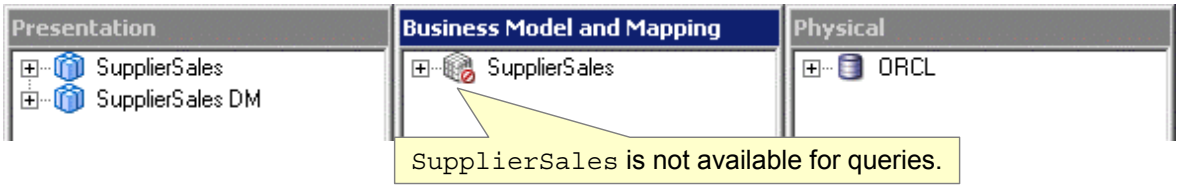
Validating a repository involves several steps that check whether a repository is constructed correctly and whether it yields the expected results when queries are executed in Oracle Business Intelligence (BI) end-user applications. At a high level, the steps include:

1. Checking a repository for consistency
2. Enabling logging
3. Loading a repository
4. Checking analysis results
5. Checking query results by inspecting the query log

Each of these steps is covered in detail in the following slides.

ABC Example

Validate and test the `SupplierSales` business model before making it available to users for querying.



ABC Example

Up to this point in the course, you have created ABC's `SupplierSales` business model but have not yet checked its consistency or made it available for querying by users. The red mark on the business model icon indicates that it is not available for queries. The following slides describe the tools and steps that you use to validate and test the model, and to make it available for user queries.

Consistency Check

Is a feature of the Administration Tool that checks whether a repository has met certain requirements, such as the following:

- All logical columns are mapped directly or indirectly to one or more physical columns.
- All logical dimension tables have a logical key.
- All logical tables have a logical join relationship to another logical table.
- There are at least two logical tables in the business model: a logical fact table and a logical dimension table. Both can map to the same physical table.
- There are no circular logical join relationships.
- A subject area exists for the business model.

Consistency Check

Repositories and the business models within them must pass the consistency check before you can make them available for queries. When a repository or business model is inconsistent, a detailed message alerts you to the nature of the inconsistency. The slide identifies some of the requirements for a consistent repository.

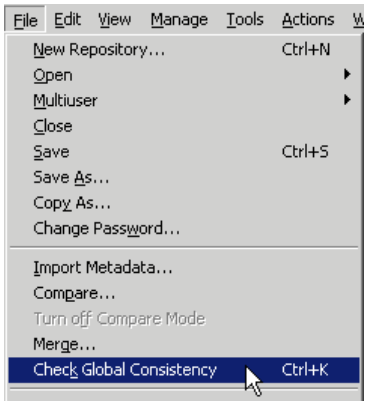
Passing a consistency check does not guarantee that a business model is constructed correctly, but it does rule out many common problems.

A consistency check also does not check the validity of objects outside the metadata using the connection. It checks only 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, a consistency check does not report these errors.

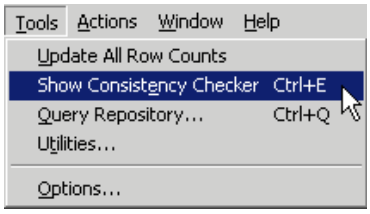
Checking Consistency

Check consistency for the entire repository or for individual repository objects by using the following methods:

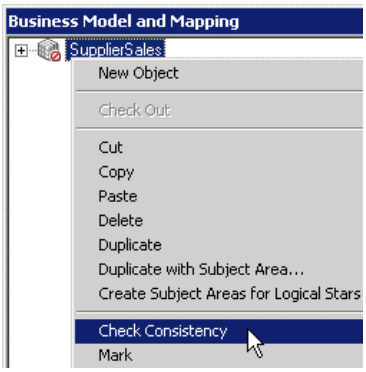
Use the File menu.



Use the Tools menu.



Right-click objects.



ORACLE

Checking Consistency

You can check consistency for the entire repository or for a single object.

To check the consistency of an entire repository, select File > Check Global Consistency.

To check the consistency of an individual repository object, select the object and then select Tools > Show Consistency Checker, or right-click the object and select Check Consistency. You can select multiple objects and check their consistency.

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

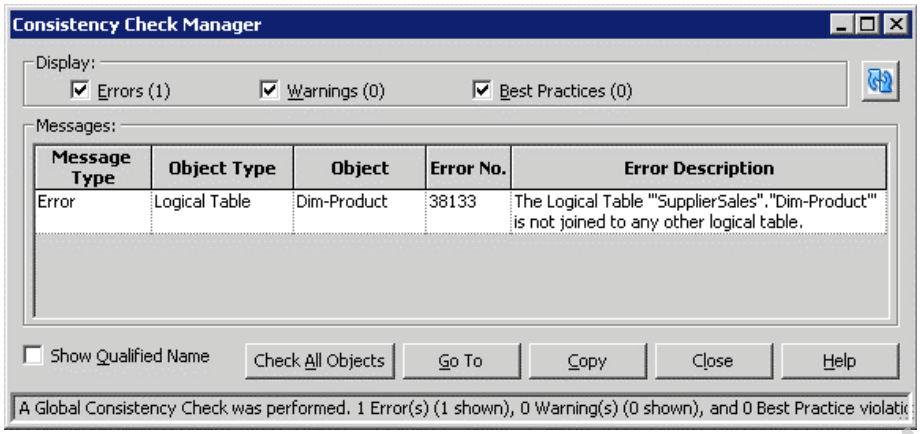
- **Yes:** Checks global consistency and then saves the repository file
- **No:** Does not check global consistency and saves the repository file
- **Cancel:** Does not check global consistency and does not save the repository file

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.

Consistency Check Manager

Displays consistency check messages

- Errors: Must be fixed to make the repository consistent
- Warnings: Conditions that may or may not be errors
- Best Practices: Conditions that do not indicate inconsistencies



Consistency Check Manager

If a repository or object is not consistent, the Consistency Check Manager displays a detailed message that contains information about the nature of the inconsistency. The Consistency Check Manager displays three types of messages:

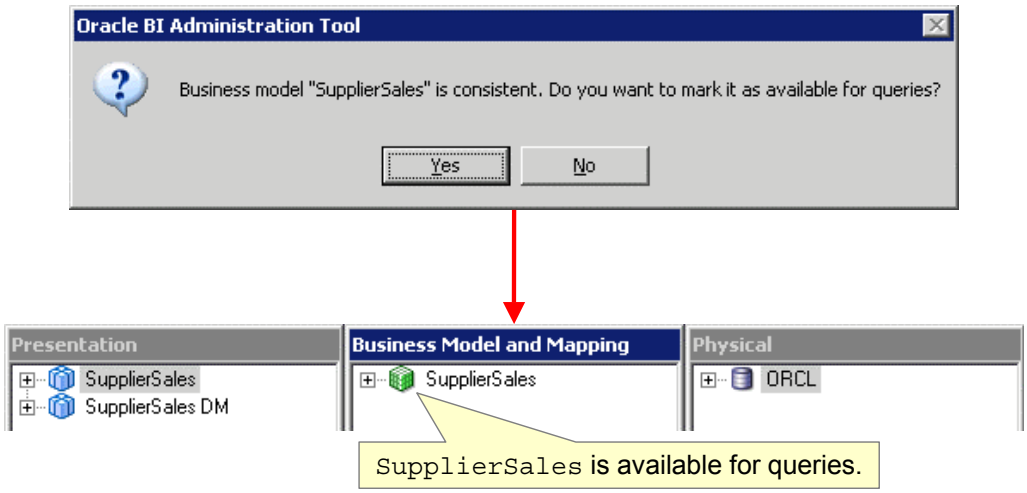
- **Error** messages indicate errors that need to be fixed to make the repository consistent.
- **Warning** messages indicate conditions that may or may not be errors, depending on the intent of the Oracle BI Server administrator. For example, a warning message about a disabled join may be the result of the administrator disabling a join intentionally during repository development.
- **Best Practice** messages provide information about conditions but do not indicate an inconsistency (for example, a physical table without a key).

You can deselect message types if you do not want them displayed in the Consistency Check Manager.

For each message, the Consistency Check Manager identifies the message type, the object type, and the object, and provides an error number and a detailed description of the message. There are options to display only selected message types, display results using qualified names, check all objects in the repository, go to the object in the repository, and copy the results to another file. If the Consistency Check Manager displays any error messages, edit the repository to correct the inconsistencies and run the consistency check again.

Marking a Business Model Available for Queries

When a business model is consistent, you can mark it as available for queries.



Marking a Business Model Available for Queries

If, after a valid consistency check, there are business models marked as unavailable for queries, you receive a message asking if you want to mark the business models as available for queries. When you click Yes, the icon color for the business model changes to green, indicating that it is now available for queries.

Confirming a Consistent Repository

When a repository is consistent, you receive the following message:

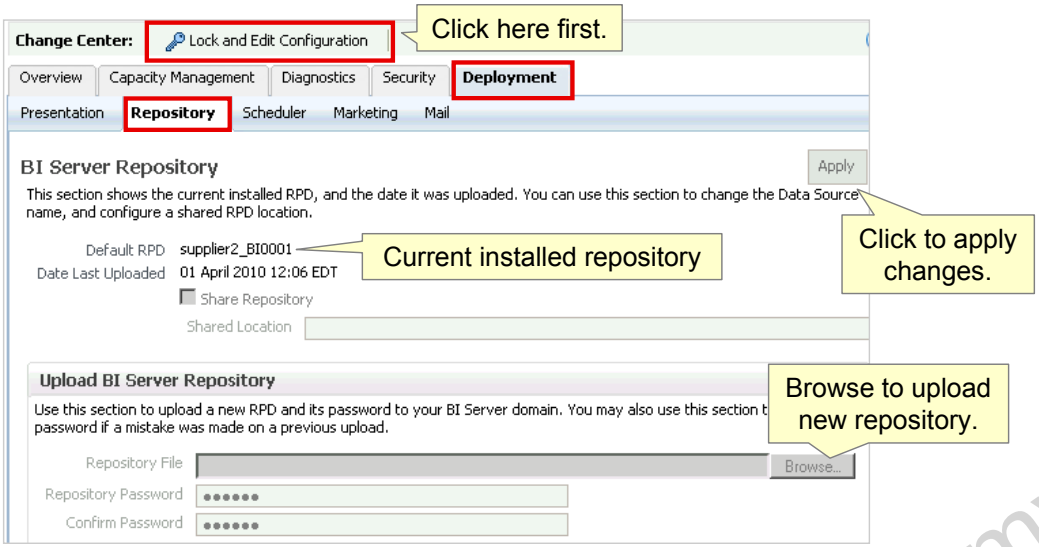
Confirming a Consistent Repository

When a repository is consistent, you receive the following message:

```
Consistency check didn't find any errors, warnings or best practice
violations.
```

Publishing a Repository

Use Fusion Middleware Control to publish a repository and make it available for queries.



Publishing a Repository

After you confirm that a repository is consistent, you use FMW Control to publish the repository. Publishing a repository enables the Oracle BI Server to load the repository into memory at startup and makes the repository available for queries by end users. When you upload a repository file for publishing, you provide the name and location of the repository that you want to upload, as well as the current repository password.

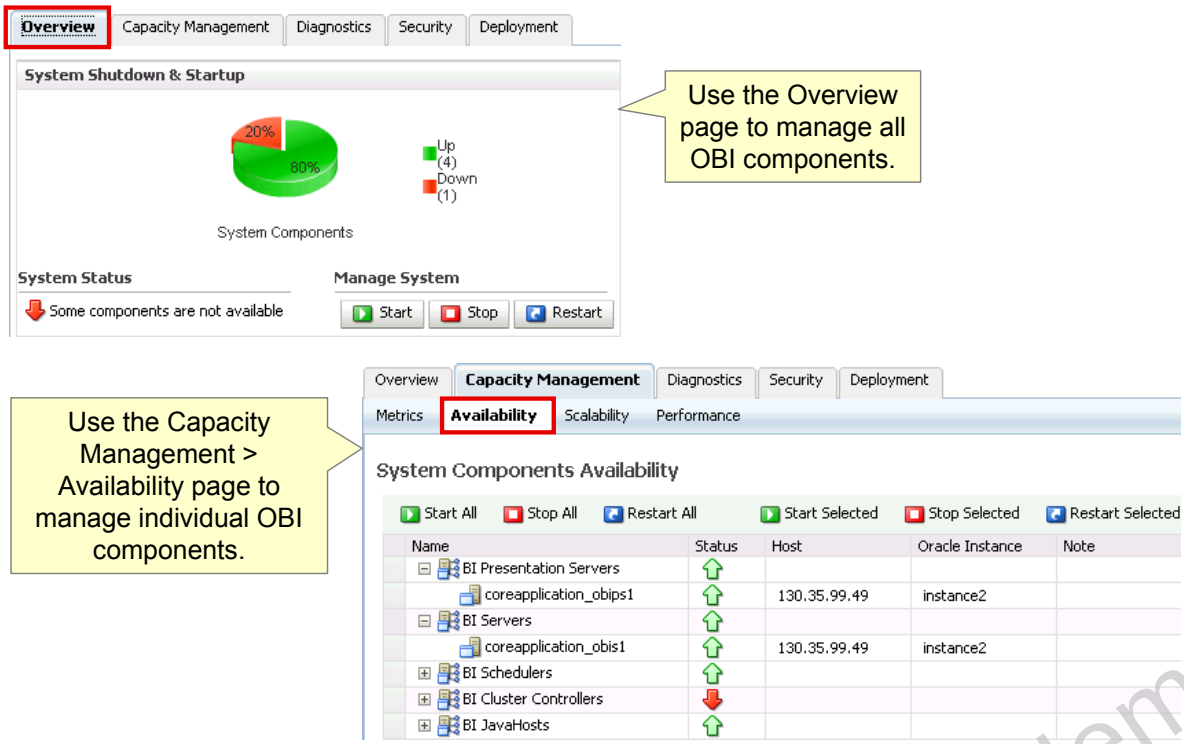
To use FMW Control to upload a repository and set repository parameters, display the Repository tab of the Deployment page. On the Repository tab, you can view the current default repository and the date it was last uploaded.

To upload a new repository, click “Lock and Edit Configuration.” In the Upload BI Server Repository section, click Browse. The Browse dialog box should be set to the default repository directory. If it is not, browse to the following location:

```
ORACLE_INSTANCE\bifoundation\OracleBIServerComponent\coreapplication_obis1\repository
```

Then select the desired repository and click Open. Click “Apply and Activate Changes” (not shown here). You must now restart the Oracle BI Server component, which is discussed in the next slide.

Using FMW Control to Start OBI Components



Using FMW Control to Start OBI Components

After you publish a repository, you must restart the Oracle BI Server component before the changes can take effect. You use FMW Control to manage Oracle BI components.

To start, stop, and restart all Oracle Business Intelligence system components by using FMW Control, go to the Business Intelligence Overview page. Use the buttons in the System Shutdown & Startup region to start, stop, or restart all Oracle Business Intelligence system components.

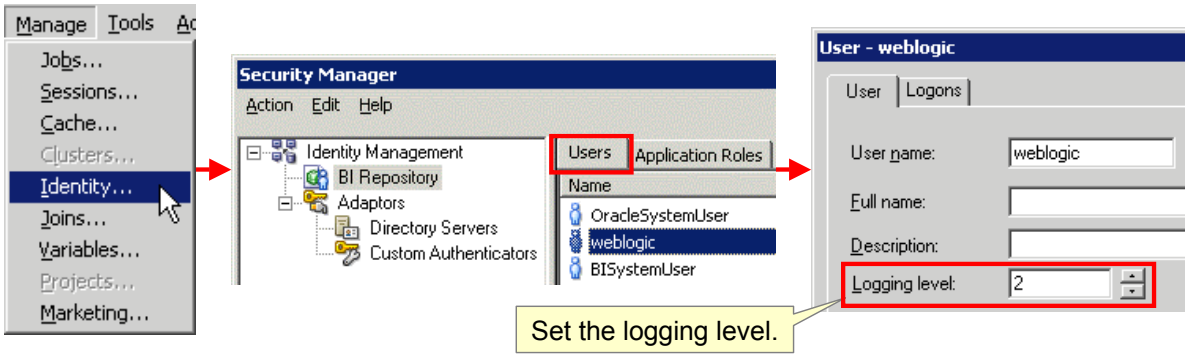
To start, stop, or restart individual Oracle Business Intelligence system components, display the Availability tab on the Capacity Management tab, select a process for a selected server, and click the appropriate button to start, stop, or restart individual processes.

Query Logging

- Oracle BI Server provides a facility for logging query activity at the individual user level.
- Logging is intended for quality assurance testing, debugging, and use by Oracle Technical Support.
- Query logging is normally disabled in production mode.
- The query log file is named `NQQuery.log` and is located in the following directory:
  `ORACLE_INSTANCE\diagnostics\logs\OracleBIServerComponent\coreapplication_obis1`

Setting a Logging Level

Open the repository in online mode and use the Security Manager to enable the logging level for individual users.



Setting a Logging Level

To test the repository, you need to generate some queries, retrieve the results, and examine the query log. Because query logging can produce very large log files, the logging system is turned off by default. It is sometimes useful, however, to enable logging to test that your repository is configured properly, to monitor activity on your system, to help solve performance problems, or to assist Oracle Technical Support.

You must enable logging on the system for each user whose queries you want logged. Use the Security Manager in the Administration Tool to enable a logging level for an individual. You cannot configure logging levels for a group of users. You must open the repository in online mode to set the logging level.

The Security Manager has other functions and is described in more detail in the lesson titled “Security.”

Logging Levels

Logging levels 1 and 2 are for BI Server administrators.

Level 1 Logs	Level 2 Logs
User name, session ID, and request ID for each query	All items for Level 1, plus those mentioned below:
SQL for the request using business model names	Repository name, business model name, subject area name
Query status (success, failure, termination, or timeout)	SQL for the request using physical data source syntax
Elapsed times for query compilation, execution, query cache, and back-end database processing	Queries issued against the cache
	Number of rows returned from a physical database
	Number of rows returned to the client

ORACLE

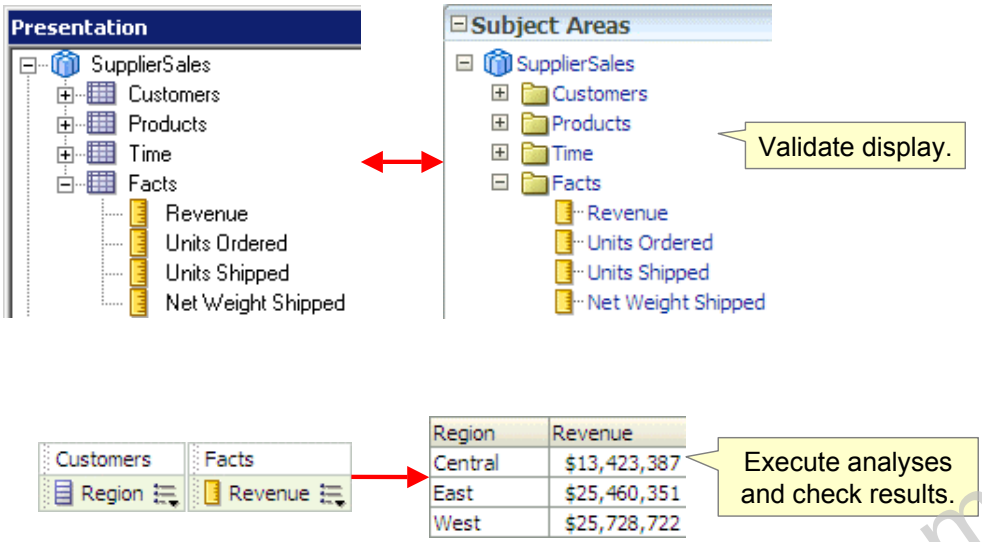
Logging Levels

Set the logging level based on the amount of logging that you want the Oracle BI Server to perform. In normal operations, logging is generally disabled (the logging level is set to 0). If you decide to enable logging, select a logging level of 1 or 2. These two levels are designed for use by Oracle BI Server administrators. Note that logging levels greater than 2 should be used only with the assistance of Oracle Technical Support.

The table shows differences between logging levels 1 and 2. Level 2 provides more detailed information than level 1. Refer to the *Oracle Fusion Middleware System Administrator's Guide for Oracle Business Intelligence Enterprise Edition* for a complete description of logging levels.

Validating by Using the Analysis Editor

Use the Analysis Editor to validate the Presentation layer display, execute analyses, and check query results.

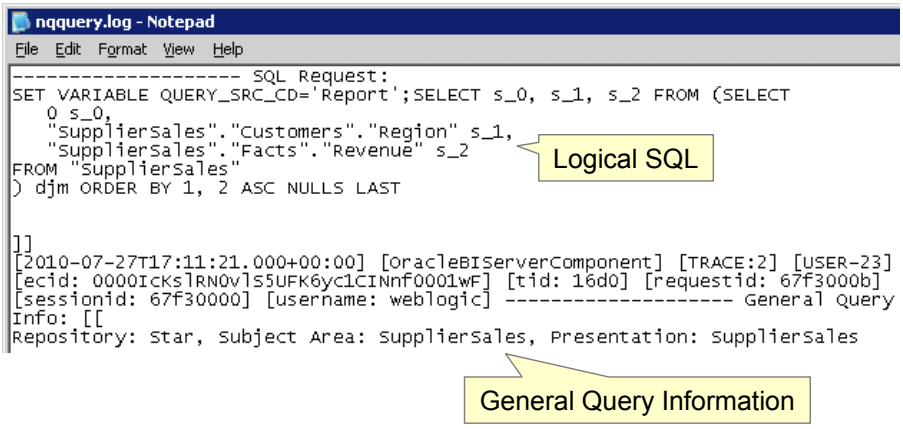


Validating by Using the Analysis Editor

Use the Analysis Editor to validate the display of subject areas, tables, and columns based on the Presentation layer. You can also create and execute analyses and check results.

Inspecting the Query Log

Use the query log to check query results:



Inspecting the Query Log

The query log contains the results of executed queries. The `NQQuery.log` file is located in the following directory:

`ORACLE_INSTANCE\diagnostics\logs\OracleBIServerComponent\coreapplication_obis1`

You can open the file directly using a program such as Notepad. You can also view log entries on the Diagnostics tab in Fusion Middleware Control Enterprise Manager, or by navigating to Administration > Manage Sessions > View Log in the OBI user interface. The log file has the following sections:

- The SQL Request section lists the SQL issued from the client application.
- The General Query Info section lists the repository, the business model, and the presentation catalog from which the query was run.
- The Database Query section records the SQL sent to the underlying databases.
- The Query Status section indicates if the query completed successfully or failed.

Log entries for levels 1 and 2 are generally self-explanatory. The query log can help you check the accuracy of applications that use Oracle BI Server. For example, use the log file to confirm that queries access the correct repository, subject areas, tables, columns, and so on.

Oracle BI SELECT Statement: Syntax

Basic syntax for an Oracle BI SELECT statement:

SELECT	Columns
FROM	Catalog folder or tables
WHERE	Filtering conditions
GROUP BY	(Optional, special use)
ORDER BY	Columns (optional)

Example

```
SELECT Region, Dollars
FROM SupplierSales
WHERE Year = 1999
ORDER BY Dollars desc
```


Oracle BI SELECT Statement: Syntax

The slide identifies the basic syntax for an Oracle BI SELECT statement. The SELECT statement is the basis for querying any SQL database. 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 (or *query specification*, as it is sometimes called) is the way to query a decision support system through Oracle BI Server. A SELECT statement returns a table to the client that matches the query. It is a table because the results are in the form of rows and columns.

Table names can be included in the SELECT list (as Table.Column) but are optional unless column names are not unique within a business model. You can specify the name of a catalog folder instead of a list of tables to simplify creation of the FROM clause. The next slide identifies the differences between standard SQL SELECT statements and Oracle BI SELECT statements.

Oracle BI SELECT Statement Compared with Standard SQL

Oracle BI SELECT statements differ from standard SQL in the following ways:

- No join information is required.
  - Join conditions are predefined in the repository.
  - Join conditions supplied in a query are ignored.
- No aggregation functions are required.
  - Aggregation rules are known to the server, and aggregation is performed automatically.
- No GROUP BY clause is required.
  - If aggregated data is requested in the SELECT statement, a GROUP BY clause is automatically assumed by the server.
- No DISTINCT keyword is required.
  - Oracle BI Server issues SELECT DISTINCT to eliminate duplicate rows automatically.

Oracle BI SELECT Statement Compared with Standard SQL

The SELECT statement is similar to standard SQL, but there are differences.

Join specifications: Because Oracle BI Server uses its metadata to determine the physical tables and the necessary join specifications, no join information is required. It is ignored if it is included.

Aggregation functions: Because the aggregation rules have also been defined in the metadata, no aggregation functions are required [for example, `sum(ColumnA)`].

GROUP BY clause: If aggregated data is requested in the SELECT statement, a GROUP BY clause is automatically assumed by the server. When no GROUP BY clause is specified, the GROUP BY specification defaults to all 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 Oracle BI Server computes the results based on the level specified in the GROUP BY clause.

DISTINCT keyword: Oracle BI Server always issues SELECT DISTINCT to eliminate duplicate rows automatically. Thus, if you want to see “Dollars by Region” for 1999, you can issue the following SELECT statement:

```
SELECT Region, Dollars FROM SupplierSales WHERE Year = 1999
```

Summary

In this lesson, you should have learned how to execute the steps to test and validate a repository.

Practice 6-1 Overview: Testing a Repository

This practice covers the following topics:

- Checking a repository for consistency
- Enabling logging
- Loading a repository
- Validating a repository by using the OBI user interface
- Inspecting the query log

Practice 6-1 Overview: Testing a Repository

You have finished building the initial business model and now need to test the repository before publishing it to users. You begin by using the check consistency option to check the repository for errors. You then enable logging and test the repository by running queries. Finally, you examine the query log file to validate the SQL generated by Oracle BI Server.

7

Managing Logical Table Sources

Objectives

After completing this lesson, you should be able to:

- Describe normalized and denormalized table structures in database designs
- Add multiple sources to a logical table source for a dimension in the business model
- Add a second logical table source to a dimension in the business model

Table Structures

- Normalized table structure:
  - Consists of many tables where data has been split or normalized
  - Is used for inserts and updates
  - Does not work well for queries that perform business data analysis
- Denormalized table structure:
  - Follows a business model and is easier to understand
  - Has data that may be duplicated in several locations in a database
  - Can take the form of a star schema
  - Provides better query performance

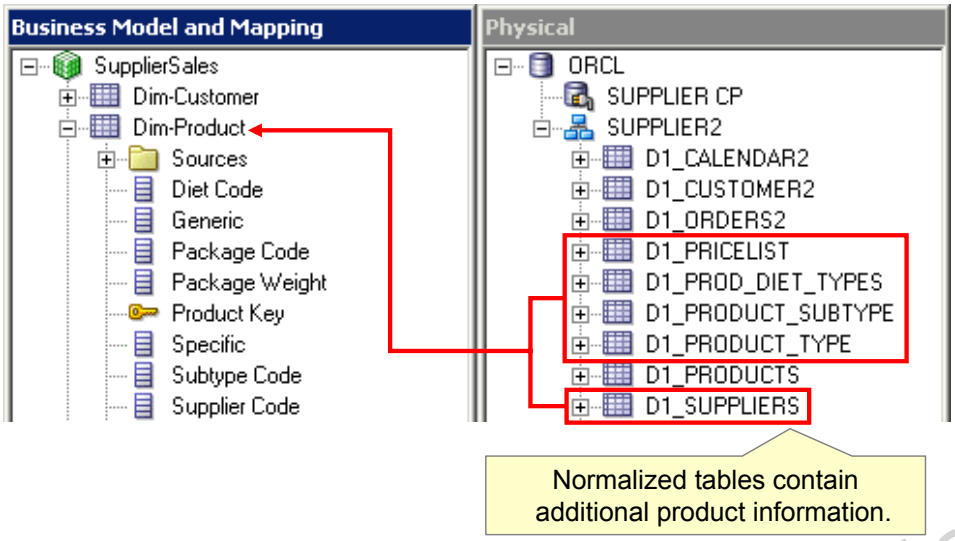
Table Structures

Normalized schemas are used in many online transaction processing (OLTP) systems. They have many tables (hundreds or even thousands) because the data has been carefully taken apart—*normalized*, in database terminology—with the primary goal of reducing data redundancy and bringing about fast update performance. These schemas generally do not work well for queries that perform historical analysis due to two major problems: poor performance and difficulty in posing the question in SQL.

A dimensional schema is a denormalized schema that follows a business model. This structure contains dimension tables (which contain attributes of the business) and fact tables (which contain individual records with facts and foreign keys to each of the dimension tables). Dimensional schemas work well for business analysis and have two major advantages: They have better query performance and are easier to understand.

Business Challenge

Data may be spread across several physical tables and needs to be mapped to a single logical table.



Business Challenge

This slide presents the business challenge: adding information to a logical dimension table when the additional information is spread across many physical tables with a normalized table structure. The challenge is to map these disparate physical tables to a single logical table regardless of whether the data is normalized or duplicated. In this example, normalized tables contain additional product information and must be mapped to the `Dim-Product` logical dimension table.

Business Solution

Model multiple physical sources for the logical table:

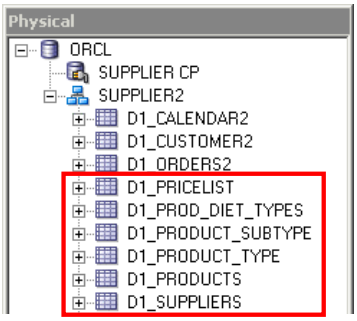
- Add multiple sources to a logical table source.
  - Where data is not duplicated across tables
- Add a new logical table source.
  - Where data is duplicated across tables

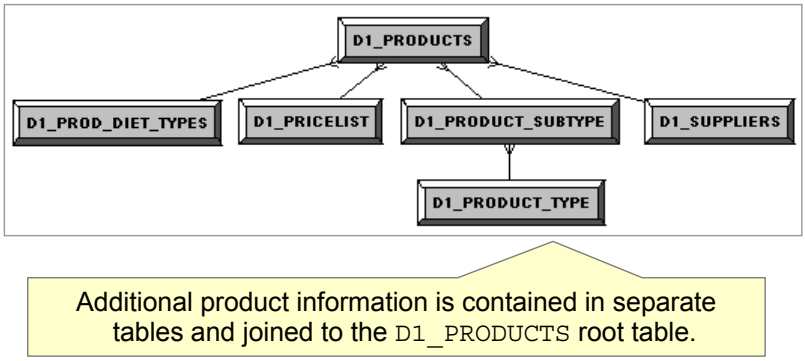
Business Solution

The solution depends on how data is stored. Where data is not duplicated across physical tables, you model multiple physical sources for a dimension table by adding physical tables to a logical table source (LTS). Where data is duplicated in the physical sources, you add a second LTS to configure the most economical source.

ABC Example: Adding Multiple Sources to a Logical Table Source (LTS)

Add normalized tables that store product code, product type, pricing, and supplier information to the LTS for the `Product` dimension in the `SupplierSales` business model.





Additional product information is contained in separate tables and joined to the `D1_PRODUCTS` root table.

ABC Example: Adding Multiple Sources to a Logical Table Source (LTS)

In the ABC example, all product information is stored in multiple normalized physical tables. `D1_PRODUCTS` is the root table, and additional product information is stored in multiple detail tables that snowflake off the root table.

Up to this point in the ABC example, the logical columns for the `Dim-Product` logical table have mapped only to the alias of the `D1_PRODUCTS` root product table. Now you want to include additional information from the other product tables that snowflake off the root table. Because the data is not duplicated across the physical tables, you add multiple physical sources to an existing LTS for the `Products` logical dimension table.

Implementation Steps: Adding Multiple Sources to an LTS

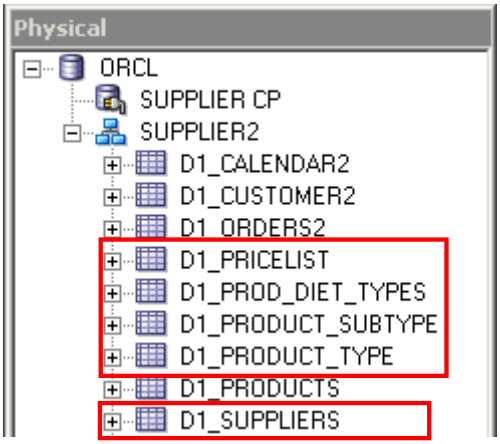
1. Import additional product tables.
2. Create aliases.
3. Define keys and joins.
4. Identify physical columns for mappings.
5. Add sources to an LTS:
   - Manual method
   - Drag method
6. Rename logical columns.
7. Add columns to the Presentation layer.

Implementation Steps: Adding Multiple Sources to an LTS

This slide lists the implementation steps for adding multiple tables and columns to an LTS where data is not duplicated across tables. Later in this lesson, you learn how to add a new LTS to a logical dimension table to handle duplicate data.

1. Import Additional Product Tables

Into the Physical layer, import additional product tables that store product code, product type, pricing, and supplier information.



ORACLE

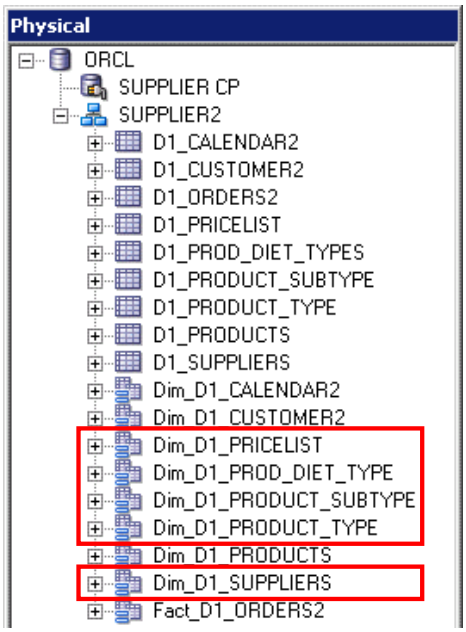
1. Import Additional Product Tables

The first step is to import additional product tables that store product code, product type, pricing, and supplier information. The product dimension is an example of where information has been stored physically in a normalized table structure.

Writers such as Ralph Kimball sometimes call this structure “snowflaking a dimension.” Many database administrators (DBAs) regard this as good database design, so it is a very common practice. So far, you have included only the root product table information in the logical subject area. After import, you can include information from the other product tables.

2. Create Aliases

Create alias tables with desired naming conventions.



ORACLE

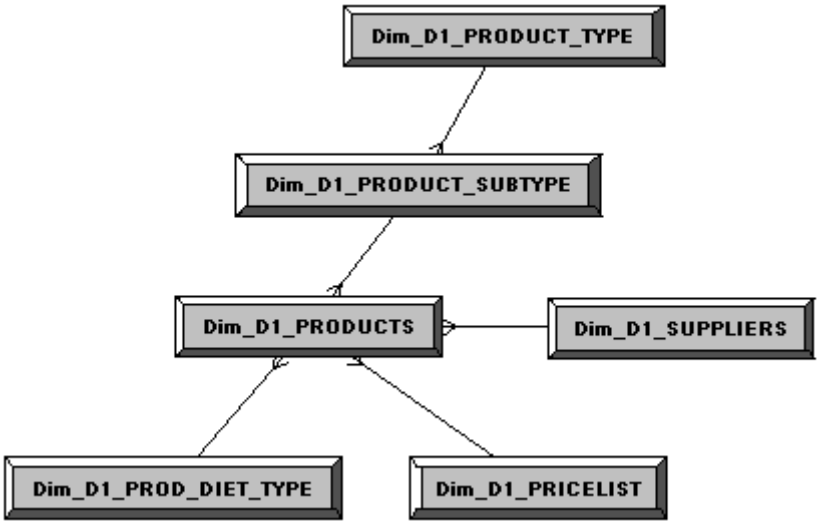
2. Create Aliases

Creating alias tables is not a required step. However, it is recommended that you use table aliases frequently in the Physical layer to eliminate extraneous joins and to include best-practice naming conventions for physical table names.

To create an alias, right-click the desired source table and select New Object > Alias. Provide a name for the alias. The alias is automatically added to the Physical layer and shares all of the attributes of its physical source table.

3. Define Keys and Joins

Define the keys and joins for the product tables.



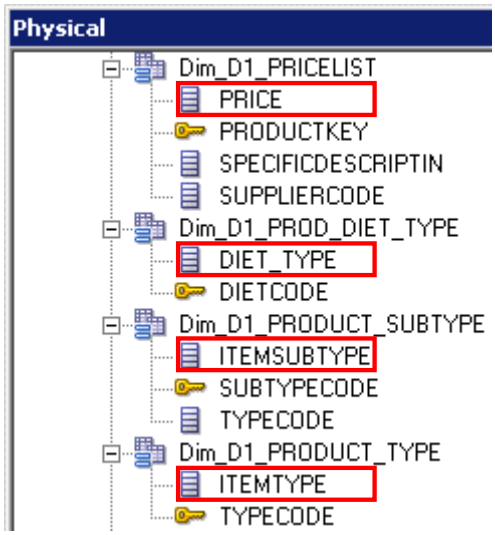
ORACLE

3. Define Keys and Joins

Use known techniques to define the keys and joins for the aliases of the newly imported product tables in the Physical layer.

4. Identify Physical Columns for Mappings

Locate the columns in the Physical layer with the additional product information.



4. Identify Physical Columns for Mappings

After importing the table schemas into the Physical layer and configuring join relationships, locate the columns that store the additional product information that you want to add to the `Product` logical table.

5. Adding Sources to an LTS

You can add sources to an existing LTS by the following methods:

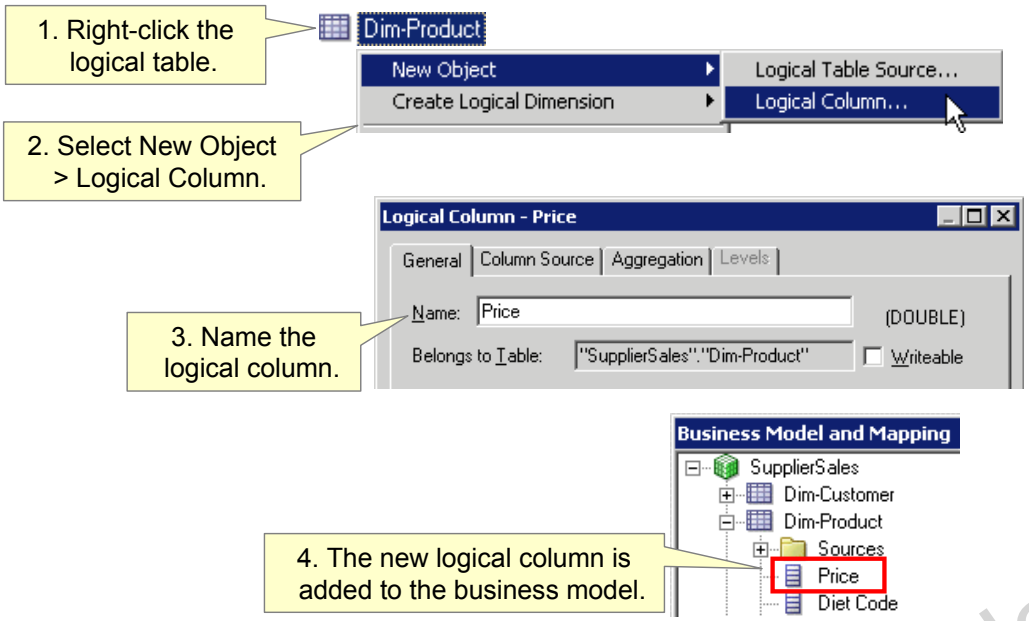
- Manual: Use the Properties dialog box of an LTS to map tables and columns.
- Drag: Drag columns from the Physical layer to an LTS to automatically map tables and columns.

5. Adding Sources to an LTS

Both methods—manual and drag—for adding sources to an LTS are described in the following slides.

5a. Manual Method: Create New Logical Column

Create a new logical column for a logical dimension table.



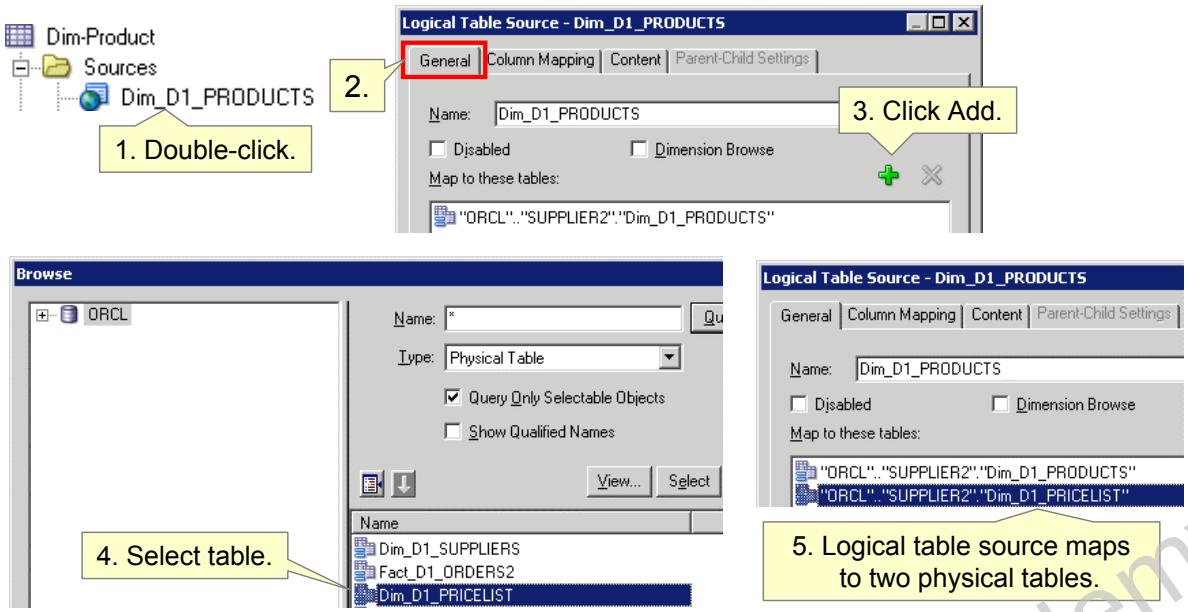
5a. Manual Method: Create New Logical Column

Using the manual method, the first step is to manually create a new logical column. In this example, you add a new `Price` logical column to the `Dim-Product` logical dimension table.

1. Right-click the `Dim-Product` logical dimension table.
2. Select New Object > Logical Column.
3. Enter `Price` in the Name field in the Logical Column dialog box.
4. The new `Price` logical column is added to the `Dim-Product` logical dimension table in the Business Model and Mapping (BMM) layer.

5a. Manual Method: Add New Physical Source

Use the Properties dialog box of an LTS to add a new physical table to the source.



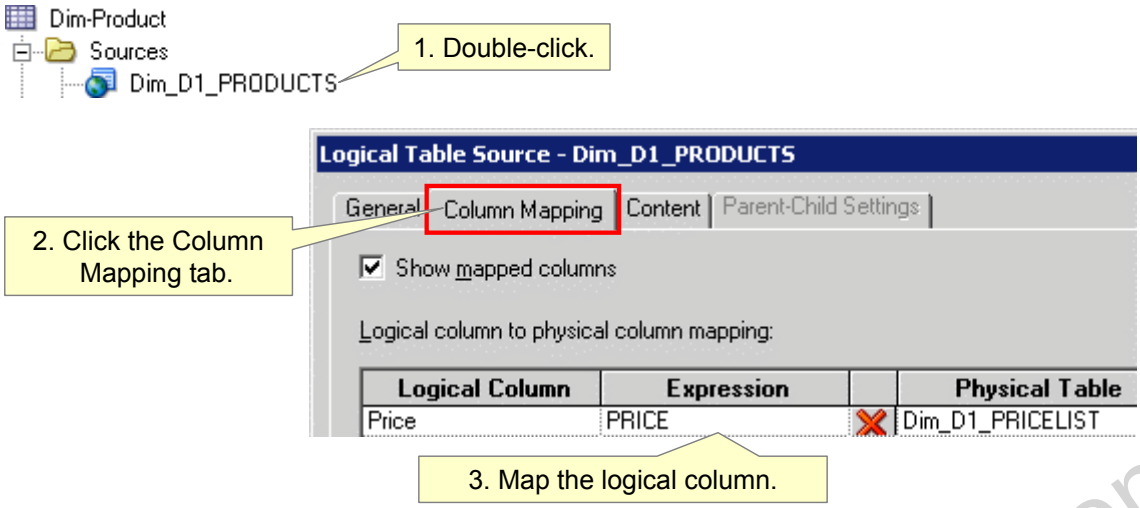
5a. Manual Method: Add New Physical Source

After the logical column is created, use the Properties dialog box of an LTS to add a new physical table to that LTS. In this example, you add the `Dim_D1_PRICELIST` physical table to the `Dim_D1_PRODUCTS` LTS.

1. Double-click the `Dim_D1_PRODUCTS` LTS to open the Logical Table Source dialog box.
2. Click the General tab.
3. Click the Add button.
   The Browse window opens and automatically displays those tables that are joined directly to the table that is already in the LTS. Only tables that join to tables included in the logical source can be added to the logical source.
4. Select the `Dim_D1_PRICELIST` physical table. You can double-click the table or select it and click the Select button.
5. The `Dim_D1_PRODUCTS` LTS now maps to two physical tables.

5a. Manual Method: Create Column Mapping

Use the Properties dialog box of an LTS to map the new logical column to a physical column in the new source.



5a. Manual Method: Create Column Mapping

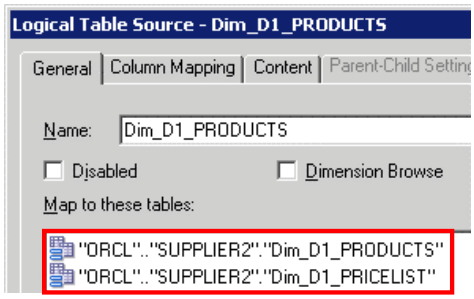
After creating the logical column and adding a new physical source to the LTS, you need to map the logical column to the appropriate physical column. In this example, you map the `Price` logical column to the `PRICE` physical column in the `Dim_D1_PRICELIST` physical table.

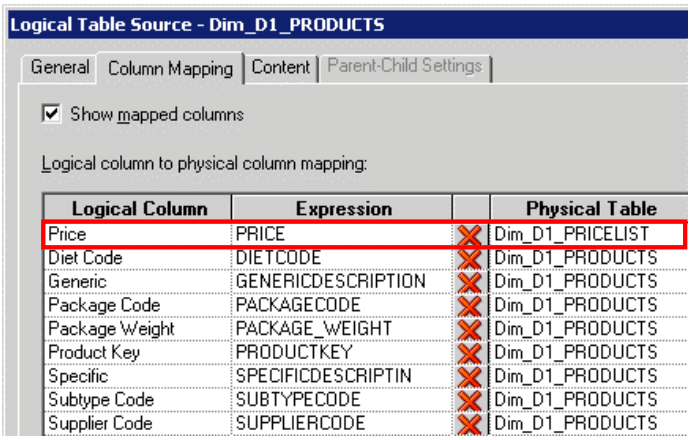
1. Double-click the `Dim_D1_PRODUCTS` LTS to open the Logical Table Source dialog box.
2. Click the Column Mapping tab.
3. Use the drop-down list in the `Expression` column to map the `Price` logical column to the `PRICE` physical column in the `Dim_D1_PRICELIST` physical table.

5a. Manual Method: End Result

`Dim_D1_PRODUCTS` LTS now maps to two physical tables: `Dim_D1_PRODUCTS` and `Dim_D1_PRICELIST`.

`Price` logical column maps to the `PRICE` physical column in the `Dim_D1_PRICELIST` physical table.





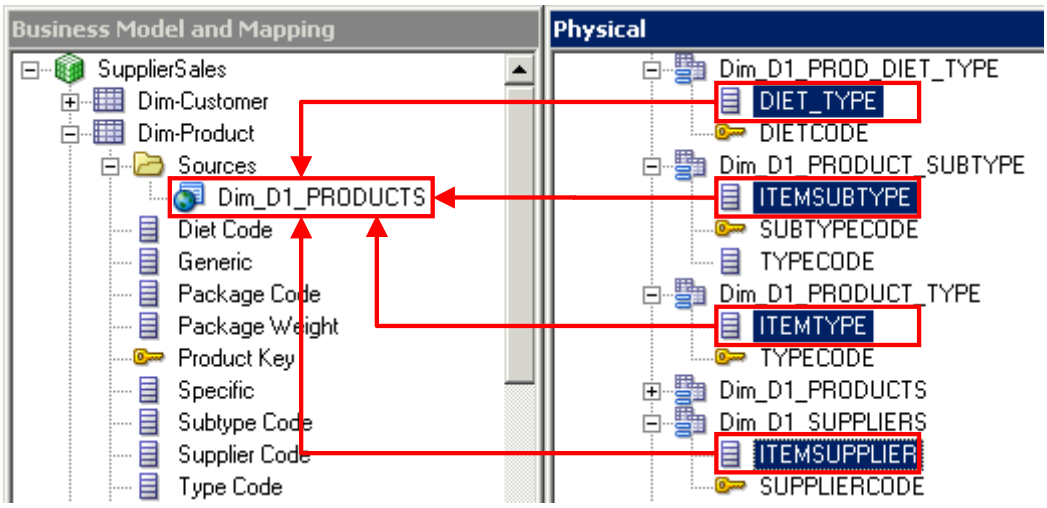
5a. Manual Method: End Result

The end result of the manual process is that the LTS maps to multiple physical tables and the new logical column maps to the appropriate physical column.

In this example, the `Dim_D1_PRODUCTS` LTS now maps to two physical tables: `Dim_D1_PRODUCTS` and `Dim_D1_PRICELIST`. The `Price` logical column maps to the `PRICE` physical column in the `Dim_D1_PRICELIST` physical table.

5b. Drag Method

Drag physical columns to the logical table source.



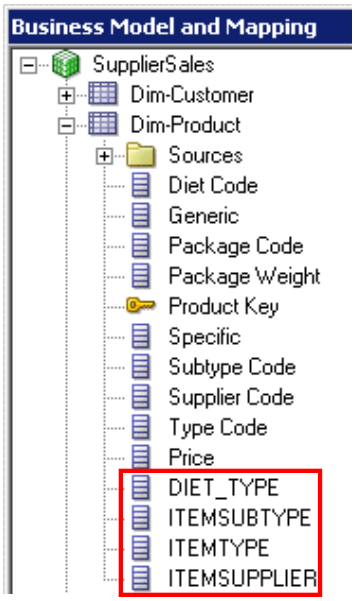
5b. Drag Method

This is a more efficient method for creating multiple sources for an LTS. The first step is to drag the desired columns from the Physical layer to the existing LTS in the BMM layer.

In this example, you drag the `DIET_TYPE`, `ITEMSUBTYPE`, `ITEMTYPE`, and `ITEMSUPPLIER` physical columns to the `Dim_D1_PRODUCTS` LTS for the `Dim-Product` logical dimension table. Note that the columns are dragged from multiple physical tables.

5b. Drag Method: Logical Columns Added

Dragging physical columns to an LTS automatically creates logical columns in the BMM layer.

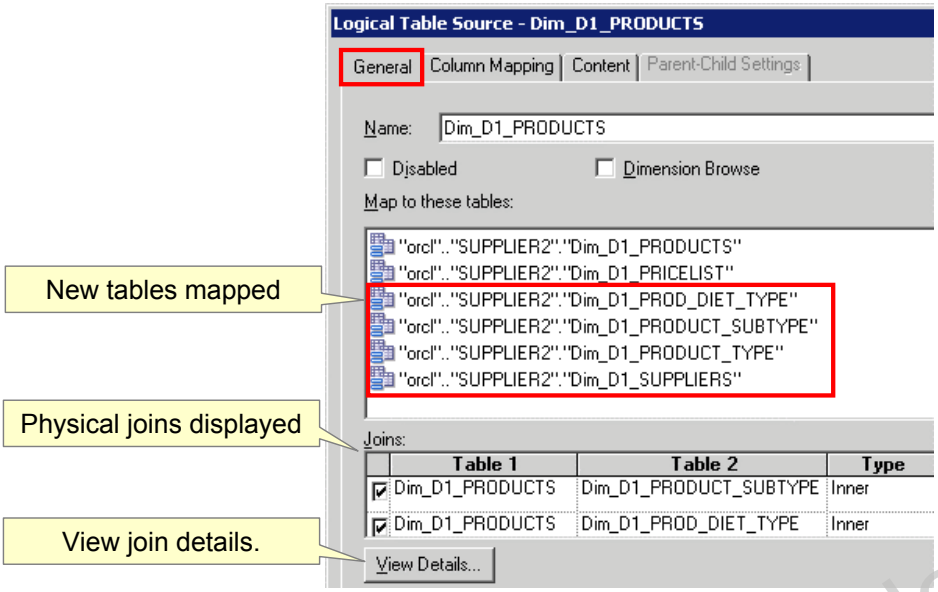


5b. Drag Method: Logical Columns Added

In this example, there are four new logical columns for the `Product` logical dimension table: `DIET_TYPE`, `ITEMSUBTYPE`, `ITEMTYPE`, and `ITEMSUPPLIER`.

5b. Drag Method: Physical Tables Added

Dragging physical columns to an LTS automatically adds physical table mappings to the LTS.



5b. Drag Method: Physical Tables Added

Dragging physical columns to an LTS also automatically adds the corresponding physical table mappings to the LTS.

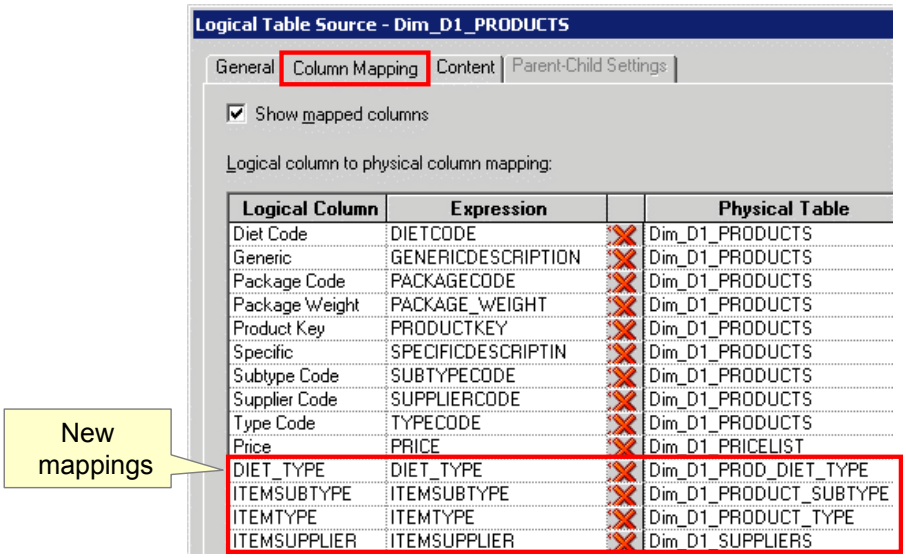
In this example, the `Dim_D1_PRODUCTS` LTS now maps to the following new tables:

- `Dim_D1_PRODUCT_SUBTYPE`
- `Dim_D1_PRODUCT_TYPE`
- `Dim_D1_PROD_DIET_TYPE`
- `Dim_D1_SUPPLIERS`

The physical joins are also displayed on the General tab, although the screenshot displays only two of them. Select a join and click View Details to open the Joins dialog box and view the join properties.

5b. Drag Method: Column Mappings Added

Dragging physical columns to an LTS automatically adds column mappings to the LTS.



5b. Drag Method: Column Mappings Added

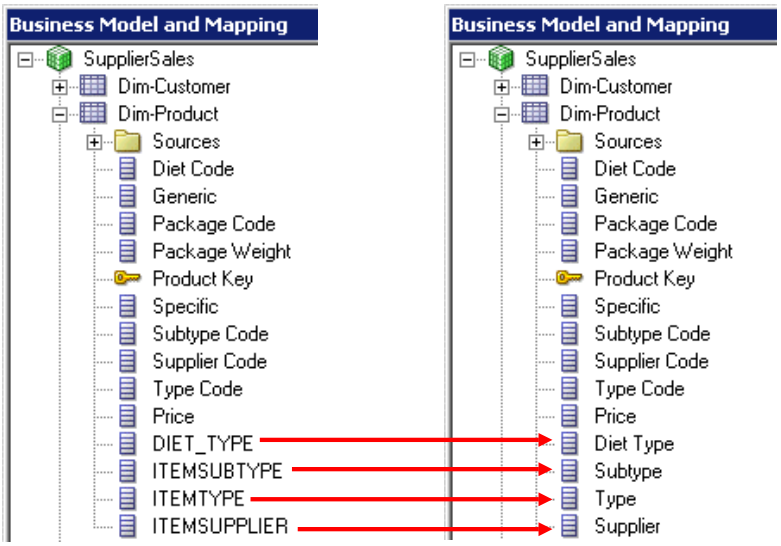
Dragging physical columns to an LTS also automatically adds column mappings to the LTS.

In this example, the following mappings have been added:

- `DIET_TYPE` logical column maps to `Dim_D1_PROD_DIET_TYPE.DIET_TYPE`.
- `ITEMSUBTYPE` logical column maps to `Dim_D1_PRODUCT_SUBTYPE.ITEMSUBTYPE`.
- `ITEMTYPE` logical column maps to `Dim_D1_PRODUCT_TYPE.ITEMTYPE`.
- `ITEMSUPPLIER` logical column maps to `Dim_D1_SUPPLIERS.ITEMSUPPLIER`.

6. Rename Logical Columns

Rename new logical columns with names that are meaningful to users.

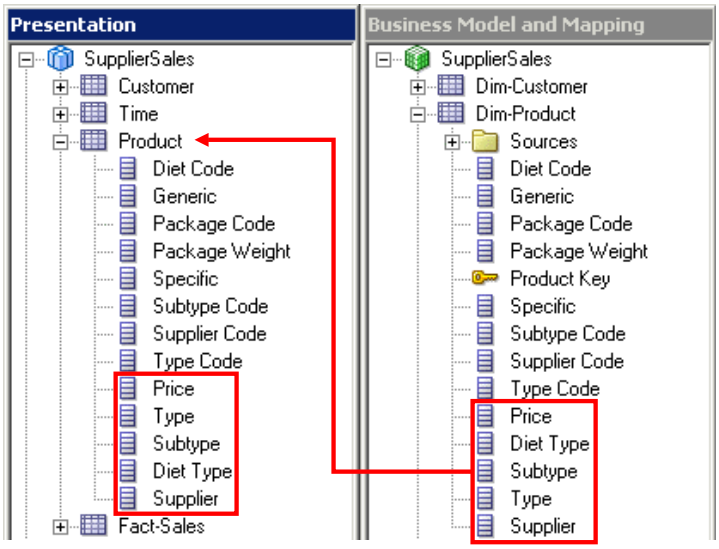


6. Rename Logical Columns

Whichever method you use to add sources to an LTS, you should rename logical columns with names that are meaningful to users.

7. Add Columns to the Presentation Layer

Drag the new logical columns to the Presentation layer to make them visible and available to users.

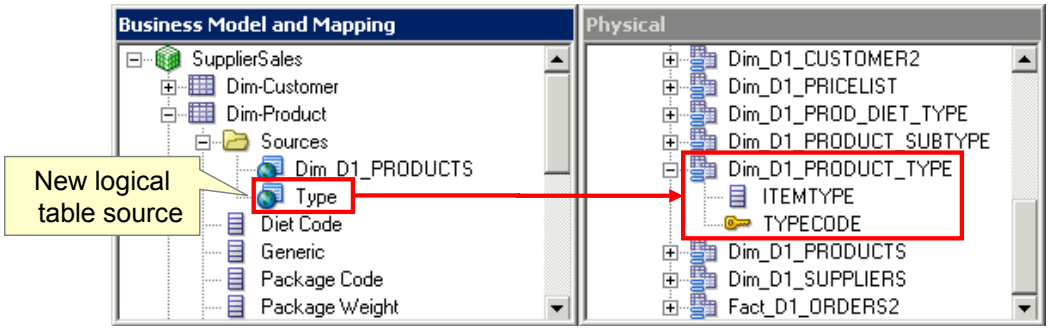


7. Add Columns to the Presentation Layer

In this example, the columns are added to the `Product` presentation table in the Presentation layer.

ABC Example:
Adding a New Logical Table Source

Add a second LTS to the `Product` dimension so that Oracle BI Server queries only one table for Type and Type Code information.



ABC Example: Adding a New Logical Table Source

Where data is duplicated in the physical sources, you need to add a second LTS so that Oracle BI Server can select the most economical source.

In this example, Type and Type Code data are duplicated across multiple physical sources. In examining the column mappings for the `Dim_D1_PRODUCTS` LTS for the `Products` dimension table, you discover that the `Type` and `Type Code` columns are mapped to different physical tables, but the information for both is stored in a common `Dim_D1_PRODUCT_TYPE` physical table.

To model the most economical method for Oracle BI Server to execute queries for these two columns, you add a second LTS to the `Dim-Product` dimension so that Oracle BI Server queries only one table for the Type and Type Code information.

Implementation Steps: Adding a New Logical Table Source

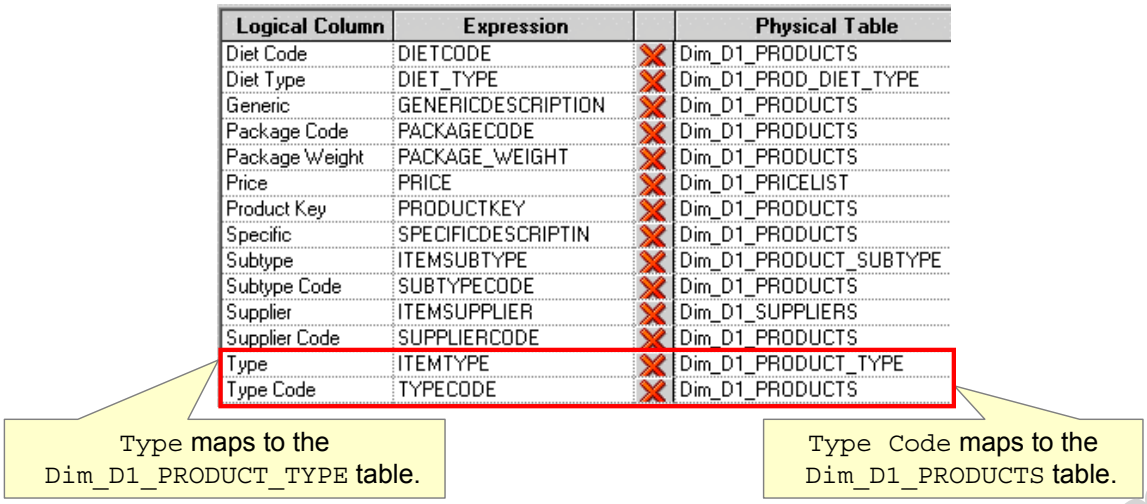
1. Examine the existing column mappings.
2. Identify a single table that stores both columns.
3. Add a new logical table source.
4. Define the content of the logical table source.
5. Verify your work.

Implementation Steps: Adding a New Logical Table Source

This slides lists the steps for adding a new LTS where data is duplicated across tables. Each step is discussed in detail in the following slides.

1. Examine Existing Column Mappings

Examine column mappings for `Type` and `Type Code` columns in the `Dim_D1_PRODUCTS` LTS.

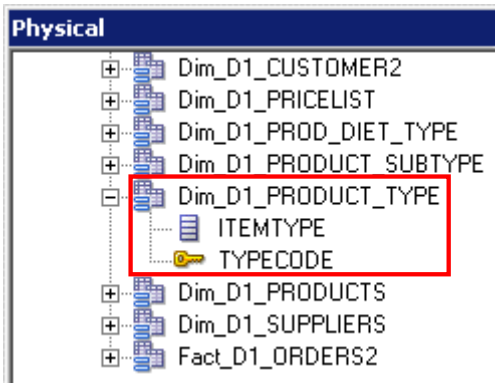


1. Examine Existing Column Mappings

Dragging columns to the `Dim_D1_PRODUCTS` LTS created new column mappings. You examine the mappings and find that the `Type` column maps to `ITEMTYPE` in the `Dim_D1_PRODUCT_TYPE` table and the `Type Code` column maps to `TYPECODE` in the `Dim_D1_PRODUCTS` table.

2. Identify a Single Table That Stores Both Columns

Examine the `Dim_D1_PRODUCT_TYPE` table and see that it contains both `ITEMTYPE` and `TYPECODE`.

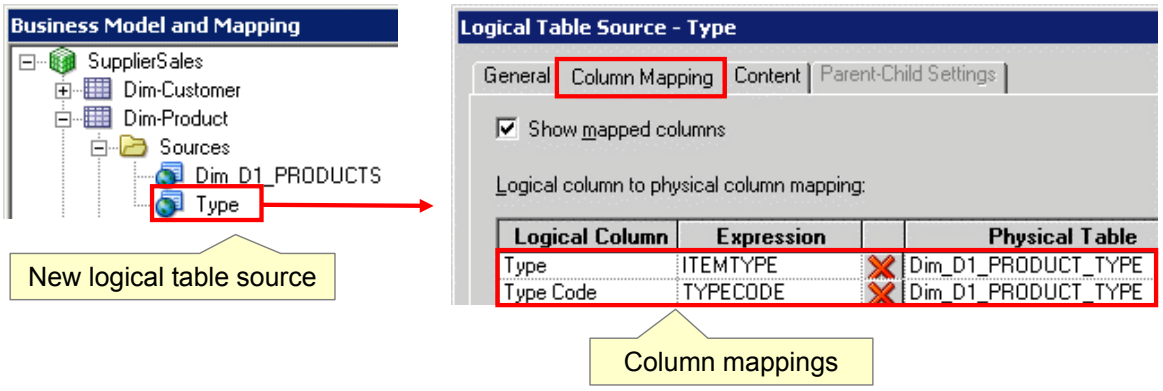


2. Identify a Single Table That Stores Both Columns

`Dim_D1_PRODUCT_TYPE` contains both columns, `ITEMTYPE` and `TYPECODE`. Thus `TYPECODE` is duplicated across the `Dim_D1_PRODUCTS` and `Dim_D1_PRODUCT_TYPE` tables.

3. Add a New Logical Table Source

Add a new LTS for the `Products` logical dimension table that maps to the columns in `D1_PRODUCT_TYPE`.

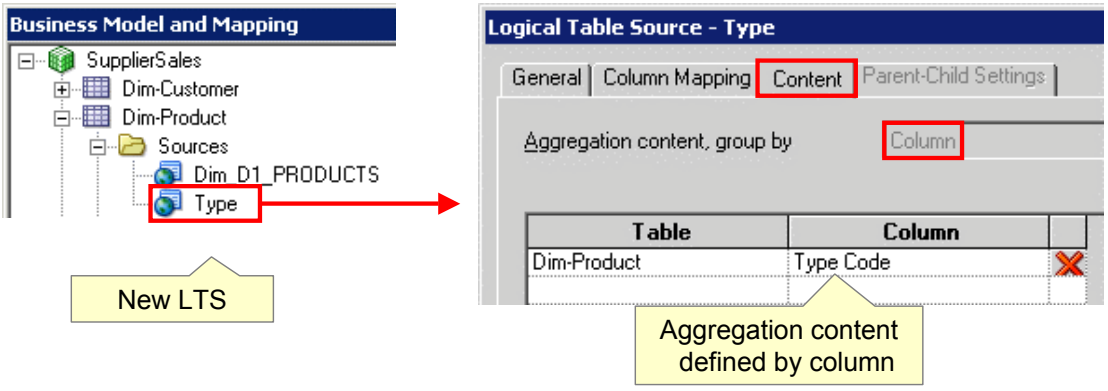


3. Add a New Logical Table Source

Use the manual or drag method to add a new LTS named `Type` to the `Dim-Product` logical dimension table. Map both the `Type` and `Type Code` logical columns to the corresponding physical columns in `Dim_D1_PRODUCT_TYPE`.

4. Define the Content of the Logical Table Source

Use the Content tab in the Logical Table Source dialog box to define content for the source.



4. Defining the Content of the Logical Table Source

There are now two logical table sources for the `Dim-Product` logical table: `Dim_D1_PRODUCTS` and `Type`. The `Type Code` logical column is mapped in both logical table sources and, therefore, maps to both the `Dim_D1_PRODUCT_TYPE` and the `Dim_D1_PRODUCTS` tables.

Now you need to specify the aggregation content for the logical table sources. To use a source correctly, 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 LTS.

Note that for "Aggregation content, group by," `Column` is selected and grayed out. Although you have the option to specify aggregation content by column or logical level, it is recommended that you use logical levels exclusively. However, in this example you have not yet created any logical levels, so `Column` is the only available option. Later, in the lesson titled "Using Aggregates," you learn how to define content by using logical levels.

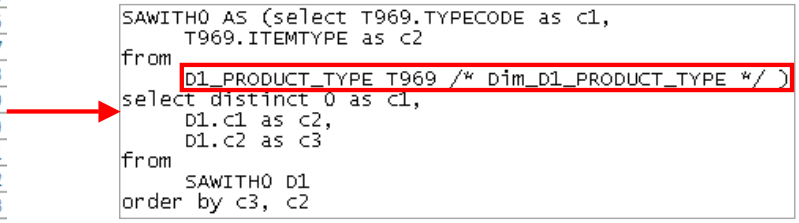
Now when a query is executed that includes the `Type` and `Type Code` logical columns, Oracle BI Server accesses only one table, `Dim_D1_PRODUCT_TYPE`, via the `Type` LTS. Because `Dim_D1_PRODUCT_TYPE` stores information for both columns, it is the more economical source.

Note that you do not have to modify the Presentation layer.

5. Verify Your Work

Execute a query and examine the query log to confirm that the expected table is accessed.

Type	Type Code
Baking	100
Beef	101
Beverage	102
Bread	103
Cereal	104
Cheese	105
Condiments	106
Dessert	107
Entre	108
Frozen	109
Grains	110
Lamb	111
Non-food	112
Pasta	113
Pork	114
Poultry	115
Rice	116
Seafood	117
Snacks	118
Soup	119
Vegetable	120



5. Verify Your Work

Execute a query and examine the query log to determine which tables have been accessed. In this example, you verify that only one table, `Dim_D1_PRODUCT_TYPE`, is accessed by the query. Because you defined the aggregation content in the logical table sources for the `Dim-Product` logical table, Oracle BI Server executes its query against the most “economical” source, which in this example is the `Dim_D1_PRODUCT_TYPE` table. BI Server bypasses the physical joins and accesses the table directly.

Summary

In this lesson, you should have learned how to:

- Describe normalized and denormalized table structures in database designs
- Add multiple sources to a logical table source for a dimension in the business model
- Add a second logical table source to a dimension in the business model

Practice 7-1 Overview: Enhancing the Product Dimension

This practice covers the following topics:

- Importing normalized tables that contain additional product information into the Physical layer of the repository
- Defining physical keys and joins

Practice 7-1 Overview: Enhancing the Product Dimension

There are product tables that store details about ABC's products. You want to add these tables to the Product dimension in the BMM layer. You import these tables into the repository and create keys and foreign key joins for the tables.

Practice 7-2 Overview: Creating Multiple Sources for a Logical Table Source (Manual)

This practice covers using the manual method to add a source to a logical table source.

Practice 7-2 Overview: Creating Multiple Sources for a Logical Table Source (Manual)

You have imported the product tables that store detailed information about ABC's products into the Physical layer of the repository, and you have configured keys and joins for the tables. So far, the `Dim-Product` logical table in the BMM layer contains only information from the root product table, `Dim_D1_PRODUCTS`. You now add the information from the price list table to the `Dim-Product` logical table. This simplifies the data structure, creating in effect a denormalized logical table.

Practice 7-3 Overview: Creating Multiple Sources for a Logical Table Source (Automated)

This practice covers using the drag method to add multiple sources to a logical table source.

Practice 7-3 Overview: Creating Multiple Sources for a Logical Table Source (Automated)

You have manually added information from the price list table to the `Dim-Product` logical table. Now you add information from the other product tables to the `Dim-Product` logical table by using a more automated method. This automated method adds multiple physical tables to the existing LTS for the `Dim-Product` logical table and simultaneously adds logical columns to the `Dim-Product` logical table.

Practice 7-4 Overview: Adding a New Logical Table Source

This practice covers the following topics:

- Adding a second logical table source to a logical dimension table
- Defining the content for a logical table source

Practice 7-4 Overview: Adding a New Logical Table Source

You examine the physical sources for the `Dim-Product` logical table and discover that the `Type Code` and `Type` columns are mapped to different physical tables. You also discover that the information for both columns is stored in a common physical table, `D1_PRODUCT_TYPE`.

To model the most economical method for Oracle BI Server to generate queries against these two columns, you add a second LTS to the `Dim-Product` logical table so that Oracle BI Server queries only one table for the `Type Code` and `Type` columns.

8

Adding Calculations to a Fact

Objectives

After completing this lesson, you should be able to:

- Describe a calculation measure and its use in a business model
- Create calculation measures based on logical columns
- Create calculation measures based on physical columns
- Create calculation measures by using the Calculation Wizard

Objectives

Calculations enable data to be processed to derive measures that are valuable in analysis.

Business Problem

- Businesses need to track the effectiveness of their operations.
- Businesses want to use business—rather than technical—language to ask business questions.
  - Example: Show me the accounts-receivable balance as of Q3.
- Some information is derived from other data, such as:
  - Derive money outstanding.
    - Compare amount billed with amount received.
  - Derive units backordered.
    - Compare units ordered with units shipped.

ORACLE

Business Problem

Businesses often want to compare values of a measure and need a calculation to express the comparison. For example, a business might want to compare the number of units ordered with the number of units that have actually shipped. The business might want to see this comparison in terms of actual units or a percentage. The business wants its users to be able to formulate business questions by using terminology that they understand.

Business Solution

Create calculation measures in the Oracle BI business model by using the following methods:

- Use the Expression Builder to create calculation measures by using existing logical columns as objects in a formula.
- Use the Expression Builder to create calculation measures by using physical columns as objects in the formula.
- Use the Calculation Wizard to create calculation measures based on existing logical columns.
- Add calculation measures to the Presentation layer so that users can build analyses by using familiar terminology.

ORACLE

Business Solution

To meet the needs of various business measures, Oracle BI Server has a calculation engine to perform a multitude of calculations. Calculation measures enable users to make inquiries such as “Show me the accounts-receivable balance as of Q3” or “Show me the difference between units ordered and units shipped.”

You can use the Expression Builder dialog boxes in the Administration Tool to create constraints, aggregations, and other definitions within a repository. The Expression Builder provides automatic color highlighting and other formatting enhancements to make expressions easier to build and to read. The expressions you create with the Expression Builder are similar to expressions created with SQL. You can use all expressions constructed with the Expression Builder in SQL queries against the Oracle BI Server. You can use existing logical columns or physical columns in the expressions.

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.

The methods for creating calculation measures are covered in detail in subsequent slides.

Creating Calculation Measures by Using Existing Logical Columns

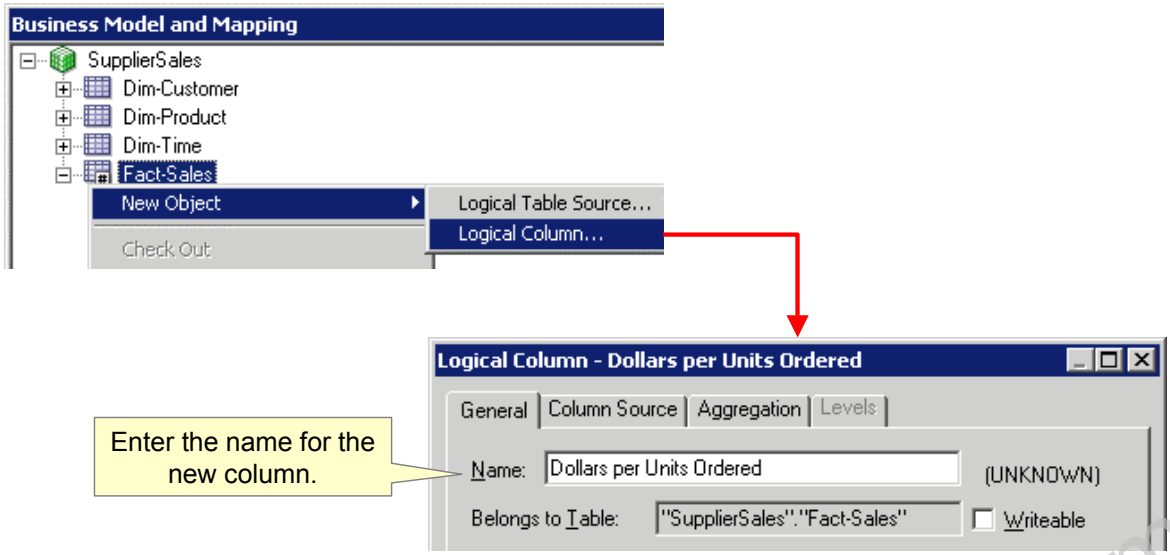
1. Create a new logical column.
2. Specify logical columns as the source.
3. Build a formula.

Creating Calculation Measures by Using Existing Logical Columns

These are the high-level steps for creating a calculation measure by using existing logical columns. Each step is described in detail in subsequent slides.

1. Create a New Logical Column

Right-click the fact table and select New Object > Logical Column.

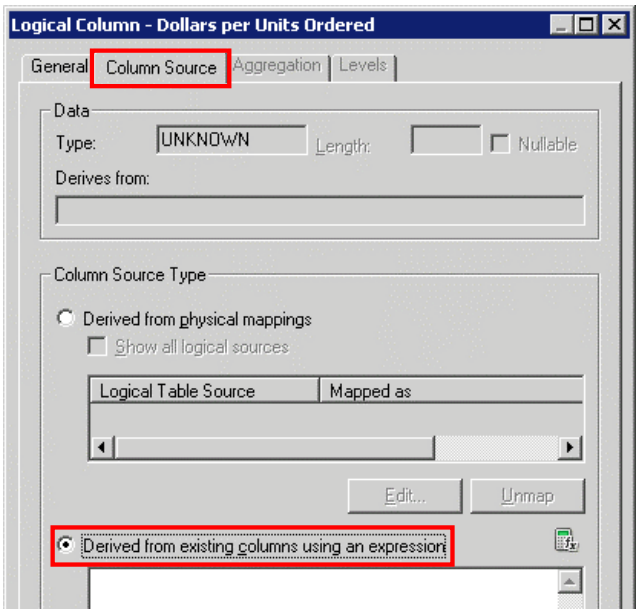


1. Create a New Logical Column

Start by creating a new logical column (`Dollars per Units Ordered`, in this example) for the fact table.

2. Specify Logical Columns as the Source

On the Column Source tab, select “Derived from existing columns using an expression.”

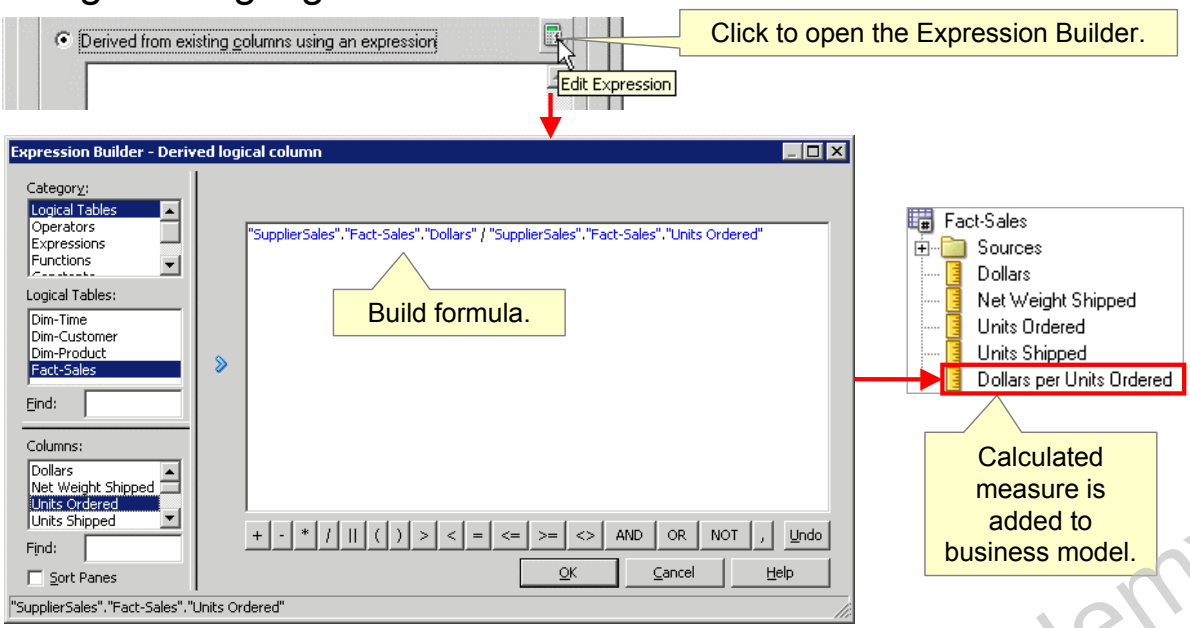


2. Specify Logical Columns As the Source

On the Column Source tab, select the “Derived from existing columns using an expression” check box to specify that you want to use existing logical columns in the calculation for the new column.

3. Build a Formula

Open the Expression Builder and build the calculation formula using existing logical columns.



3. Build a Formula

In this example, you divide dollars by units ordered. If there are any aggregation rules applied to the logical columns, those aggregation rules are automatically applied in the formula.

Both `Dollars` and `Units Ordered` already have `SUM` aggregation rules applied. (Recall that you applied these aggregation rules in an earlier lesson.) Thus the calculation formula is `sum(Dollars) / sum(Units Ordered)`.

After you finish, `Dollars per Units Ordered` appears as a logical column for the `Fact-Sales` logical fact table in the BMM layer.

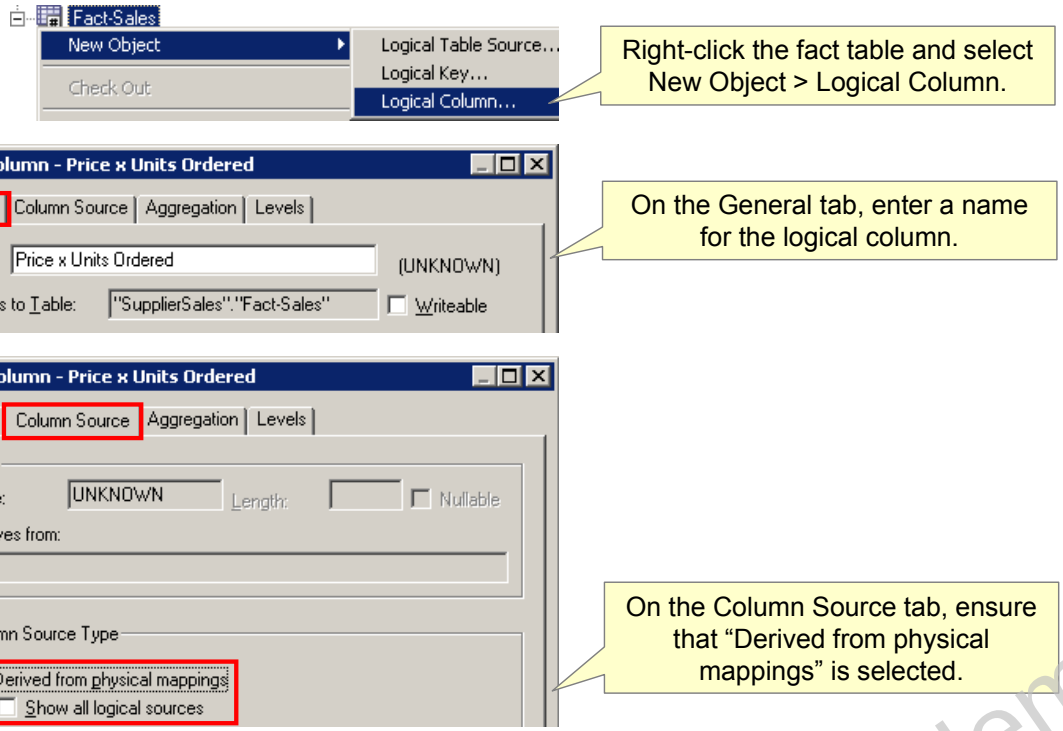
Creating Calculation Measures by Using Physical Columns

1. Create a new logical column.
2. Map the new column.
3. Build the formula.
4. Specify an aggregation rule.

Creating Calculation Measures by Using Physical Columns

These are the high-level steps for creating a calculation measure by using physical columns. Each step is described in subsequent slides.

1. Create a New Logical Column

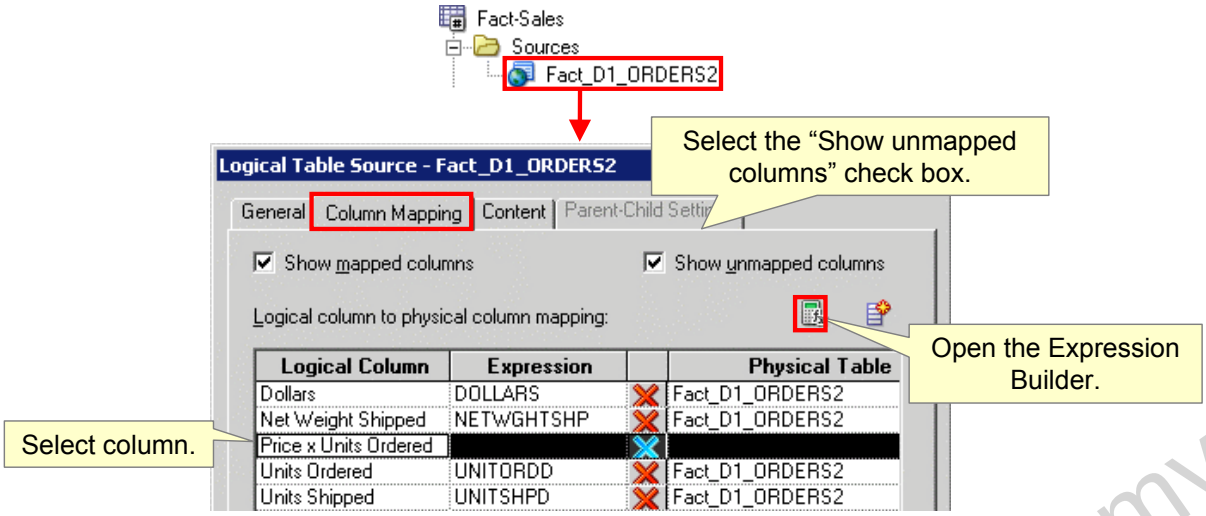


1. Create a New Logical Column

Start by creating a new logical column for the fact table. On the General tab, enter a name (`Price × Units Ordered`, in this example). On the Column Source tab, ensure that "Derived from physical mappings" is selected.

2. Map the New Column

Use the Column Mapping tab of the Logical Table Source dialog box to open the Expression Builder for the new column.

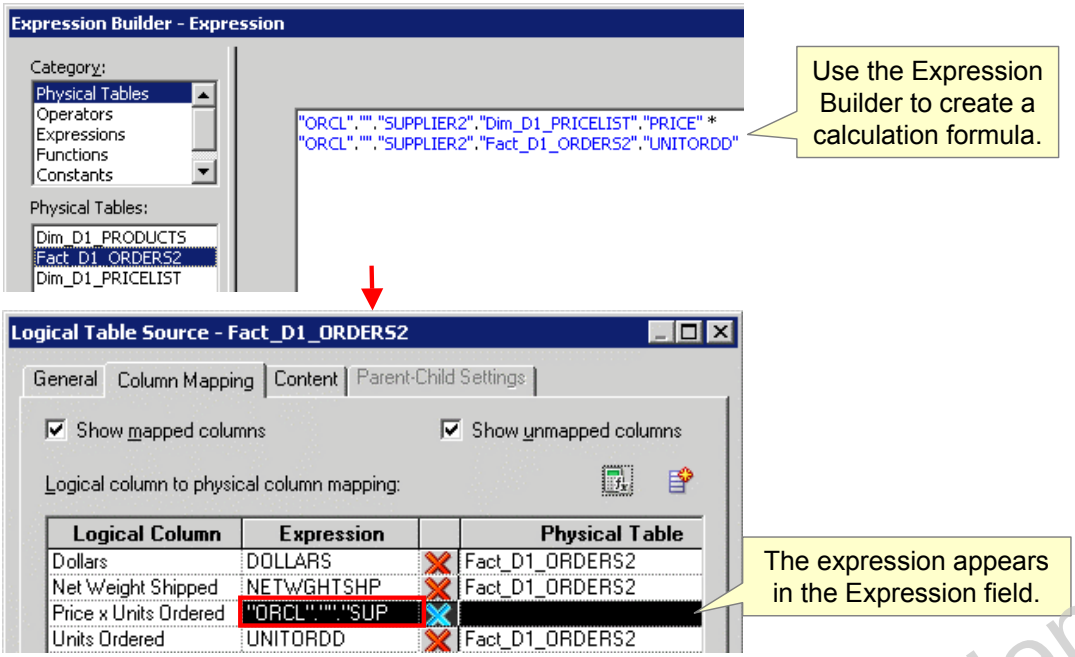


2. Map the New Column

To map the new column, open the properties dialog box for the existing logical table source (`Fact_D1_ORDERS2` in this example). On the Column Mapping tab, make sure that "Show unmapped columns" is selected. Select the unmapped column, `Price x Units Ordered`, and click the Edit Expression button to open the Expression Builder.

3. Build the Formula

Build the calculation formula by using physical columns.

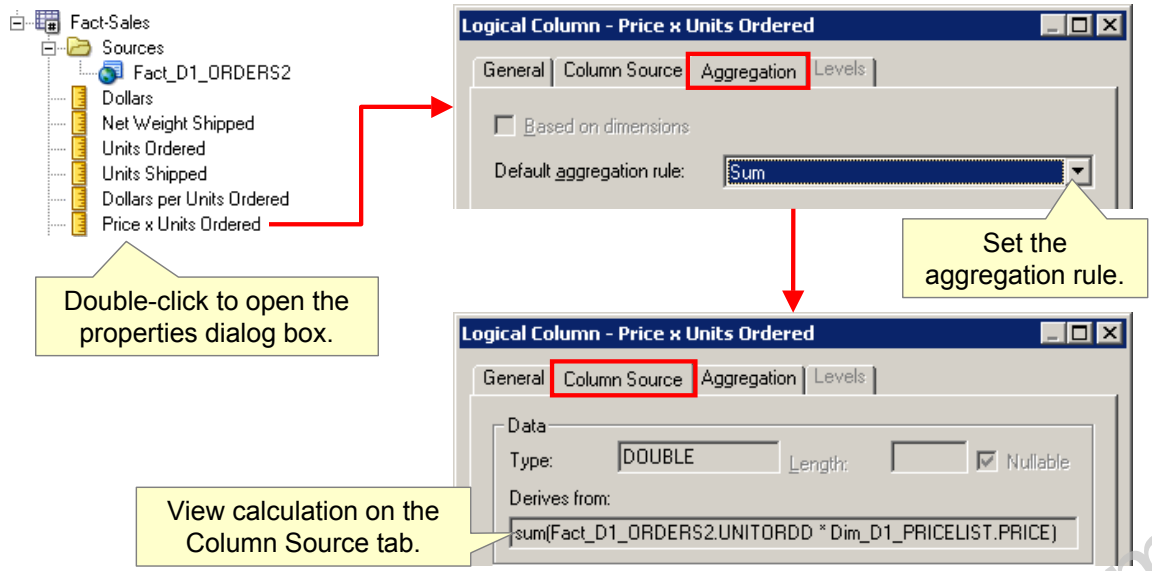


3. Build the Formula

Using the Expression Builder, create the calculation formula by using the physical columns (`PRICE` and `UNITORDD` in this example). When you close the Expression Builder, the expression appears in the Expression field for the column on the Column Mapping tab of the Logical Table Source dialog box.

4. Specify an Aggregation Rule

Click the Aggregation tab and set the aggregation rule for the column.



ORACLE

4. Specify an Aggregation Rule

Double-click the new logical column to open the column properties dialog box and then click the Aggregation tab. You need to set the aggregation rule on the new logical column because you are using physical columns, not logical columns, in the calculation formula.

Recall that in the previous example using logical columns in the formula, `SUM` aggregation rules were already set for the `Dollars` and `Units Ordered` logical columns, but not for the corresponding physical columns. Therefore, the calculation formula using logical columns looked like this: `sum(Dollars) / sum(Units Ordered)`. Here, the aggregation rules are applied before the calculation.

In this example, because you set the aggregation rule on a logical column by using physical columns as the source, the calculation formula looks like this: `sum(UNITORDD * PRICE)`. Here, the aggregation rule is applied after the calculation.

You can view the calculation for the logical column on the Column Source tab.

Steps for Using the Calculation Wizard

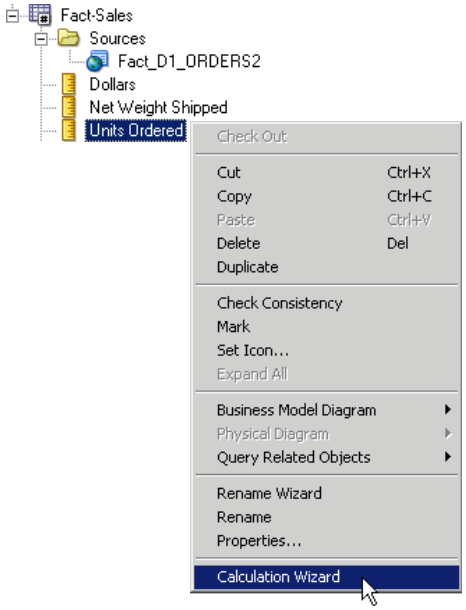
1. Open the Calculation Wizard.
2. Choose the columns for comparison.
3. Select the calculations.
4. Confirm the calculation measures.
5. New calculation measures are automatically added.

Steps for Using the Calculation Wizard

These are the high-level steps for creating calculation measures by using the Calculation Wizard. Each step is described in subsequent slides.

1. Open the Calculation Wizard

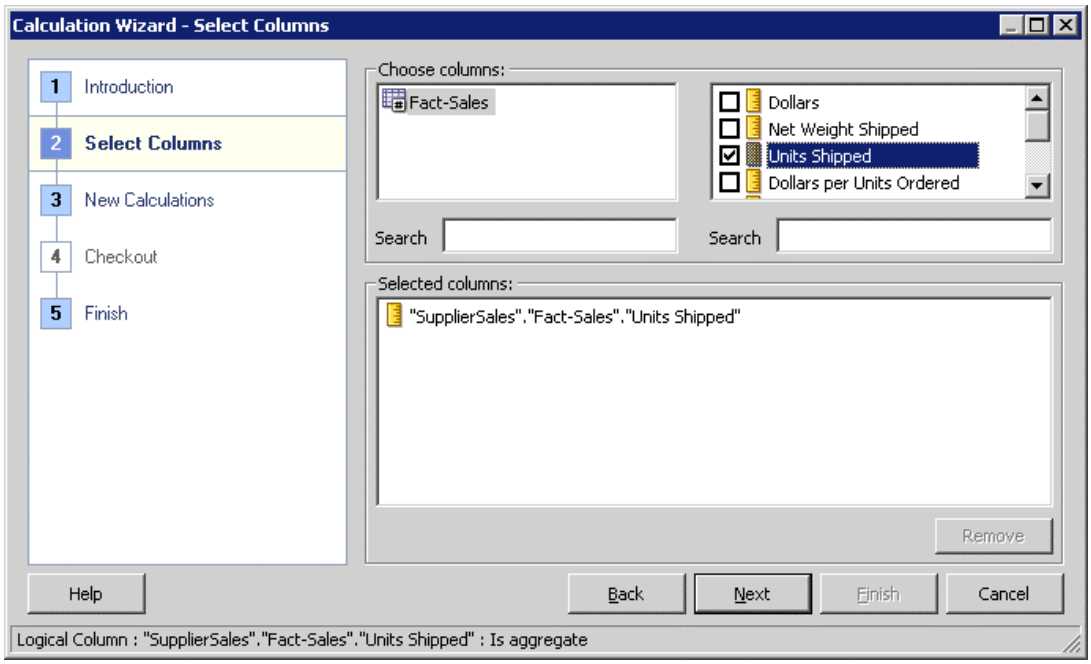
Right-click a logical column that you want to use in the calculation and select Calculation Wizard.



1. Open the Calculation Wizard

This slide shows how to start the Calculation Wizard. In this example, right-click `Units Ordered` and select Calculation Wizard to launch the wizard.

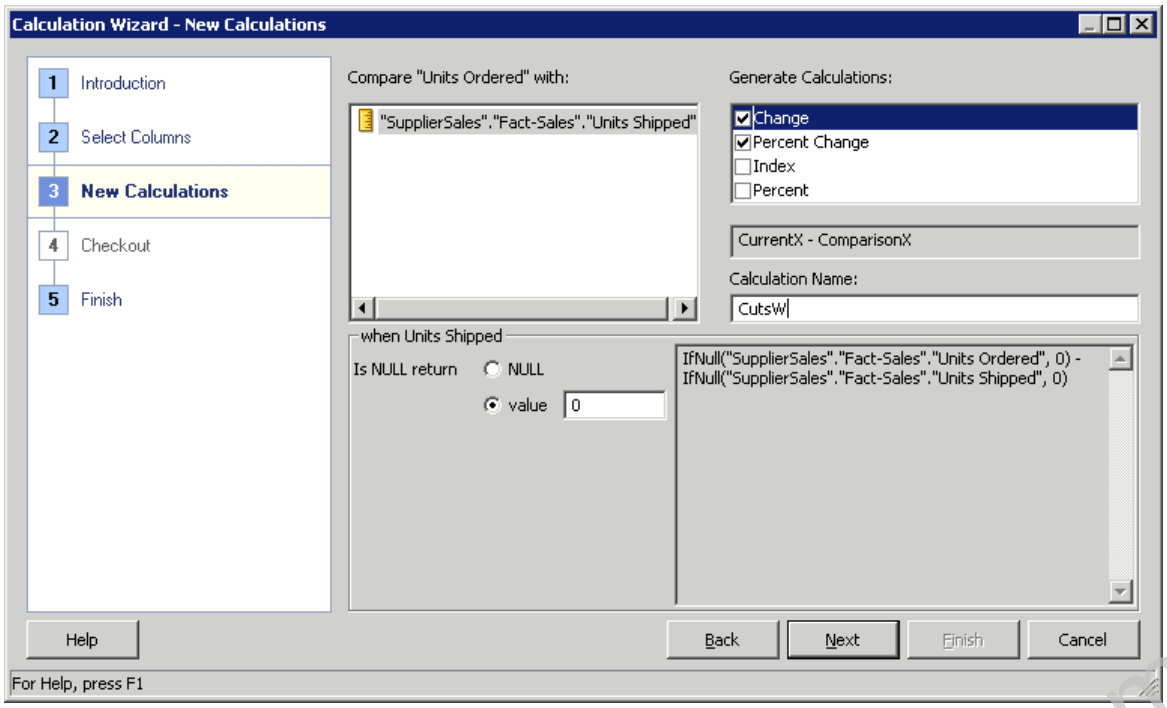
2. Choose the Columns for Comparison



2. Choose the Columns for Comparison

The wizard prompts you to select the columns that you want to compare with `Units Ordered`. In this example, `Units Shipped` is selected. Click Next to continue.

3. Select the Calculations

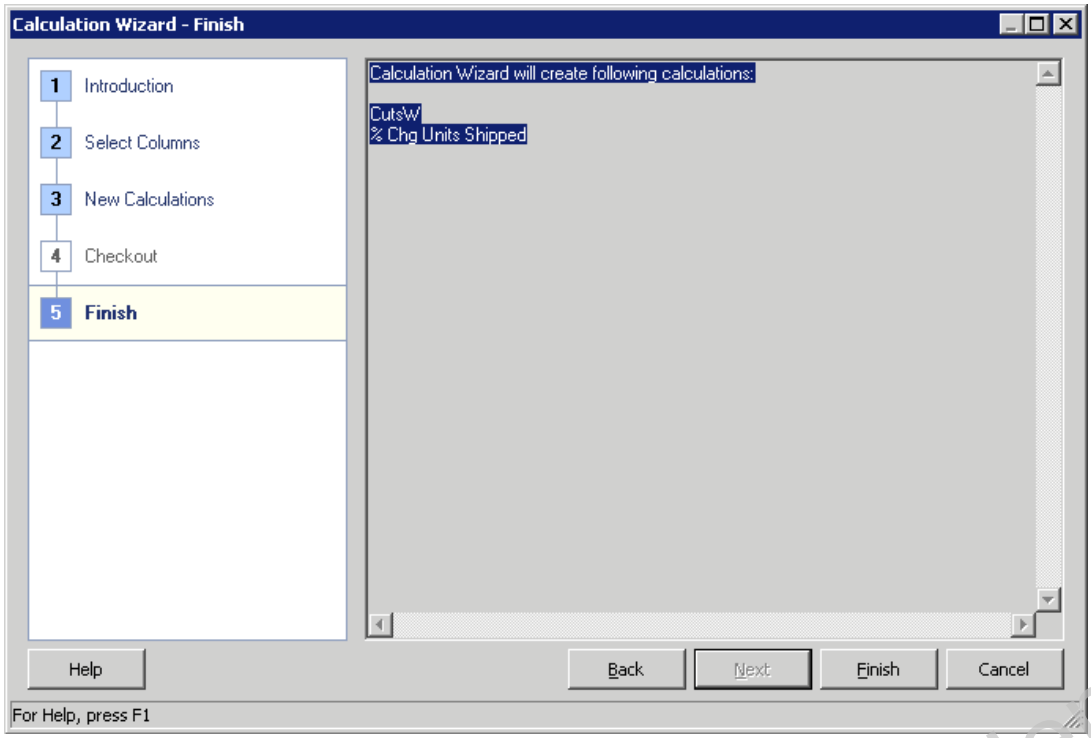


3. Select the Calculations

The wizard prompts you to select the calculations that you want to model (Change and Percent Change in this example). The Calculation Wizard can create up to four calculation measures (Change, Percent Change, Index, and Percent). The wizard also prompts you to specify what value should be returned if the column has no value (null). This must be specified for all selected calculations. Select each calculation and specify what value should be returned. You can enter any numeric value or `NULL`. Do not enter a value that would change the data type. For example, do not enter `NA`, which is a character value and not a numeric value.

You can also change the calculation name at this point. This is the name that appears in the business model when the Calculation Wizard finishes. In this example, the calculation measure that calculates the difference between `Units Ordered` and `Units Shipped` is renamed `CutsW`.

4. Confirm the Calculation Measures

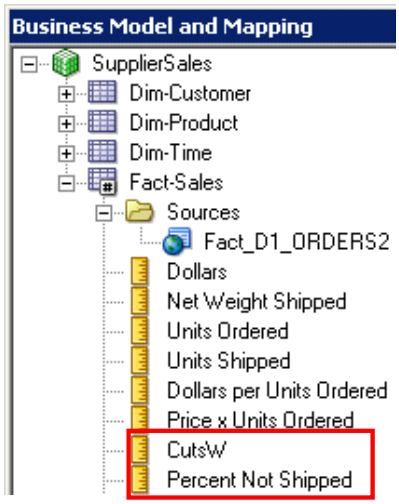


4. Confirm the Calculation Measures

The Calculation Wizard identifies the two calculation measures that it will create. Click Finish to add the new measures to the fact table.

5. New Calculation Measures Are Added

New calculation measures are automatically added to the business model.

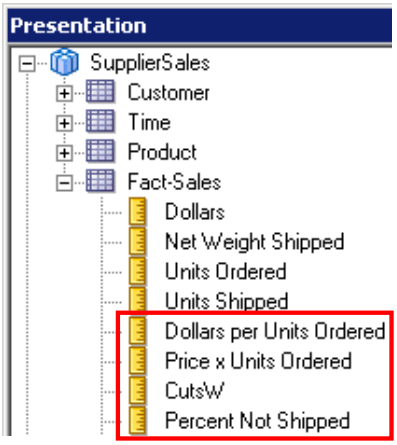


5. New Calculation Measures Are Added

When you click Finish, the new calculation measures are automatically added to the business model.

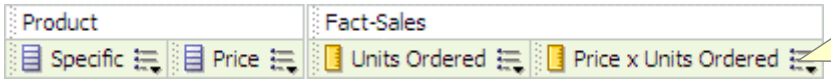
Add New Measures to the Presentation Layer

Add new calculation measures to the Presentation layer regardless of the method used to create the measures.



Examining a Query Using Physical Columns

Use physical columns for calculations that require an aggregation rule to be applied *after* the calculation.



Query uses a calculation measure based on physical columns.

```
select distinct 0 as c1,
     D1.c3 as c2,
     D1.c4 as c3,
     D1.c2 as c4,
     D1.c1 as c5
from
     (select sum(T514.UNITORDD) as c1,
               sum(T514.UNITORDD * T957.PRICE) as c2,
               T957.PRICE as c3,
               T503.SPECIFICDESCRIPTIN as c4
          from
               D1_PRODUCTS T503 /* Dim_D1_PRODUCTS */ ,
               D1_ORDERS2 T514 /* Fact_D1_ORDERS2 */ ,
               D1_PRICELIST T957 /* Dim_D1_PRICELIST */
          where  ( T503.PRODUCTKEY = T514.PRODKEY and T503.PRODUCTKEY = T957.PRODUCTKEY )
          group by T503.SPECIFICDESCRIPTIN, T957.PRICE
     ) D1
order by c3, c2
```

SQL calculates UNITORDD × PRICE and applies the SUM aggregation rule after the calculation.

Examining a Query Using Physical Columns

You use physical columns for calculations that require that an aggregation rule be applied after the calculation. This slide examines the query log for a query that uses the `Price x Units Ordered` calculation measure, which was built using physical columns in the formula. In this case, the SQL calculates the Units Ordered (`UNITORDD`) multiplied by the Price (`PRICE`), and then applies the `SUM` aggregation rule after the calculation. The calculation is expressed as `sum(UNITORDD * PRICE)`.

This is easily determined by knowing when an aggregation rule is applied in the calculation.

- Use logical columns when the aggregation rule is applied before the calculation.
- Use physical columns when the aggregation rule is applied after the calculation.

The next slide provides further explanation of when to configure a calculation measure based on physical columns.

Example: Using Physical Columns

To calculate total revenue accurately, multiply the `Unit Price` by the number of `Units Ordered` for each row, and then apply the `SUM` aggregation rule to the total.

Correct

Unit Price	Units Ordered	Total Dollars
500	2	1000
400	3	1200
300	4	1200
	Total Revenue	**3400**

SUM aggregation rule is applied after the calculation.

Incorrect

Unit Price	Units Ordered	Total Dollars
500	2	1000
400	3	1200
300	4	1200
1200	9	**10800**

SUM aggregation rule is applied before the calculation.

Example: Using Physical Columns

In this example, the calculation measure `Total Revenue` is based on physical columns and the aggregation rule is applied after the calculation.

Total revenue is derived by calculating, row-by-row, the unit price multiplied by the units ordered and then applying the `SUM` aggregation rule after the calculation to find total revenue, which equals $3,400 in this example.

What happens if the aggregation rule is applied *before* the calculation? In other words, what happens if you sum the unit price and units ordered columns first, and then you apply the calculation?

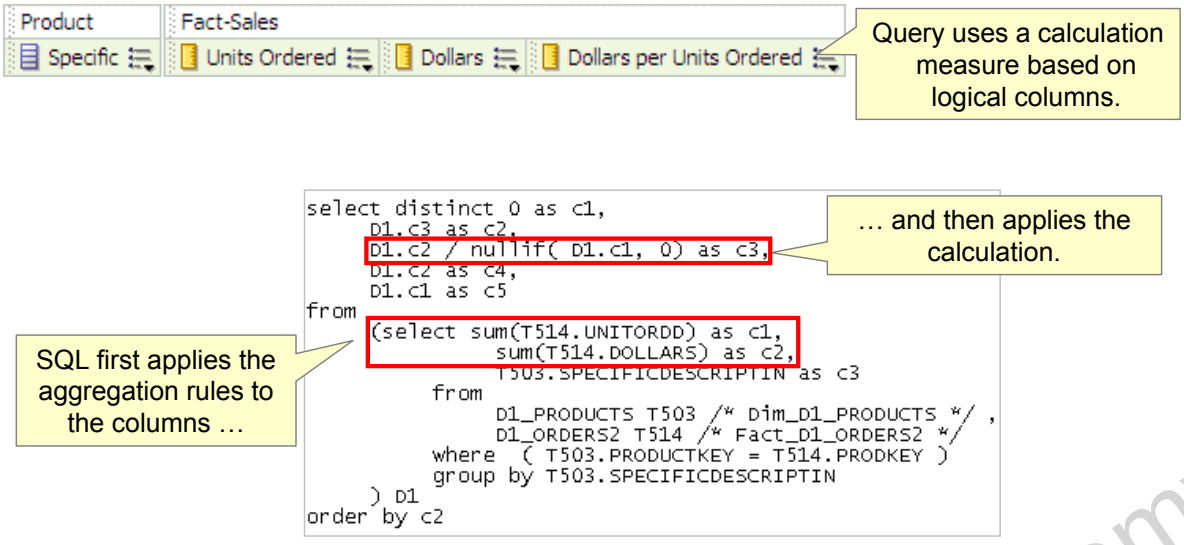
Applying the summation (aggregation rule) before the calculation produces the following incorrect results:

- `SUM(Unit price)`: $500 + $400 + $300 = $1,200
- `SUM(Units ordered)`: 2 + 3 + 4 = 9
- $1,200 × 9 = $10,800

This calculation is not correct. Applying the aggregation rule (summing the unit price column and the units ordered column) *before* the calculation (unit price × units ordered) does not accurately provide total revenue.

Examining a Query Using Logical Columns

Use logical columns for calculation formulas that require an aggregation rule that is applied *before* the calculation.



Examining a Query Using Logical Columns

Use logical columns for calculation formulas that require an aggregation rule that is applied before the calculation. This slide examines the query log for a query that uses the `Dollars per Units Measured` calculation measure, which was built using existing logical columns in the formula.

In this case, the SQL applies the `SUM` aggregation rule to `Units Ordered` (`UNITORDD`) and `Dollars` (`DOLLARS`) first and then calculates the division. This is expressed in the example in the slide as `D1.c2 / nullif(D1.c1,0)`.

The next slide provides further explanation of when to configure a calculation measure based on logical columns.

Example: Using Logical Columns

To accurately calculate unit price, sum `Dollars` and `Units Ordered` first and then divide `Dollars` by `Units Ordered` to determine `Unit Price`.

Dollars	Units Ordered	Unit Price
6000	6	1000
5000	5	1000
100	5	20

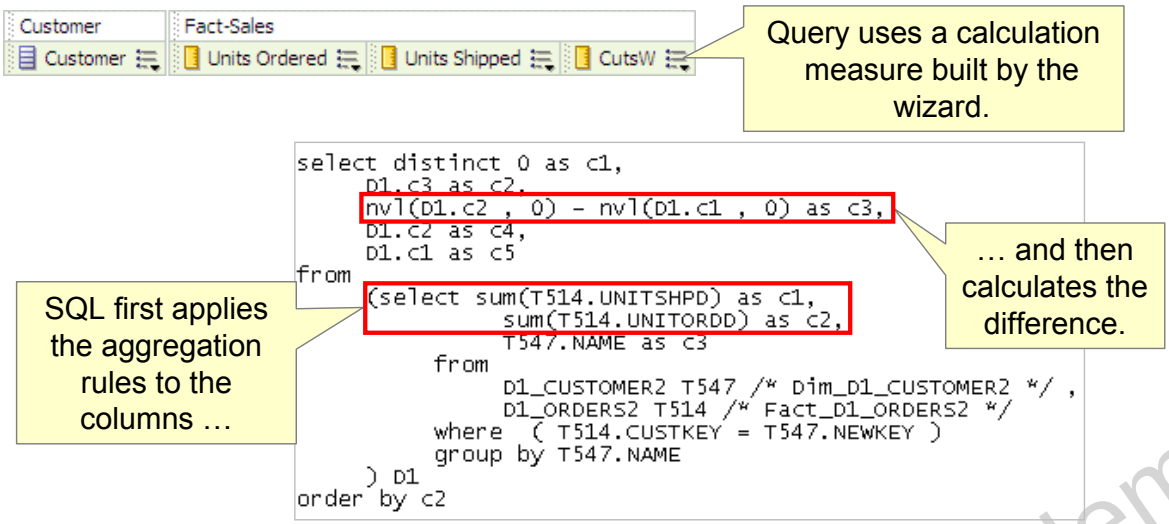
Example: Using Logical Columns

In this example, the calculation measure `Unit Price` is based on logical columns; therefore, the aggregation rule is applied before the calculation. `Unit Price` is found by applying the `SUM` aggregation rule to both `Dollars` and `Units Ordered` before calculating the unit price (dividing `Dollars` by `Units Ordered`).

The advantage of defining a logical column formula based on existing logical columns is that you have to define it only once. When you define a formula based on physical columns, you must define a column mapping for each physical source from which the formula can be derived. However, sometimes you have no choice if you have to use physical columns to apply an aggregation rule after a calculation.

Examining a Query Using the Calculation Wizard

The Calculation Wizard uses logical columns as objects in its calculations, so aggregation rules are applied before the calculation.

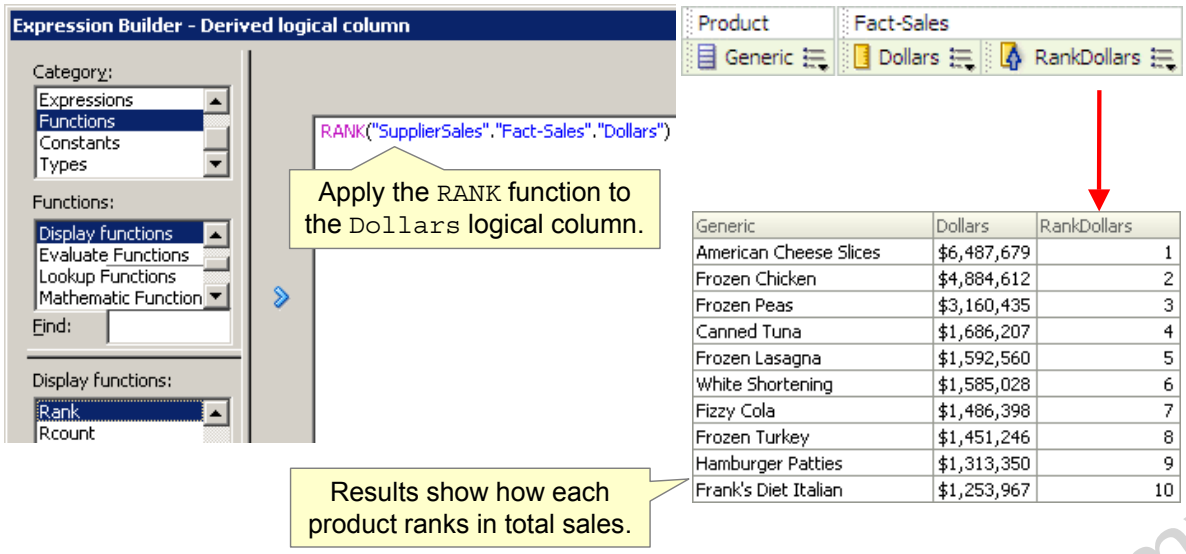


Examining a Query Using the Calculation Wizard

When you use the Calculation Wizard to model calculation measures, it is important to consider that the wizard uses logical columns as objects in the calculation formulation. Therefore, an aggregation rule is applied *before* the calculation.

Using Functions to Create Expressions

Use the provided functions to build expressions in the Expression Builder.



Using Functions to Create Expressions

To set up an expression using a function, select the Functions folder from the Category pane, select a function type from the Functions pane, and 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 the Logical Tables folder 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.

The screenshot shows an example of an expression that uses the `RANK` function. The `RANK` function is applied to the `Dollars` logical column. The result shows total dollars for each product and each product's rank in total sales.

Summary

In this lesson, you should have learned how to:

- Describe a calculation measure and its use in a business model
- Create new calculation measures based on existing logical columns
- Create new calculation measures based on physical columns
- Create new calculation measures by using the Calculation Wizard

Practice 8-1 Overview: Creating Calculation Measures

This practice covers two different methods for creating measures that contain calculations:

- Create a measure that is derived from other existing logical columns as a way to apply post-aggregation calculations to the measure.
- Create a measure using physical columns as a way to apply pre-aggregation calculations to the measure.

Practice 8-1 Overview: Creating Calculation Measures by Using Logical Columns

You use two different methods to create measures that contain calculations.

First you create a measure that is derived from other existing logical columns as a way to apply post-aggregation calculations to the measure. Then you create a measure using physical columns as a way to apply pre-aggregation calculations to the measure.

In both cases, you use Oracle BI queries and the query log to verify your results.

Practice 8-2 Overview: Creating Calculation Measures by Using the Calculation Wizard

This practice covers creating a calculation measure by using the Calculation Wizard.

Practice 8-2 Overview: Creating Calculation Measures by Using the Calculation Wizard

Using the Calculation Wizard, you create two calculation measures: `Change Units Shipped` and `Percent Change Units Shipped`. The `Change Units Shipped` measure calculates the difference between the units ordered and the units shipped. The `Percent Change Units Shipped` measure calculates the percentage of total units ordered that have not been shipped.

The calculation measures created by the Calculation Wizard are based on existing logical columns. You rename these columns `CutsW` and `Percent Not Shipped` in the Presentation layer.

Practice 8-3 Overview: Creating a RANK Measure

This practice covers using the RANK function to calculate rank for a logical column.

Practice 8-3 Overview: Creating a RANK Measure

In this practice, you use the RANK function to calculate rank for the Dollars logical column.

9

Working with Logical Dimensions

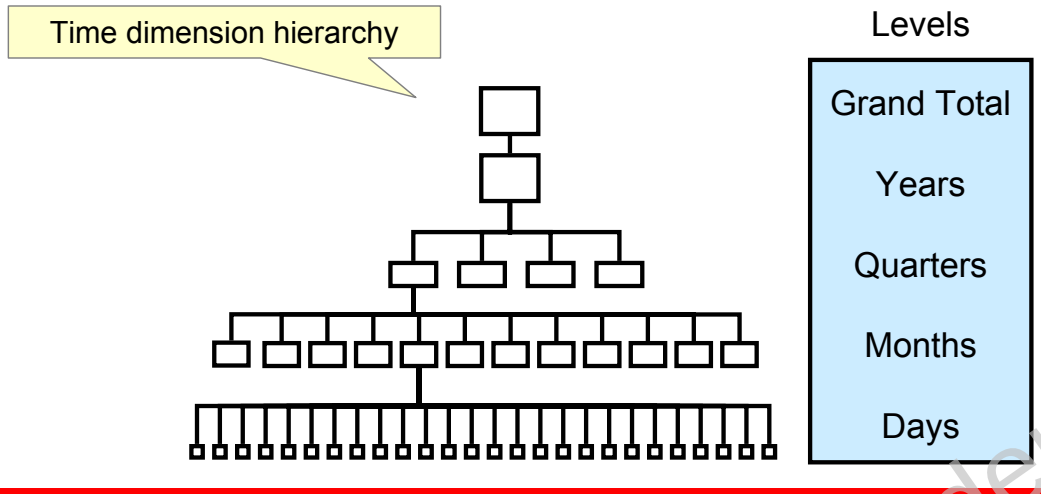
Objectives

After completing this lesson, you should be able to:

- Create logical dimensions with level-based hierarchies
- Create logical dimensions with parent-child hierarchies
- Create level-based measures
- Create share measures
- Create calculated members

Logical Dimensions

- Introduce formal hierarchies into a business model
- Establish levels for data groupings and calculations
- Provide paths for drill down



Logical Dimensions

Logical dimensions introduce formal hierarchies into a business model, enabling Oracle BI Server to calculate useful measures and enabling users to drill down to more detail. In a business model, a logical dimension represents a hierarchical organization of logical columns belonging to a single logical dimension table. Common logical dimensions used in a business model are time periods, products, customers, suppliers, and so on.

Logical dimensions are created in the Business Model and Mapping (BMM) layer and can be added to the Presentation layer for use in analyses and dashboards. In each dimension, you organize dimension attributes into hierarchical levels. These levels represent the organizational rules and reporting needs required by your business. They provide the structure that Oracle BI Server uses to drill into and across dimensions to get more detailed views of the data. Dimension hierarchy levels are used to perform aggregate navigation, configure level-based measure calculations, and determine which attributes appear when Oracle BI users drill down in their data requests.

The graphic illustrates a Time dimension hierarchy, where Grand Total is the level representing the grand total for the dimension, Years is the first parent level, Quarters is a child level of Years, Months is a child level of Quarters, and Days is a child level of Months.

Logical Dimensions: Types

There are two types of logical dimensions:

- Dimensions with *level-based hierarchies*
  - Members of the same type occur only at a single level.
  - Oracle BI Server supports unbalanced (ragged), skip-level, and time hierarchies.
- Dimensions with *parent-child hierarchies*
  - All members have the same type (for example, an organizational reporting hierarchy).

Logical Dimensions: Types

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 of the same type occur only at a single level, while members in parent-child hierarchies all have the same type. The most common real-world example of a parent-child hierarchy is an organizational reporting hierarchy chart. Parent-child logical dimensions are discusses in more detail later in this lesson.

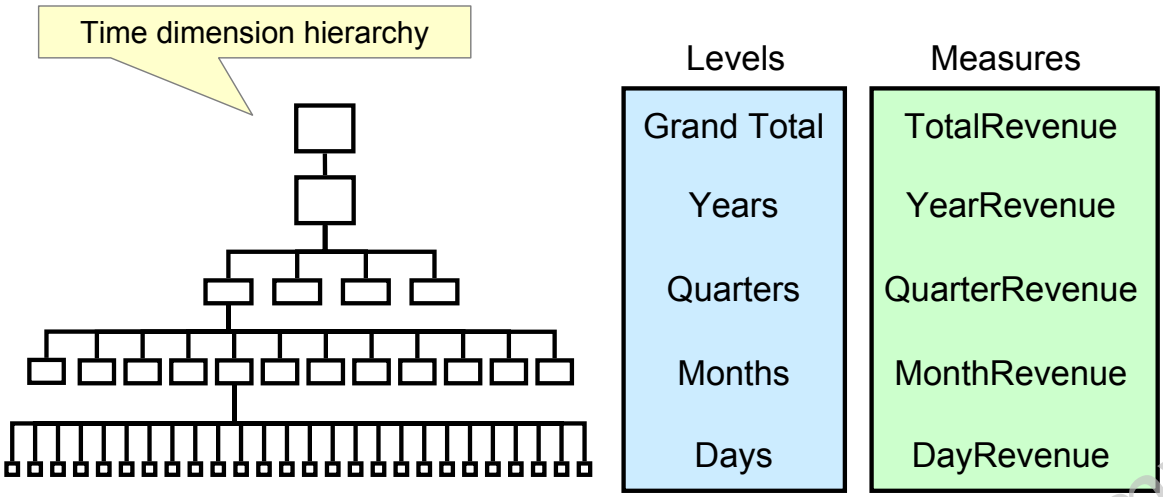
Oracle BI Server also supports unbalanced (ragged), skip-level, and time hierarchies. An unbalanced hierarchy is a hierarchy in which the leaves (members with no children) do not necessarily have the same depth. For example, a site can choose to have data for the current month at the day level, previous months data at the month level, and the previous five years of data at the quarter level.

A skip-level hierarchy is a hierarchy in which 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.

Time hierarchies provide special functionality for modeling time series data and are discussed in more detail in the lesson titled “Modeling Time Series Data.”

Level-Based Measures

Are columns whose values are calculated to a specific level of aggregation



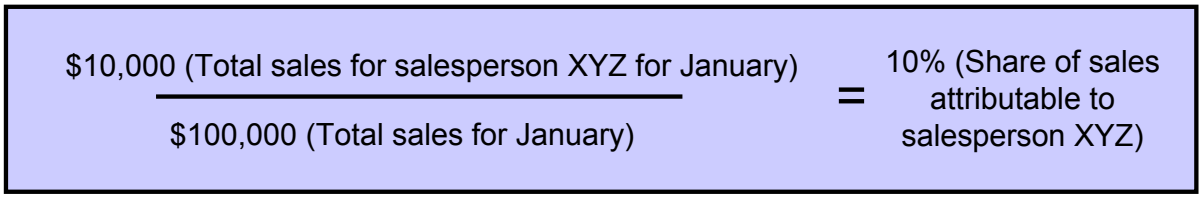
Level-Based Measures

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 year, quarter, month, or day.

To achieve this, you could set up logical columns to measure TotalRevenue, YearRevenue, QuarterRevenue, MonthRevenue, and DayRevenue. The TotalRevenue measure is an example of a level-based measure at the Grand Total level. This would calculate all revenue across all years. Level-based measures allow a single query to return data at multiple levels of aggregation. For example, a single query could return total revenue by year, quarter, month, and day for a single customer.

Share Measures

Are calculated by dividing a measure by a level-based measure to calculate a percentage

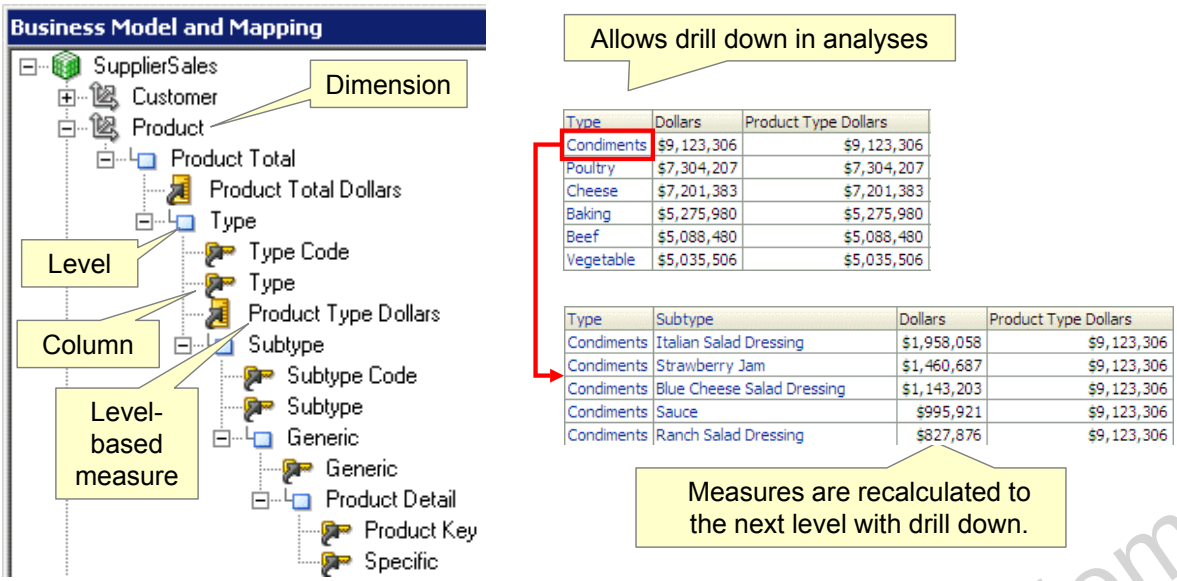


Share Measures

Level-based measures are also useful in creating share measures, which are calculated by taking a measure and dividing it by a level-based measure to calculate a percentage. For example, you can divide “salesperson revenue for a specific month” by “total revenue for a specific month” to calculate the share of the monthly revenue that is generated by each salesperson.

Logical Dimension: Example

This is an example of a `Product` logical dimension based on the `Dim-Product` logical dimension table.



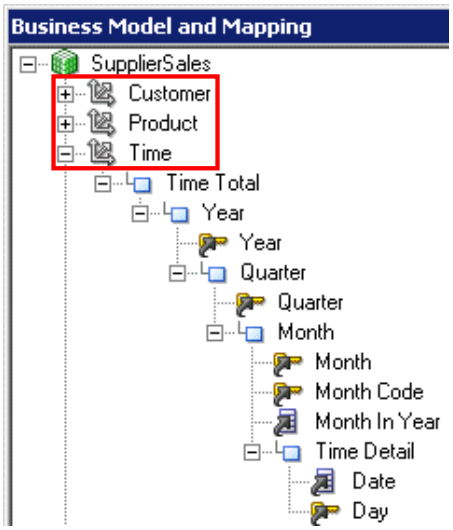
Logical Dimension: Example

Here is a specific example of a logical dimension. In the `SupplierSales` business model, there is a logical dimension hierarchy based on the `Dim-Product` logical dimension table. In this example, the logical dimension is named Product and is identified by the icon with multiple arrows. The hierarchy contains five levels: Product Total, Type, Subtype, Generic, and Product Detail. Level-based measures are associated with two levels of the hierarchy. Product Total Dollars is associated with the Product Total level. Product Type Dollars is associated with the Type level. There are also other columns and keys associated with each level.

The screenshots on the right provide an example of what a user might see in the Oracle BI UI. The blue highlighted field indicates that it is possible to drill down to the next level. In this example, the user drilled down on the Condiments type to view the Subtypes. As you will see later in this lesson, the Product Total Dollars level-based measure is used to build the Product Share measure shown here. The measures are automatically recalculated as you drill down through the hierarchy levels.

ABC Example

ABC wants to implement logical dimensions for Customer, Product, and Time.



ABC Example

ABC wants to implement three logical dimensions: Customer, Product, and Time. These logical dimensions enable ABC to do the following:

- Build level-based measures.
- Build share measures.
- Provide drill down on charts and tables.
- Define aggregation rules that vary by dimension.

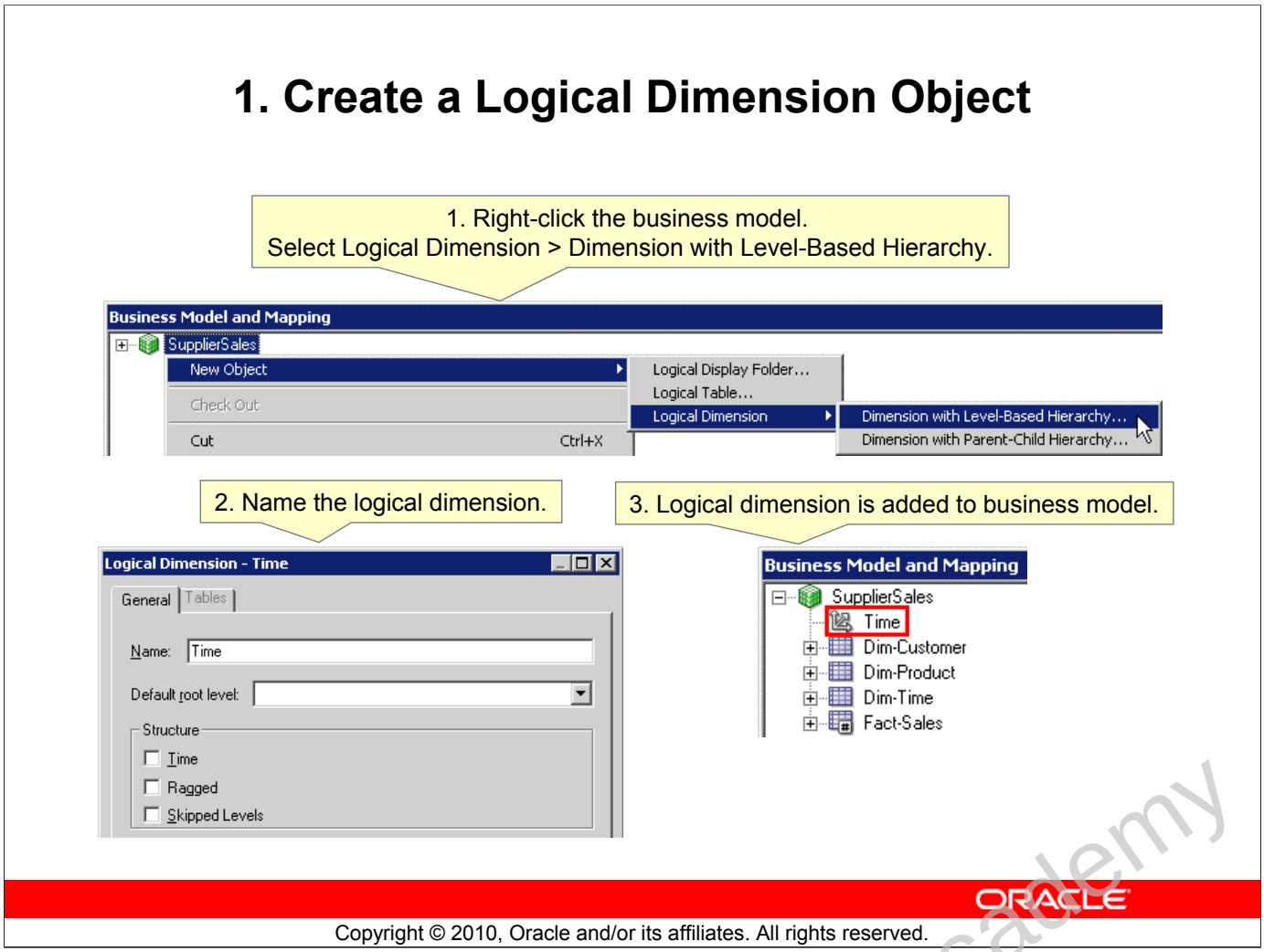
Each of these topics is discussed in detail in the following slides.

Creating a Level-Based Logical Dimension

1. Create a logical dimension object.
2. Add a parent-level object.
3. Add child-level objects.
4. Specify level columns.
5. Create level keys.
6. Set the preferred drill path.
7. Create level-based measures.
8. Create additional level-based measures.
9. Create share measures.
10. Add measures to the Presentation layer.
11. Create presentation hierarchies.
12. Test measures and hierarchies.

Creating a Level-Based Logical Dimension

This slide lists the high-level steps to create a level-based logical dimension. Each step is covered in detail in subsequent slides.



1. Create a Logical Dimension Object

The first step is to create the logical dimension object. There are two methods for creating a logical dimension object. The slide illustrates one method, which is to right-click the business model object, select New Object > Logical Dimension > Dimension with Level-Based Hierarchy, and name the dimension in the Logical Dimension dialog box.

In the Logical Dimension dialog box, note that there are options to select Time, Ragged, or Skipped-Levels hierarchy structures. The steps for creating these different types of level-based hierarchies are fundamentally the same. You select the appropriate hierarchy structure based on the structure of the underlying data.

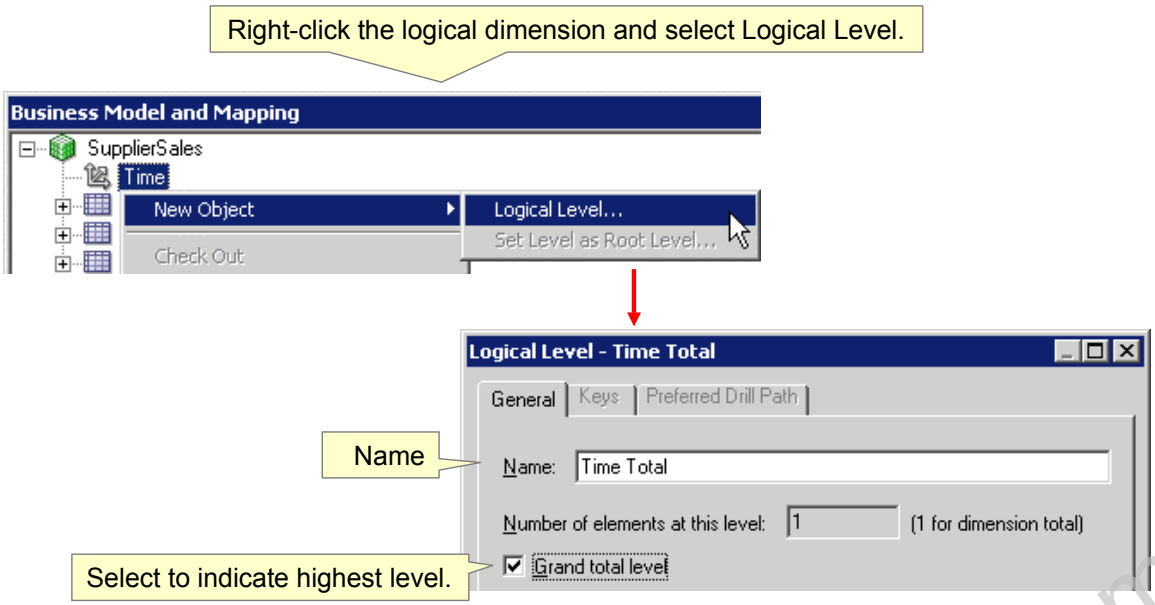
Another method is to right-click a logical dimension table and select Create Dimension. For example, if you right-click the `Dim-Time` logical dimension table and select Create Dimension, a new dimension called TimeDim is automatically created with two levels, Dim-Time Total and Dim-Time Detail. The Dim-Time Detail level is populated with all columns from the `Dim-Time` logical dimension table.

After it is created, the dimension hierarchy appears in the business model with a three-arrow icon.

The remaining steps assume that you have used the first method to create the dimension.

2. Add a Parent-Level Object

Create the highest level of the hierarchy first.



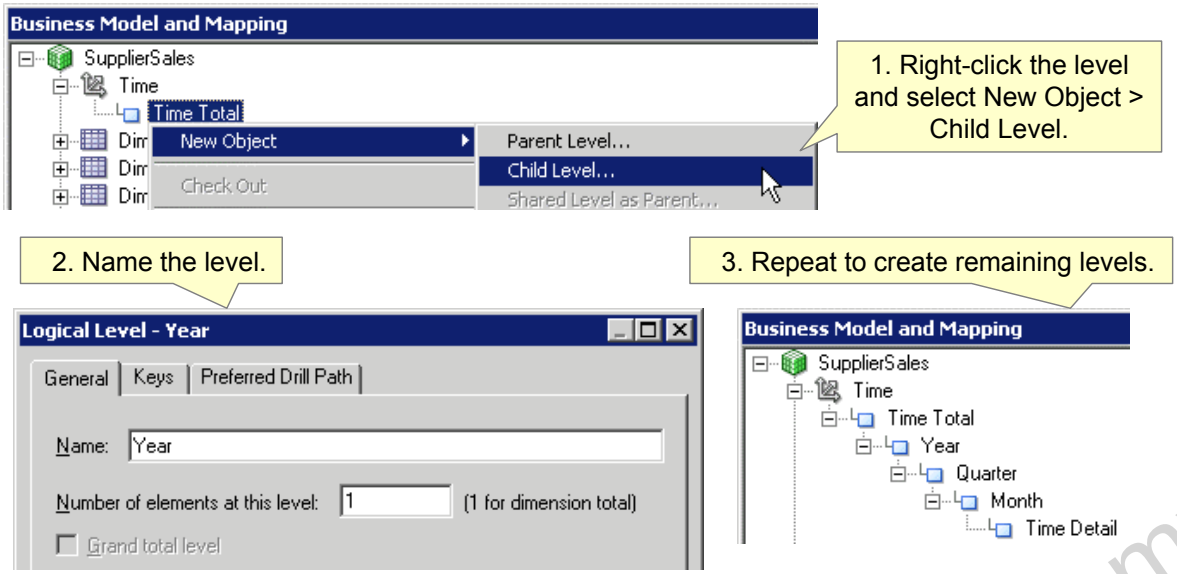
2. Add a Parent-Level Object

The next step is to begin to create the levels of the hierarchy. You create the highest level of the hierarchy first.

Right-click the logical dimension and select New Object > Logical Level. Typically the first level you create is the grand total level. Give the level a name and select the “Grand total level” check box to indicate that this is the grand total level.

3. Add Child-Level Objects

Add subsequent levels in the hierarchy.



3. Add Child-Level Objects

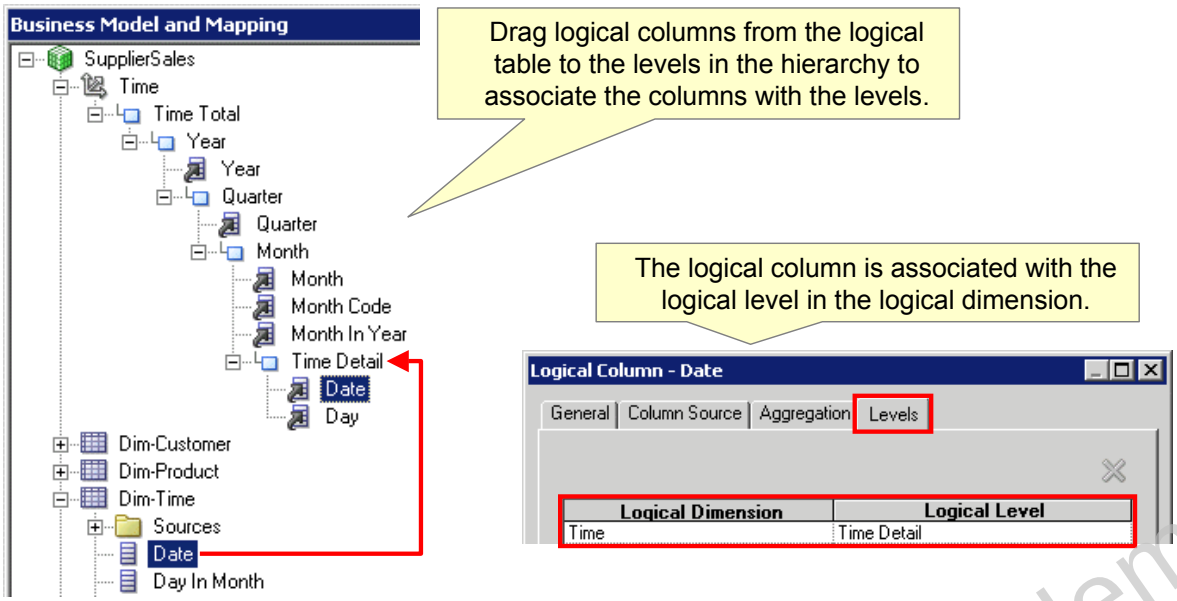
The next step is to create the child levels.

Right-click a level in the dimension and select New Object > Child Level. Then name the level.

Repeat this procedure to create the desired number of levels.

4. Specify Level Columns

Associate logical columns with logical levels in the hierarchy.



4. Specify Level Columns

After you create all levels in a dimension, you must associate logical columns with the levels. The quickest way is to drag one or more columns from the logical dimension table to each level (except the Grand Total level). The first time you drag a column to a dimension, it associates the logical table with the logical dimension. It also associates the logical column with that level of the dimension.

In this example, you drag the Date column from the `Dim-Time` dimension table to the Time Detail level in the Time logical dimension hierarchy. The Date column is now associated with the Time Detail level in the Dim-Time logical dimension hierarchy. The screenshot shows the Levels tab in the Logical Column - Date properties box, indicating that the Date column is associated with the Time Detail level in the Time logical dimension.

Here are some guidelines for associating columns with levels:

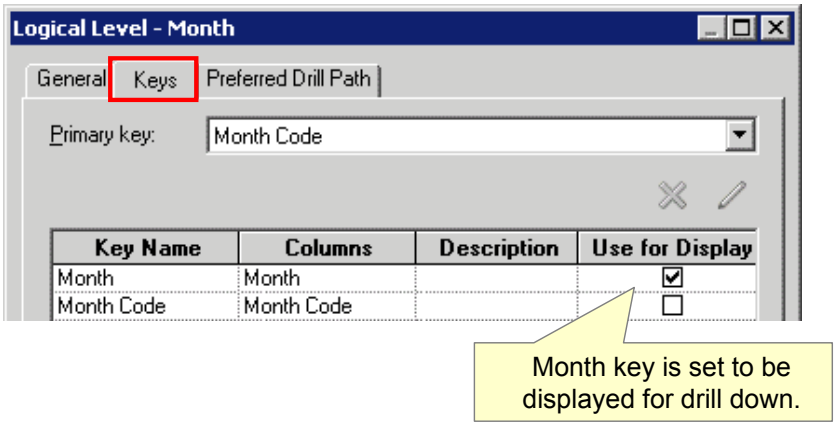
- Not all columns in the dimension table must be associated explicitly with a level. The ones that are not will be associated implicitly with the lowest level.
- No column can be associated with more than one level, although it may be part of the level key of a lower level.

4. Specify Level Columns (continued)

- If a column pertains to more than one level, associate it with the highest level to which it belongs.
- No level except the Grand Total level can exist without at least one column associated with it.
- The Detail level (lowest level) must have the logical key of the dimension table that is associated with it, and the logical key of the dimension table must be the level key for the Detail level.

5. Create Level Keys

Level keys define the unique elements in each level and provide the context for drill down.



5. Create Level Keys

Each level except the Grand Total level must have one or more attributes that compose a level key. The level key defines the unique elements in each level and provides the context for drill down. A level may have more than one level key. When that is the case, you must specify which key is the primary key of that level. All dimension sources that have aggregate content at a specified level must contain the column that is the primary key of that level.

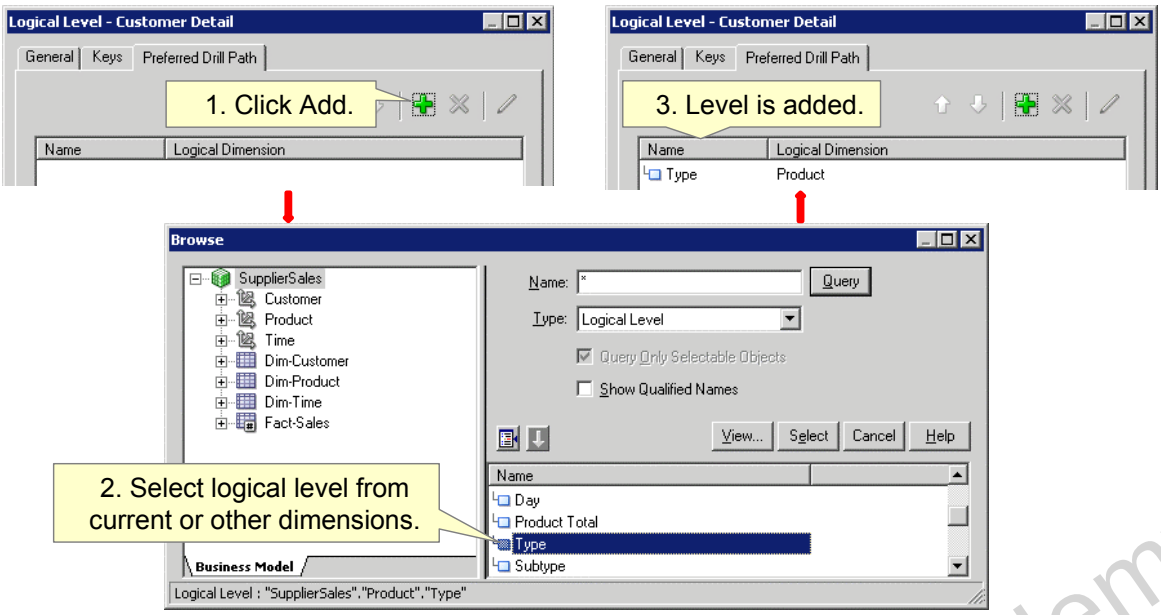
To create a level key:

1. Double-click a level to open the level properties window.
2. Click the Keys tab.
3. Enter a name for the key and select the appropriate column or columns.
4. Each level should have one level key that is displayed when a user clicks to drill down. This may or may not be the primary key of the level. To set the level key to be displayed, select the "Use for Display" check box in the Logical Level Key dialog box.

An alternative method is to right-click a level column and select New Logical Level Key. In this example, there are two separate keys: Month and Month Code. It is possible to create composite keys consisting of multiple columns, but that is not the case in this example. Month Code is the primary key and "Use for Display" is selected for the Month key. Therefore, when users drill down in a report from the next-highest level, the default is to drill down to the Month column rather than the Month Code column.

6. Set the Preferred Drill Path

Use Preferred Drill Path to specify a drill path outside the normal drill path defined by the dimension-level hierarchy.



6. Set the Preferred Drill Path

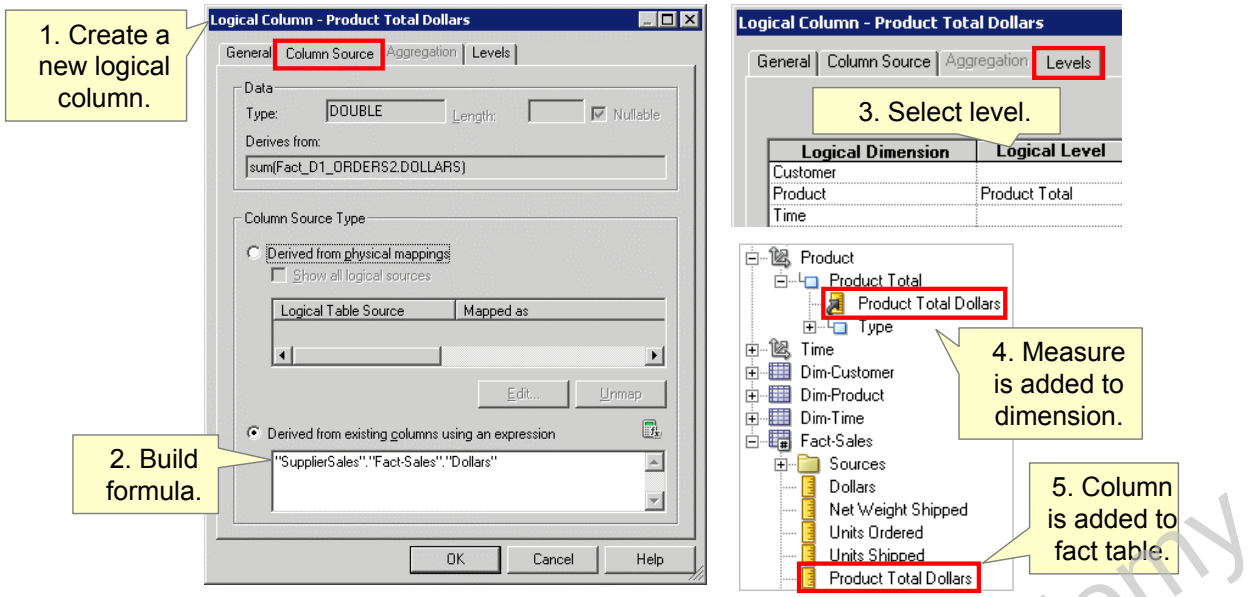
You can use the Preferred Drill Path tab to identify the path to use when Oracle BI Presentation Services users drill down in their data requests. You should use this to specify only a drill path that is outside the normal drill path defined by the dimensional level hierarchy. It is most commonly used to drill from one dimension to another. For example, from the Customer Detail level you set the preferred drill path to Product Type so that users can drill down to see products ordered by customers.

To add a dimension level to the preferred drill path:

- Click the Add button to open the Browse dialog box, where you can select the logical levels to include in the drill path. You can select logical levels from the current dimension or from other dimensions.
- Click OK to return to the Level dialog box, where the names of the levels are added to the Names pane.

7. Create Level-Based Measures

Example: Create a level-based measure for a Grand Total level that refers to an existing logical fact column.



7. Create Level-Based Measures

The next step is to add level-based measures to the dimensions. The examples in this slide and the slides that follow assume that you have built a Product dimension hierarchy in addition to the Time dimension hierarchy illustrated in the previous slides.

ABC wants to build level-based measures that calculate total dollars (revenue) at various levels (for example, the Product Total and Product Type levels). The Product Total level-based measure is used later in this lesson to build a share measure.

This example shows how to build the Product Total Dollars level-based measure and associate it with the Grand Total (highest) level in the Product dimension.

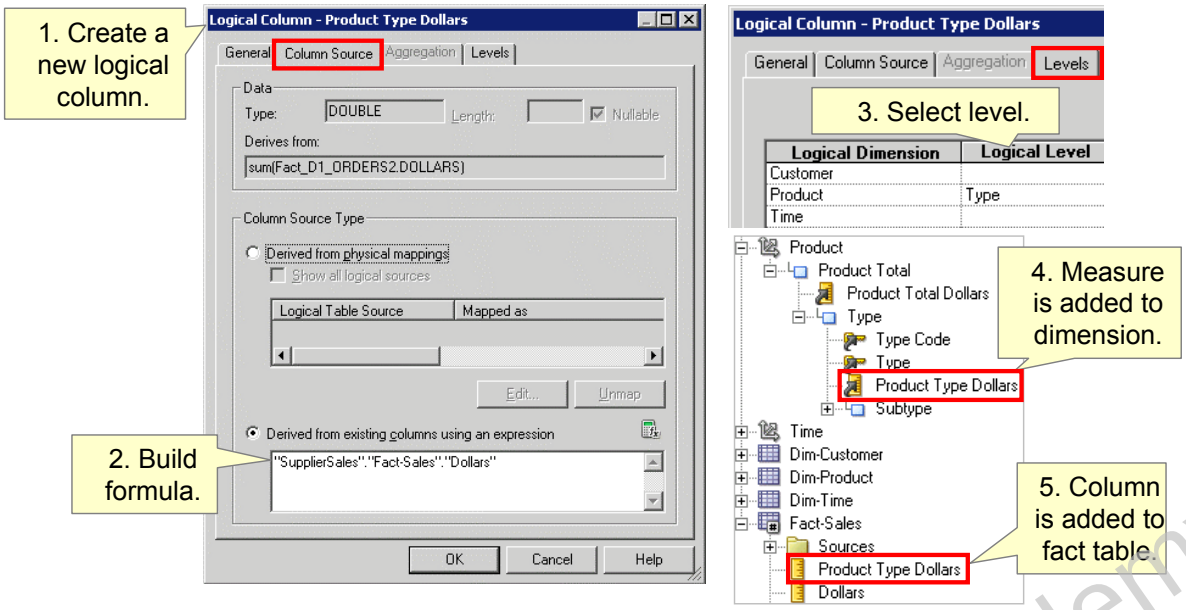
1. Create a new logical column named Product Total Dollars in the `Fact-Sales` table.
2. In the Product Total Dollars properties box, use the Expression Builder to create a measure based on the existing Dollars fact column. In this case, the Dollars column has a default aggregation rule of `SUM`.

7. Create Level-Based Measures (continued)

3. Select the Levels tab and associate the Product Total Dollars column with the Grand Total level (Product Total, in this example) in the Product logical dimension.
4. In the business model, the level-based measure is added to the Product logical dimension hierarchy.
5. The new fact column, Product Total Dollars, is added to the `Fact-Sales` table.

8. Create Additional Level-Based Measures

Create other level-based measures to make measures from various levels available simultaneously in queries.



8. Create Additional Level-Based Measures

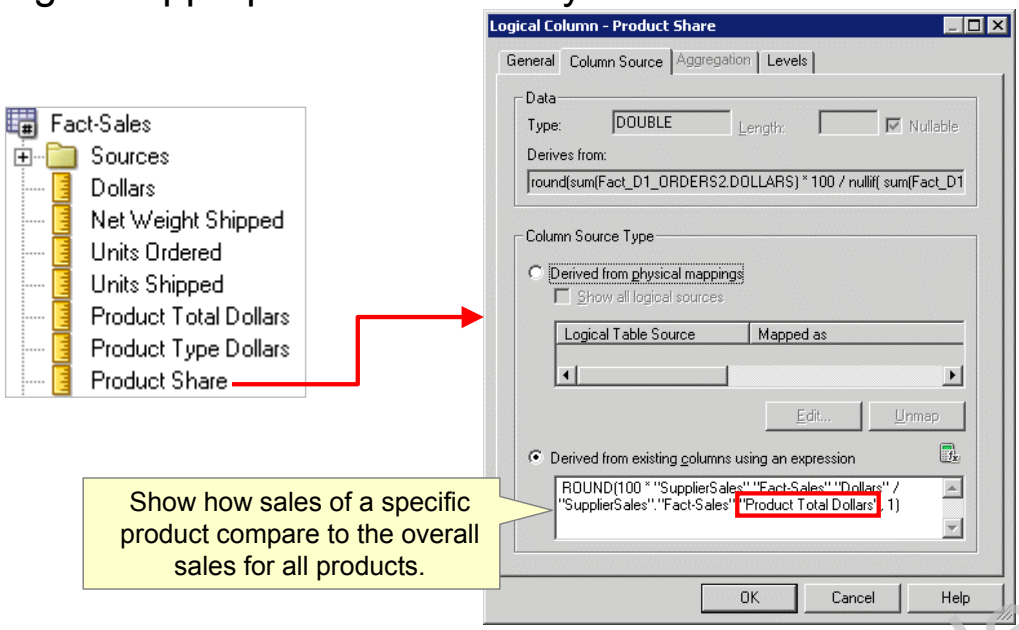
You can use the same techniques to create additional level-based measures. The example in the slide shows how to build a level-based measure named Product Type Dollars at the product type level.

There is no need to repeat the total calculation if the basis is the same for different levels, because the measure is calculated to the appropriate level at run time. However, if you want to show totals for different levels in the hierarchy at the same time in a request, then you must create explicit logical columns for them. For example, you might want to show results for Product Type Dollars and Dollars in the same query.

You also need separate logical columns if the calculation for a level is different from that of another level. If the calculation is the same, you can right-click an existing level-based measure, select Duplicate, rename the measure, and then change the properties. In this example, Product Type Dollars is calculated using the same formula as that used for Product Total Dollars. Both formulas are based on the existing Dollars fact column.

9. Create Share Measures

Create a new logical fact column that calculates the share by dividing the appropriate measure by a total measure.



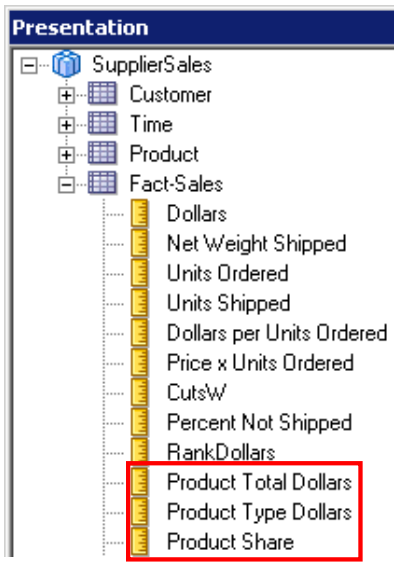
9. Create Share Measures

This slide illustrates an example of a share measure. Share measures are calculated by taking some measure and dividing it by a level-based measure to calculate a percentage.

The share measure in this example, Product Share, uses the level-based measure Product Total Dollars in its formula, which allows you to run an analysis to show how sales of a specific product compare to overall sales for all products. For example, an analysis with the criteria Product, Dollars, and Product Share, filtered on the year 2008, would show total sales for each product and percentage of total sales for each product in 2008.

10. Add Measures to the Presentation Layer

Add new measures to the Presentation layer so that they can be used in queries.

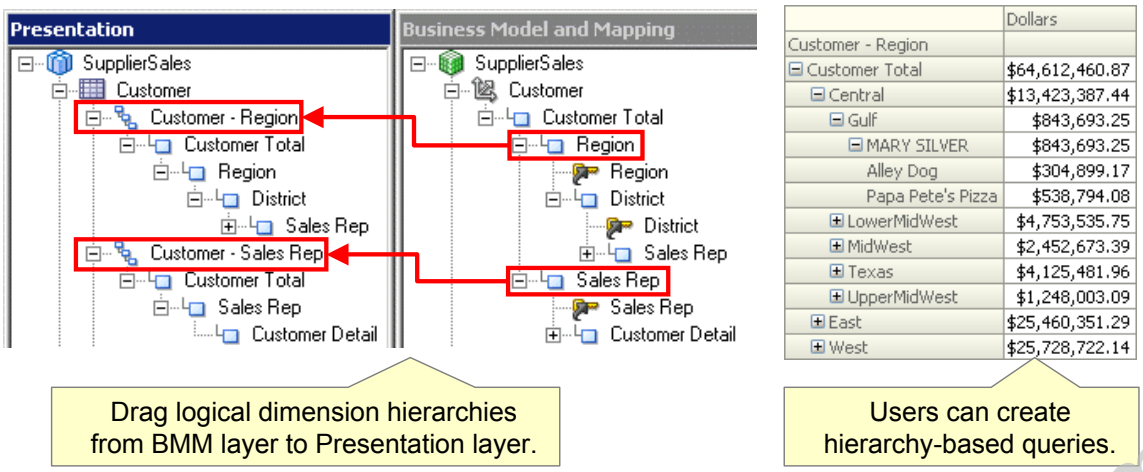


10. Add Measures to the Presentation Layer

If measures are to be available for use in analyses and dashboards, they must be made available by dragging or adding them to the Presentation layer. The screenshot shows level-based and share measures added to the `Fact-Sales` presentation table in the `SupplierSales` subject area.

11. Create Presentation Hierarchies

To create a presentation hierarchy, drag a logical dimension hierarchy from the BMM layer to a table in the Presentation layer.



11. Create Presentation Hierarchies

To create a presentation hierarchy, drag a logical dimension hierarchy from the BMM layer to a table in the Presentation layer. For logical dimensions that contain multiple logical hierarchies, multiple separate presentation hierarchies are created, as shown in the screenshot. You can also drag an individual logical level from the BMM layer to a presentation table to create a presentation hierarchy that is a subset of the logical dimension hierarchy

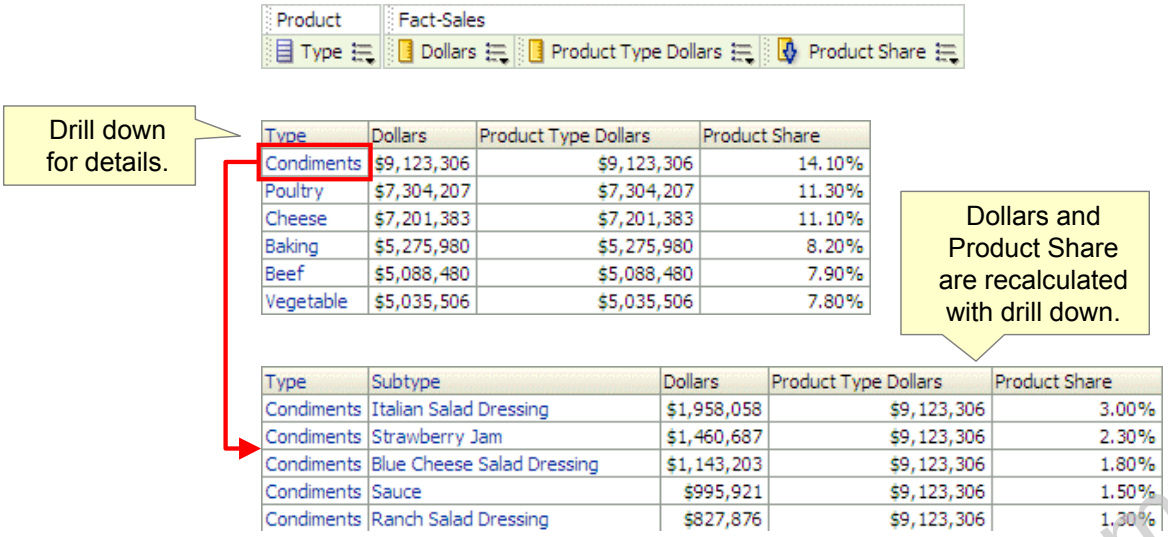
Presentation hierarchies and presentation levels provide an explicit way to expose the multidimensional model in the Oracle BI UI. When presentation hierarchies and levels are defined in the Presentation layer, roll-up information is displayed in the Oracle BI UI, providing users with important contextual information.

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.

12. Test Measures and Hierarchies

Run analyses to test results. Drill down to check relative recalculations.



Type	Dollars	Product Type Dollars	Product Share
Condiments	$9,123,306	$9,123,306	14.10%
Poultry	$7,304,207	$7,304,207	11.30%
Cheese	$7,201,383	$7,201,383	11.10%
Baking	$5,275,980	$5,275,980	8.20%
Beef	$5,088,480	$5,088,480	7.90%
Vegetable	$5,035,506	$5,035,506	7.80%

Type	Subtype	Dollars	Product Type Dollars	Product Share
Condiments	Italian Salad Dressing	$1,958,058	$9,123,306	3.00%
Condiments	Strawberry Jam	$1,460,687	$9,123,306	2.30%
Condiments	Blue Cheese Salad Dressing	$1,143,203	$9,123,306	1.80%
Condiments	Sauce	$995,921	$9,123,306	1.50%
Condiments	Ranch Salad Dressing	$827,876	$9,123,306	1.30%

12. Test Measures and Hierarchies

Build and run analyses with share measures, level-based measures, and hierarchies to test results. Confirm that measures recalculate correctly as you drill down through the hierarchy columns.

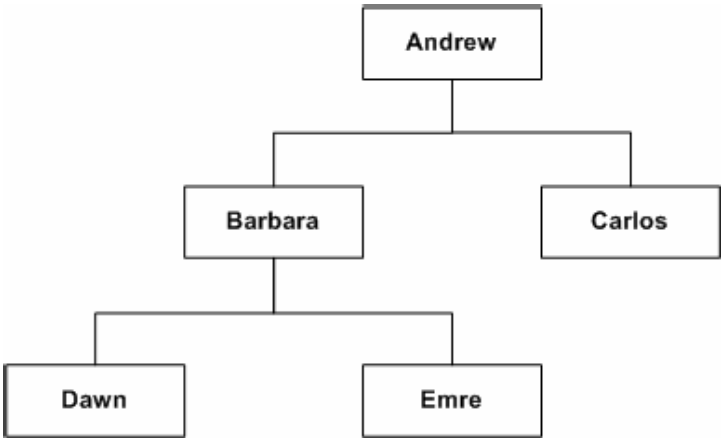
The slide shows an example of an analysis that uses the Product Type Dollars level-based measure and the Product Share measure.

Parent-Child Logical Dimensions

- A parent-child logical dimension is a hierarchy of members that all have the same type.
- This contrasts with level-based hierarchies, in which members of the same type occur only at a single level of the hierarchy.
- The most common real-world example of a parent-child hierarchy is an organizational reporting hierarchy chart, in which 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.

Parent-Child Hierarchy: Example

This chart provides a simple example of a multilevel parent-child hierarchy.



Parent-Child Logical Table

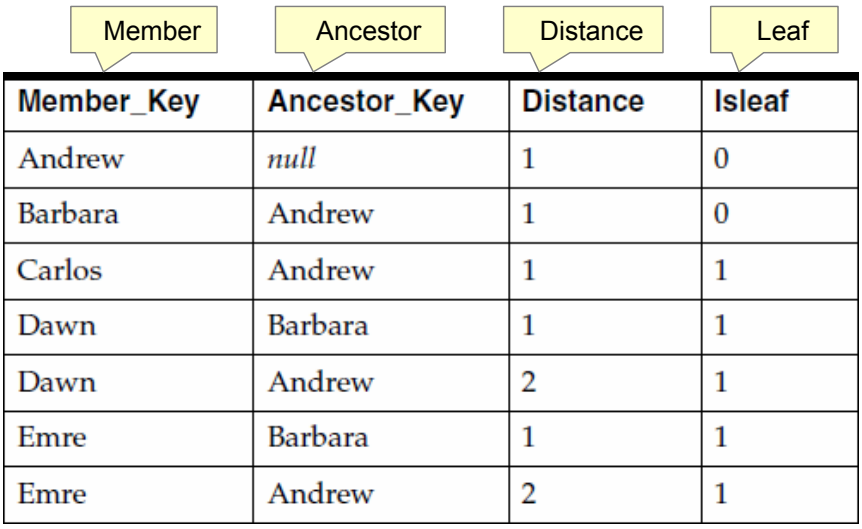
- A parent-child hierarchy is typically based on a single logical table (for example, the `Employees` table).
- Each row in the table contains two identifying keys: one to identify the member itself and the other to identify the "parent" of the member.

Member | Parent

Emp_ID	Mgr_ID
Andrew	*null*
Barbara	Andrew
Carlos	Andrew
Dawn	Barbara
Emre	Barbara

Parent-Child Relationship Table

For each Oracle BI Server parent-child hierarchy defined on a relational table, you must explicitly define the intermember relationships in a separate parent-child relationship table.



Member_Key	Ancestor_Key	Distance	Isleaf
Andrew	*null*	1	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

Parent-Child Relationship Table

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 intermember relationships in a separate parent-child relationship table.

The slide shows an example of a parent-child relationship table with rows that represent the intermember relationships for the employees. 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 (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). A leaf column is a dimension member without descendants.

Creating a Parent-Child Logical Dimension

1. Create a logical dimension object.
2. Set the member key.
3. Set the parent column.
4. Open the Parent-Child Relationship Table Settings dialog box.
5. Enter parent-child relationship table script information.
6. Enter parent-child relationship table details.
7. Preview scripts.
8. Confirm parent-child relationship table settings.
9. Confirm changes to the BMM layer.
10. Confirm changes to the Physical layer.
11. Modify the Physical layer.
12. Modify the BMM layer.
13. Create the presentation hierarchy.
14. Test your work.

Creating a Parent-Child Logical Dimension

In Oracle BI, you generally create scripts to create and populate the parent-child relationship table through an Administration Tool wizard that you can choose to use to define the parent-child hierarchy. You can then use these scripts as often as required to reflect the current state of the data in the parent-child hierarchy.

If you do not choose to use the wizard, then you must have previously created the parent-child relationship table, and you can then manually associate it with the parent-child hierarchy. In the latter case, it is also your responsibility to populate the table with the data required to describe the intermember relationships in the parent-child hierarchy.

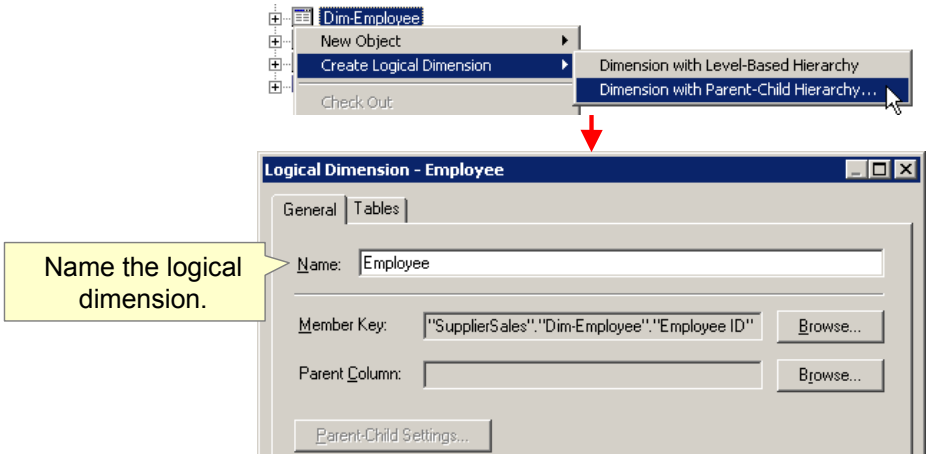
The remaining slides in this lesson show the steps for using the wizard to build the scripts that are used to create and populate the parent-child relationship table.

In the practices for this lesson, you use both techniques: the wizard as well as the manual steps to associate a parent-child hierarchy with a previously built parent-child relationship table.

1. Create a Logical Dimension Object

Do either of the following:

- Right-click the business model object and select New Object > Logical Dimension > Dimension with Parent-Child Hierarchy.
- Right-click the logical dimension table and select Create Logical Dimension > Dimension with Parent-Child Hierarchy.

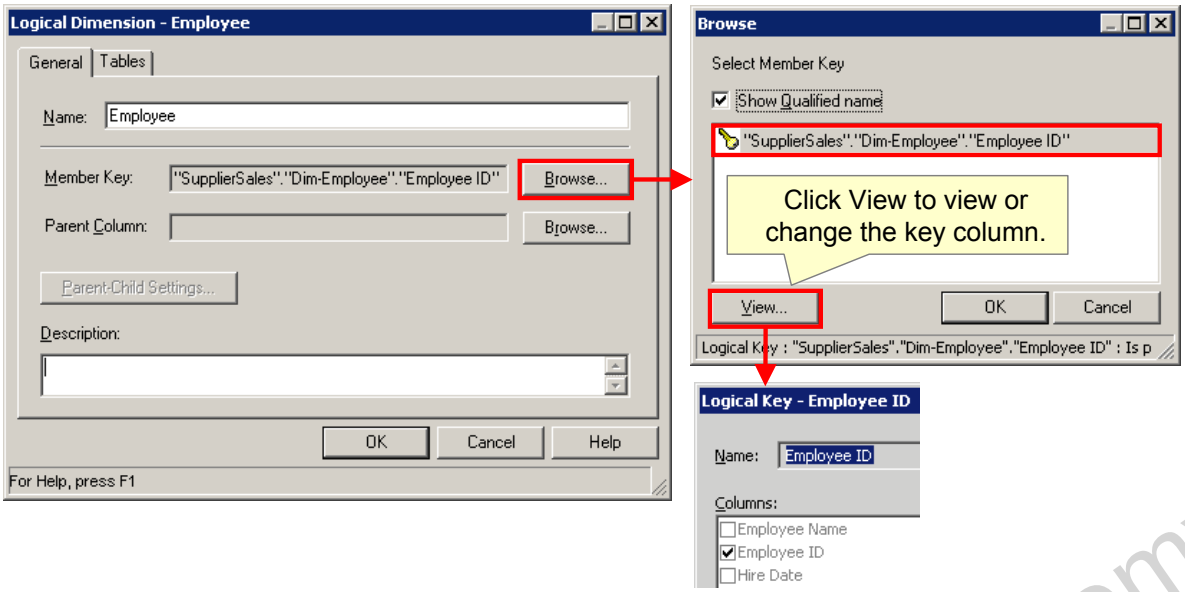


1. Create a Logical Dimension Object

These steps are similar to those used to create a level-based logical dimension.

2. Set the Member Key

Click Browse next to the Member Key field to view or set the member key.

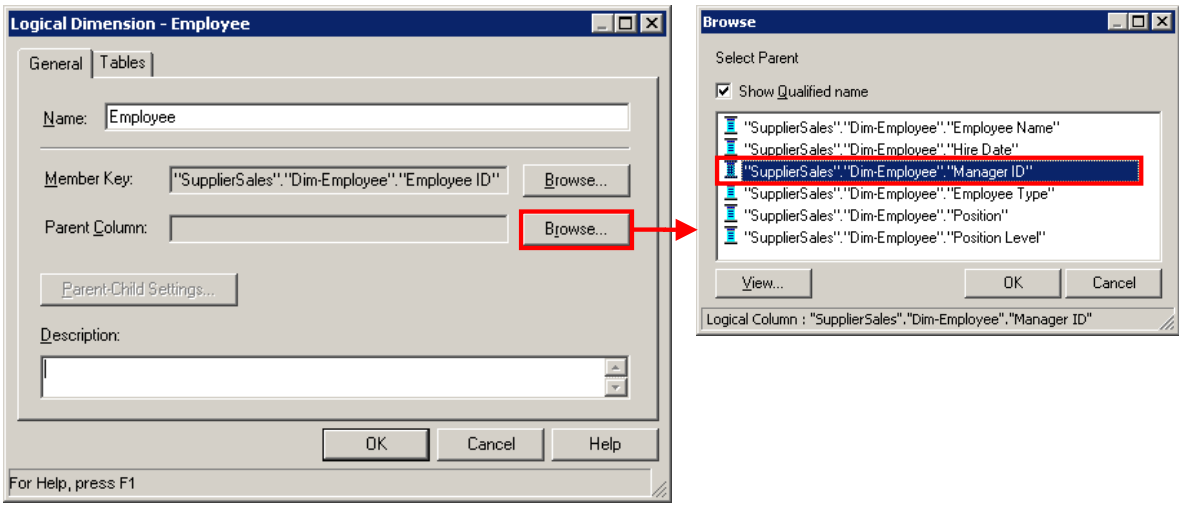


2. Set the Member Key

The member key identifies the logical key in the logical table on which the parent-child logical dimension is based.

3. Set the Parent Column

Click Browse next to the Parent Column field to set the parent column.

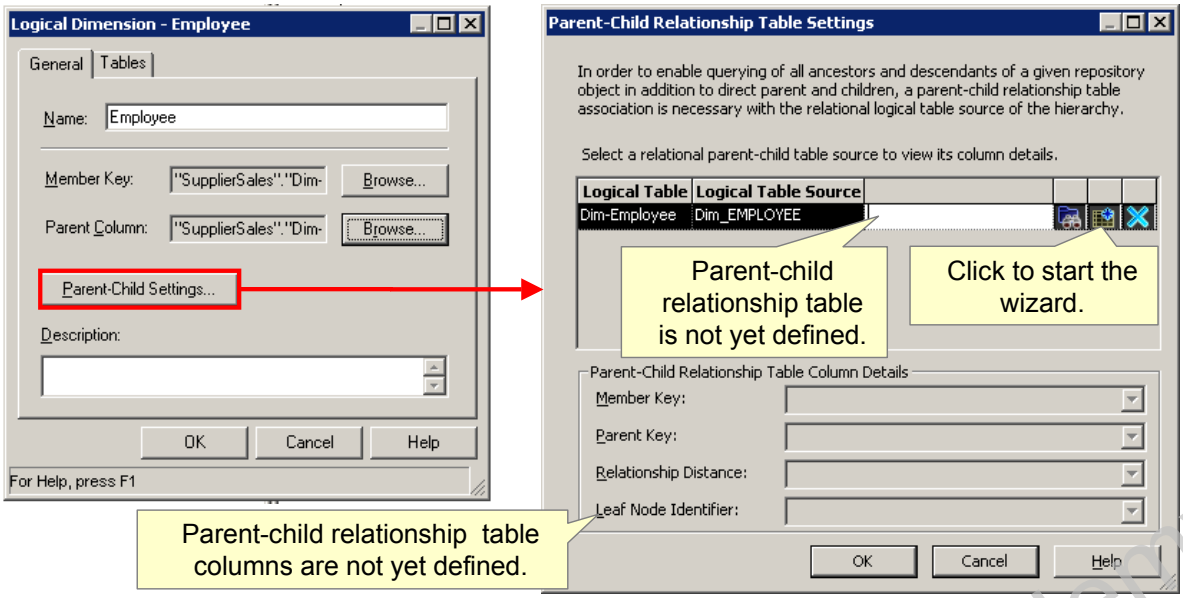


3. Set the Parent Column

The parent column is a column that identifies an ancestor of the member in the logical table. The ancestor may be the parent of the member or a higher-level ancestor. In this example, the ancestor is the Manager ID column.

4. Open the Parent-Child Relationship Table Settings Dialog Box

Click the Parent-Child Settings button to open the Parent-Child Relationship Table Settings dialog box.



4. Open the Parent-Child Relationship Table Settings Dialog Box

At this point, if the logical table that you selected was not from a relational table source, you can click OK to finish the process of creating the dimension. However, because the logical table in this example is from a relational table source, you must continue the dimension definition process to set up the parent-child relationship table for the hierarchy.

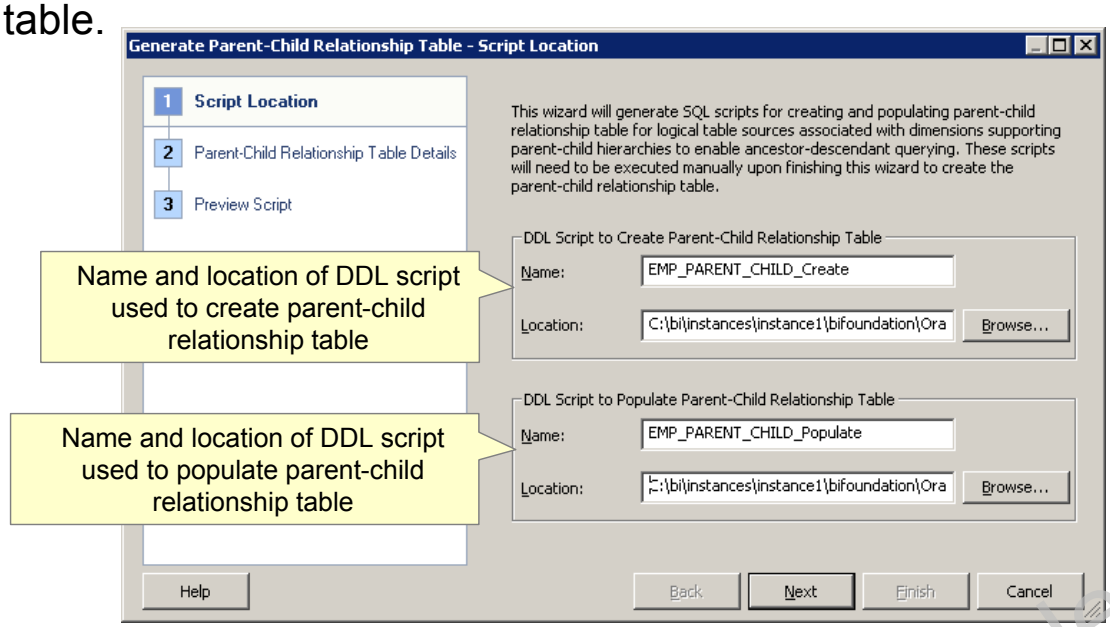
As stated earlier in this lesson, you must define a parent-child relationship table for parent-child hierarchies based on relational tables. When you create the parent-child relationship table, you must choose one of the following methods:

- Use a wizard that generates scripts to create and populate the parent-child relationship table.
- Select a previously created parent-child relationship table.

The slides that follow demonstrate the wizard method to generate scripts to create and populate the parent-child relationship table. In the practices for this lesson, you use both methods. Click the Create Parent-Child Relationship Table icon to start the wizard.

5. Enter Parent-Child Relationship Table Script Information

In the Script Location screen, enter names and locations for the DDL scripts to create and populate the parent-child relationship table.



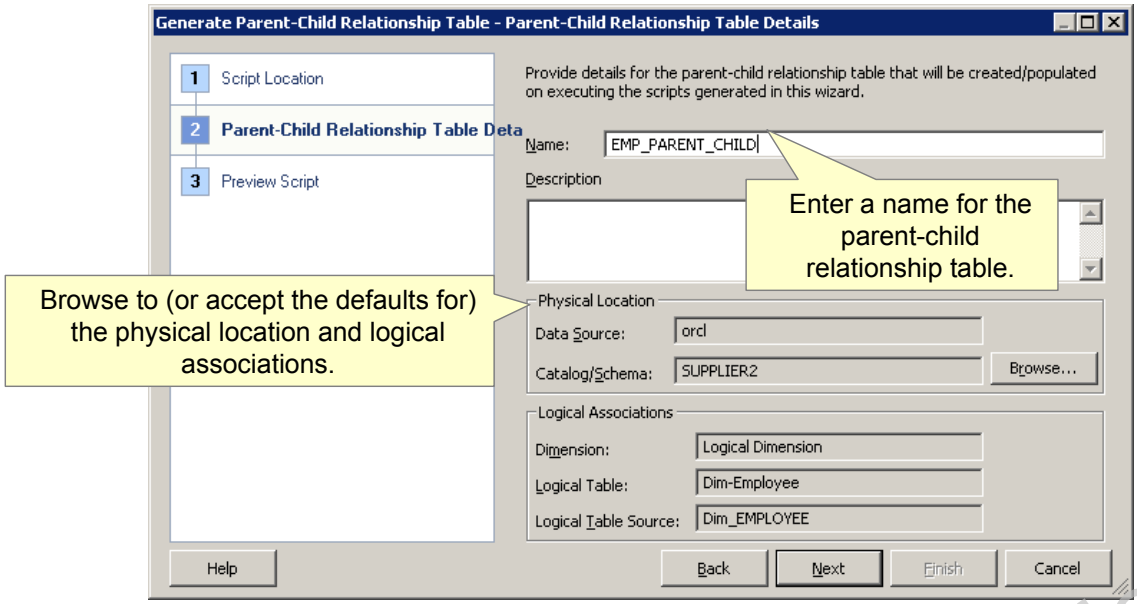
5. Enter Parent-Child Relationship Table Script Information

Script Location is the first screen in the Generate Parent-Child Relationship Table Wizard. The wizard generates SQL scripts for creating and populating the parent-child relationship table.

In the Script Location screen, enter names and locations for the DDL scripts to create and populate the parent-child relationship table. At the end of the wizard, Oracle BI Server stores the scripts in directories chosen during the wizard session. The scripts, when executed, create and populate the parent-child relationship table in the physical data source.

6. Enter Parent-Child Relationship Table Details

In the Parent-Child Relationship Table Details screen, provide details for the table that is generated by the scripts.

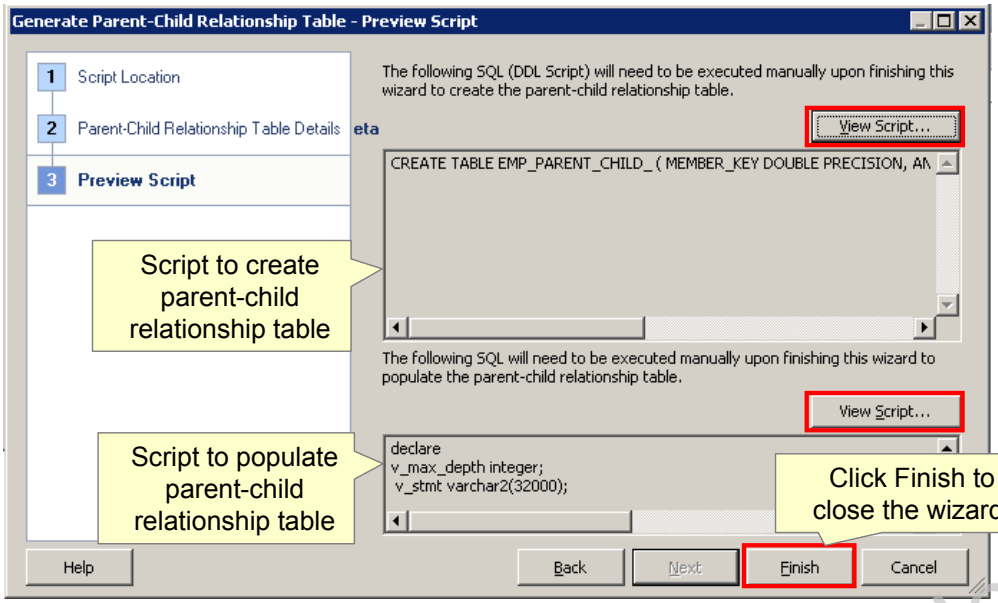


6. Enter Parent-Child Relationship Table Details

In the Parent-Child Relationship Table Details screen, provide details for the parent-child relationship table that is generated by the scripts. Enter a name for the table. By default, the physical location details and logical associations are based on the originating logical table. You can accept the defaults or browse to select new details.

7. Preview Scripts

In the Preview Script screen, click View Script to view either or both of the scripts.



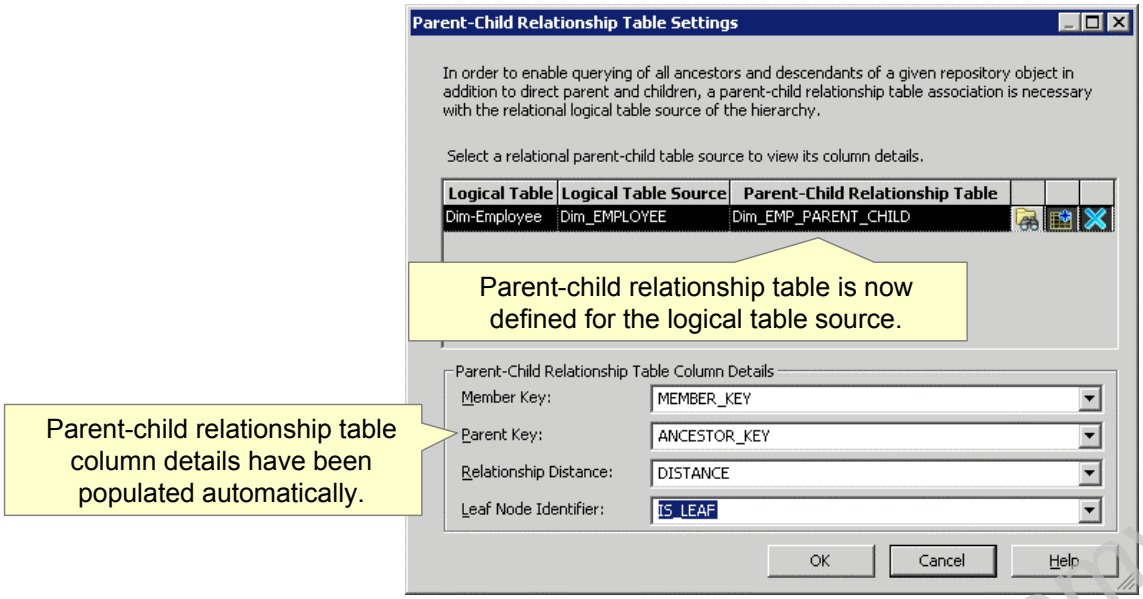
7. Preview Scripts

The Preview Script screen provides an opportunity to preview the scripts before they are created.

Click the View Script button for either the “create” script or the “populate” script. In the window that opens (not shown here), you can search or copy the script.

8. Confirm Parent-Child Relationship Table Settings

When the wizard completes, the Parent-Child Relationship Table details are populated.

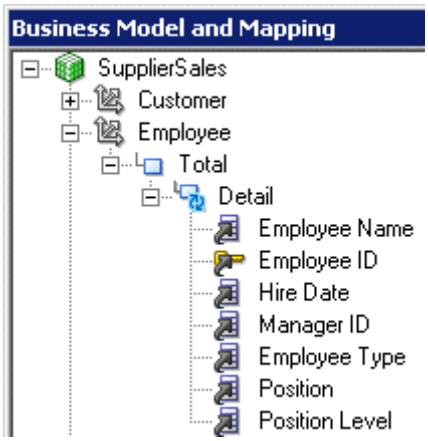


8. Confirm Parent-Child Relationship Table Settings

The parent-child relationship table is now defined for the logical table source. The parent-child relationship table column details have been populated automatically.

9. Confirm Changes to the BMM Layer

When the wizard completes, the parent-child logical dimension is added to the BMM layer.

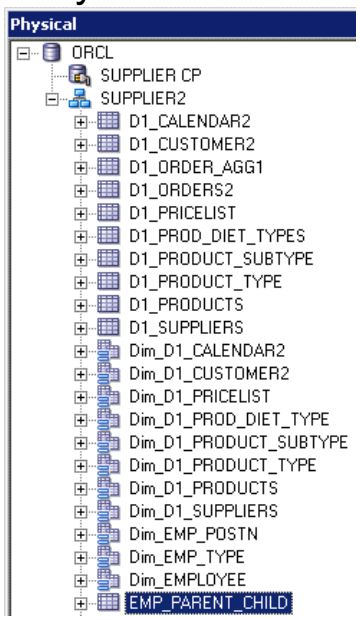


9. Confirm Changes to the BMM Layer

After you close the Parent-Child Relationship Table Settings dialog box and the Logical Dimension dialog box, the parent-child logical dimension hierarchy is added to the business model.

10. Confirm Changes to the Physical Layer

When the wizard completes, the parent-child relationship table is added to the Physical layer.



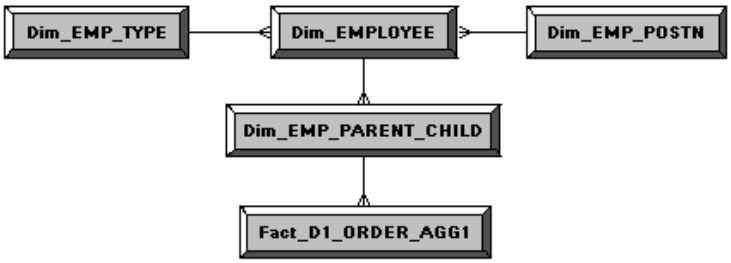
ORACLE

10. Confirm Changes to the Physical Layer

The wizard saves the DDL scripts to the selected locations. The next step (not shown here) is to run the scripts to create and populate this parent-child relationship table in the database.

11. Modify the Physical Layer

After adding the parent-child relationship table to the Physical layer, you must make modifications (such as creating aliases and joins).



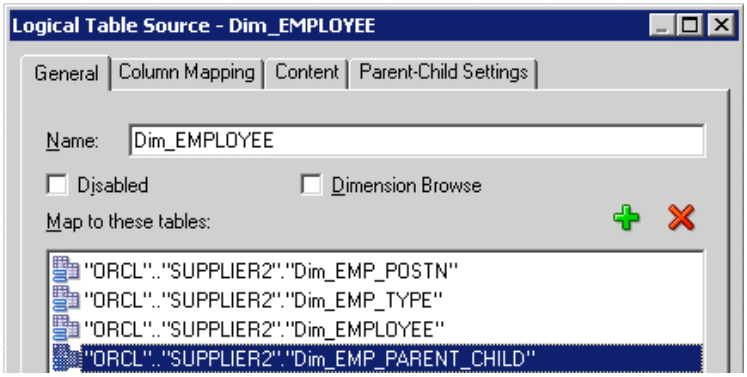
11. Modify the Physical Layer

After adding the parent-child relationship table to the Physical layer, you must make some modifications in both the Physical layer and the BMM layer. For example, you must create join relationships with the other tables related to the parent-child relationship table.

In this example, the `Dim_EMP_PARENT_CHILD` alias table is created and then joined to the `Dim_EMPLOYEE` and `Fact_D1_ORDER_AGG1` tables. Notice the direction of the join relationships. There is a one-to-many join from `Dim_EMPLOYEE` to `Dim_EMP_PARENT_CHILD`, and then a one-to-many join from `Dim_EMP_PARENT_CHILD` to `Fact_D1_ORDER_AGG1`.

12. Modify the BMM Layer

Map the logical table source in the business model to the parent-child relationship table in the Physical layer.



ORACLE

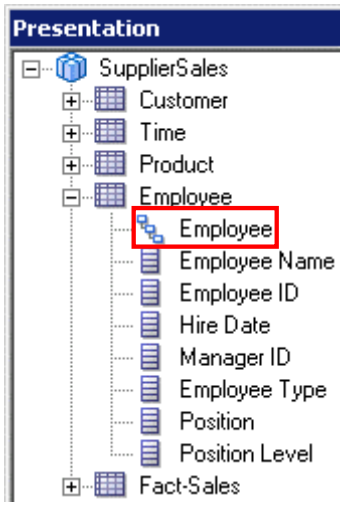
12. Modify the BMM Layer

In the business model, you must map the logical table source in the business model to the parent-child relationship table in the Physical layer.

In this example, the `Dim_EMPLOYEE` logical table source is now mapped to four employee tables, including the `Dim_EMP_PARENT_CHILD` alias parent-child relationship table.

13. Create the Presentation Hierarchy

Add the hierarchy to the corresponding table in the Presentation layer to make the hierarchy available for queries.

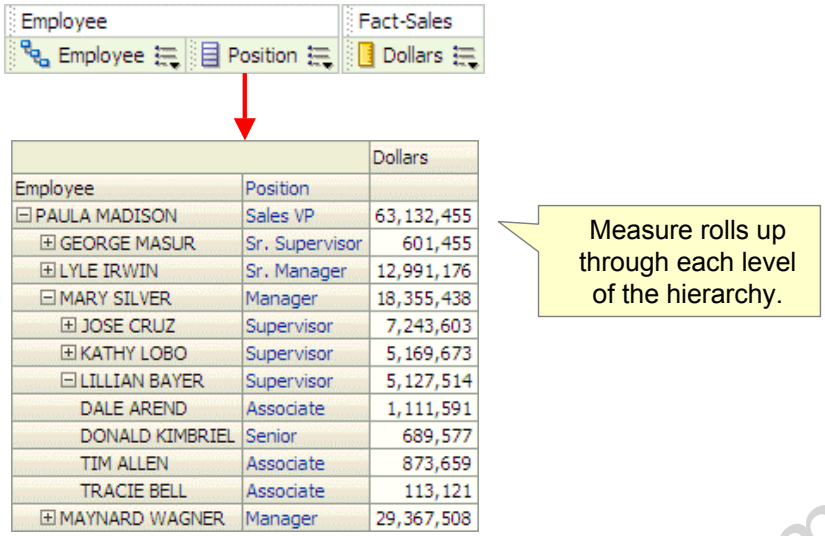


13. Create the Presentation Hierarchy

In this example, the Employee parent-child hierarchy is added to the `Employee` presentation table in the `SupplierSales` subject area.

14. Verify Your Work

Add the hierarchy to an analysis and check the results.



		Dollars
Employee	Position	
PAULA MADISON	Sales VP	63,132,455
GEORGE MASUR	Sr. Supervisor	601,455
LYLE IRWIN	Sr. Manager	12,991,176
MARY SILVER	Manager	18,355,438
JOSE CRUZ	Supervisor	7,243,603
KATHY LOBO	Supervisor	5,169,673
LILLIAN BAYER	Supervisor	5,127,514
DALE AREND	Associate	1,111,591
DONALD KIMBRIEL	Senior	689,577
TIM ALLEN	Associate	873,659
TRACIE BELL	Associate	113,121
MAYNARD WAGNER	Manager	29,367,508

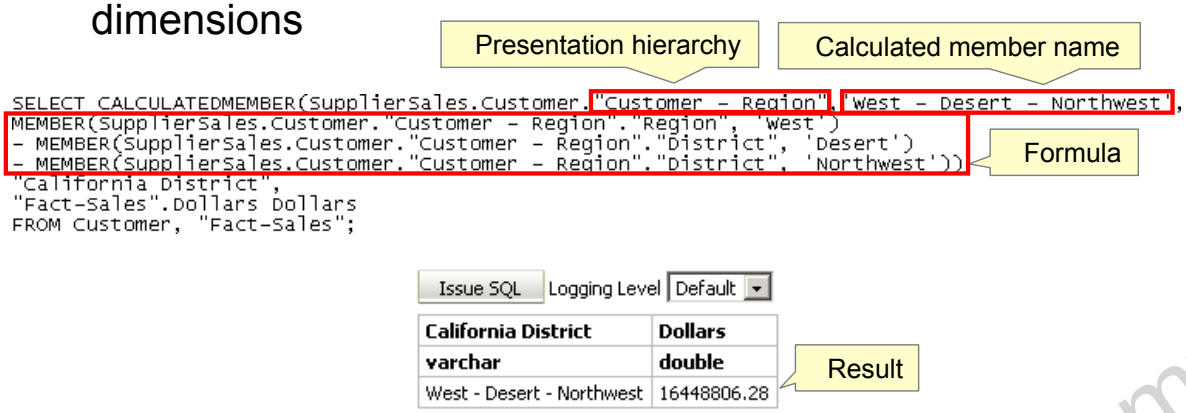
14. Verify Your Work

Create and run analyses and verify that you get the expected results.

In this example, you can expand the Employee hierarchy to see position and dollars data at different levels of the hierarchy. Notice that the dollars measure rolls up through each level of the hierarchy.

Calculated Members

- Are user-defined dimension members whose measure values are calculated at run time
- Are defined within a dimension through a formula that references other members of the same dimension
- Can be defined for a single dimension or across multiple dimensions



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.

Three standard components of a calculated member:

- Presentation hierarchy on which the calculated member is based (in this example, “Customer – Region”)
- Name to identify the calculated member and to distinguish it from other members in the dimension (in this example, “West – Desert – Northwest”)
- Formula used to calculate the calculated member, which consists of one or more examples of a “member clause” connected by standard arithmetic operators. In this example, you calculate total dollars for the West region and then subtract dollars for the Desert and Northwest districts. Because there are only three districts in the West region, the remainder represents total dollars for the California district. `"California District", "Fact-Sales".Dollars Dollars` provides formatting for the SQL results.

Summary

In this lesson, you should have learned how to:

- Create logical dimensions with level-based hierarchies
- Create logical dimensions with parent-child hierarchies
- Create level-based measures
- Create share measures
- Create calculated members

Practice 9-1 Overview: Creating Logical Dimension Hierarchies

This practice covers the following topics:

- Creating logical dimension with level-based hierarchies
- Creating dimension levels
- Associating logical columns with levels
- Specifying level keys

Practice 9-1 Overview: Creating Dimension Hierarchies

A logical dimension represents a hierarchical organization of logical columns belonging to a single logical dimension table. Logical dimensions can exist in the BMM layer and in the Presentation Layer. Adding logical dimensions to the Presentation layer exposes them to users, which enables users to create hierarchy-based queries.

In this practice, you implement three level-based logical dimensions for ABC: Product, Customer, and Time. Creating logical dimensions with level-based hierarchies enables you to build level-based measures, define aggregation rules that vary by dimension, provide drill down on charts and tables in analyses and dashboards, and define the content of aggregate sources.

Practice 9-2 Overview: Creating Level-Based Measures

This practice covers the following topics:

- Creating logical columns to represent level totals
- Associating logical columns with levels

Practice 9-2 Overview: Creating Level-Based Measures

Now that you have created dimensions with hierarchical levels, you use them to implement level-based measures that calculate total dollars at various levels.

Practice 9-3 Overview: Creating Share Measures

This practice covers creating a share measure.

Practice 9-3 Overview: Creating Share Measures

After creating level-based measures, you use them to create a share measure for products.

Practice 9-4 Overview: Creating Dimension-Specific Aggregation Rules

This practice covers creating dimension-specific aggregation rules.

Practice 9-4 Overview: Creating Dimension-Specific Aggregation Rules

ABC wants a measure named AvgDailyDollars, which sums dollar amounts over the Customer and Product dimensions and divides by the number of days in the Period dimension. This measure can be used to compare the average daily dollar amount from month to month when the number of order days in each month varies.

Practice 9-5 Overview: Creating Presentation Hierarchies

This practice covers creating presentation hierarchies that expose logical dimension hierarchies in the Oracle BI user interface.

Practice 9-5 Overview: Creating Presentation Hierarchies

Presentation hierarchies and presentation levels provide an explicit way to expose the multidimensional model in the Oracle BI user interface. When presentation hierarchies and levels are defined in the Presentation layer, roll-up information is displayed in the Oracle BI user interface, providing users with important contextual information. Most importantly, users can create hierarchy-based queries using these objects.

Practice 9-6 Overview: Creating Parent-Child Hierarchies

This practice covers creating logical dimensions with parent-child hierarchies.

Practice 9-6 Overview: Creating Parent-Child Hierarchies

A parent-child hierarchy is a hierarchy of members that all have the same type. This contrasts with level-based hierarchies, in which members of the same type occur at only a single level of the hierarchy. The most common real-world example of a parent-child hierarchy is an organizational reporting hierarchy chart, which has the following characteristics:

- 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.

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 intermember relationships in a separate parent-child relationship table.

Practice 9-7 Overview: Using Calculated Members

This practice covers creating a user-defined dimension member whose measure values are calculated at run time.

Practice 9-7 Overview: Using 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 by using a formula that references other members of the same dimension. Calculated members can be defined for a single dimension or across multiple dimensions.

In Oracle BI there are also several places where you can issue logical SQL. In this practice, you build `CALCULATEDMEMBER` SQL statements and run them on the Issue SQL page of the Oracle BI Administration tab.

10

Using Aggregates

Objectives

After completing this lesson, you should be able to:

- Describe aggregate tables and their purpose in dimensional modeling
- Model aggregate tables
- Set the number of elements for a hierarchy level
- Use the Aggregate Persistence Wizard to create aggregates

Business Challenge

- Data in fact and dimension sources is stored at the lowest level of detail.
- Data often needs to be rolled up or summarized during analysis.
- Based on the amount of data, performing calculations at the time of the query can be resource intensive and can delay results to the user.

Business Solution: Aggregate Tables

- Aggregate tables store pre-computed measures that have been aggregated over a set of dimensional attributes.
- Using aggregate tables is a popular technique for speeding up response time.

Detailed fact data table; many rows; longer time to read and calculate

Invoice	Customer Key	Period Key	Product Key	Dollars
122222	1001	20080102	100	100
133333	1001	20090105	200	200
144444	1002	20080505	300	100
155555	1002	20080601	400	20
166666	1005	20080101	600	20

Aggregate fact table; fewer rows; quicker to read; pre-calculated

Product Type	Sales Rep	Month	Total Dollars
112	JCRUZ	200902	1500
110	AZIFF	200903	1200
111	MWEST	200904	1100

Summarized

Business Solution: Aggregate Tables

Aggregate tables store pre-computed results, which are measures that have been aggregated (typically summed) over a set of dimensional attributes. Using aggregate tables is a very popular technique for speeding up query response time in decision support systems. This eliminates the need for run-time calculations and delivers faster results to users.

Note that these are real physical tables. The calculations are done ahead of time and the results are stored in the tables. The key point is that the aggregate table should have fewer rows than the nonaggregate table and, therefore, processing should be quicker.

Oracle BI Aggregate Navigation

Enables queries to use the information stored in aggregate tables automatically:

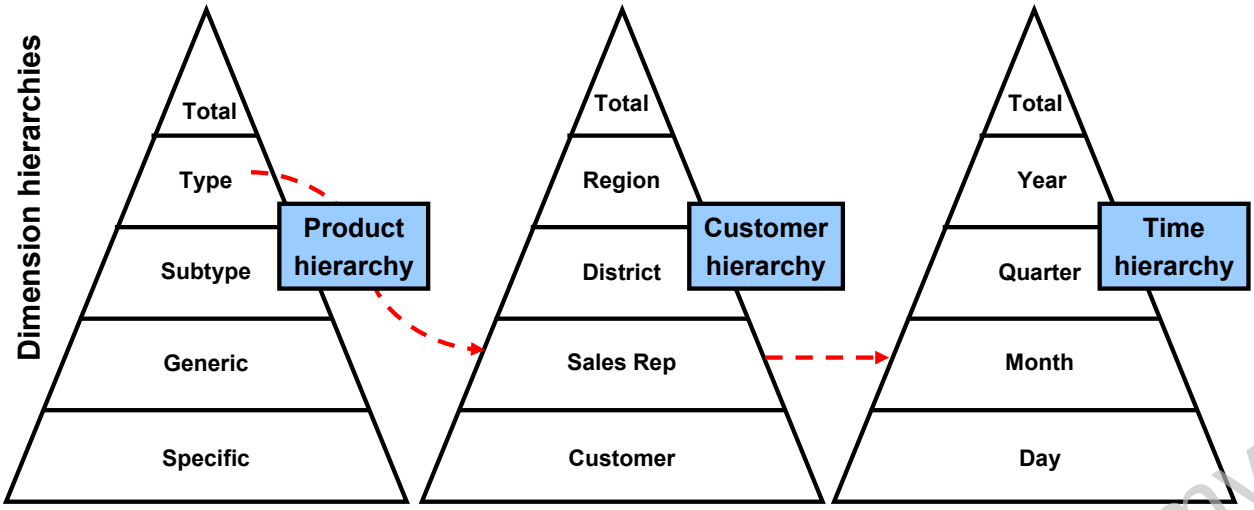
- Oracle BI Server decides which tables provide the fastest answers.
- Metadata must be configured for aggregate navigation.

Oracle BI Aggregate Navigation

If you are writing SQL queries or using a tool that understands only which physical tables exist (and not their meaning), putting aggregate tables to good use becomes more difficult as their number increases. The aggregate navigation capability of Oracle BI Server, however, allows queries to use the information stored in aggregate tables automatically, without query authors or query tools having to specify aggregate tables in their queries. Oracle BI Server allows you to concentrate on asking the right business question; the server "decides" which tables provide the fastest answers. For Oracle BI Server to have enough information to navigate to aggregate tables, you need to configure certain metadata in the repository.

Aggregated Facts

Aggregated sales fact table columns store pre-computed results at a given set of levels.



Aggregated Facts

Each aggregate table column contains data at a given set of levels. For example, an aggregated sales fact table might contain a pre-computed sum of the revenue for each product type for each sales representative during each month.

The graphic indicates that there is more data at lower levels in a hierarchy. As you move higher in the hierarchy, there is less data because the results are aggregated.

Modeling Aggregates

Model aggregate tables in the same way that you model other source data.

- Physical layer
  - Create data source connection.
  - Import physical sources.
  - Create physical joins.
- Business Model and Mapping layer
  - Add sources to logical tables. New step
  - Specify aggregation content.
- Presentation layer
  - No changes necessary; aggregate navigation is independent of the Presentation layer objects.

Modeling Aggregates

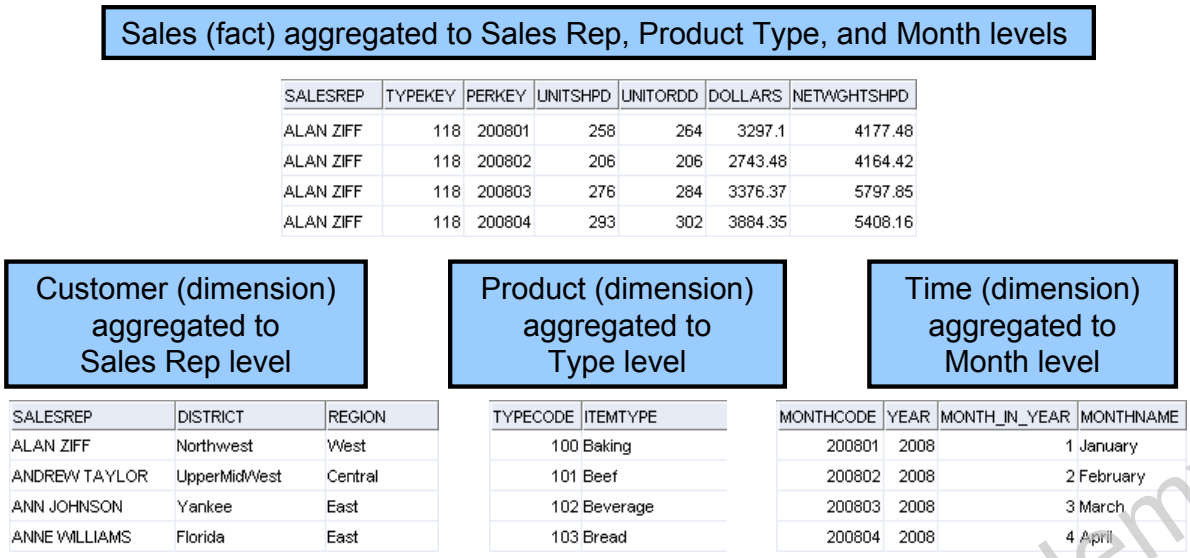
This slide outlines the steps for implementing aggregates. Modeling aggregate tables is similar to modeling other source data. You should be familiar with most of the steps because you were exposed to them previously in the course.

The new step is specifying aggregation content. This involves identifying the aggregation level of the data contained in the aggregate table so that Oracle BI Server can determine when to use it for queries.

These steps assume that logical dimension hierarchies have already been created. Note that changes to the Presentation layer are not needed. Aggregate navigation occurs at the logical business model layer and is independent of Presentation layer objects.

ABC Example

- Uses prebuilt aggregate tables to improve performance
- Must have matching levels of aggregation for fact and dimensions



SALESREP	TYPEKEY	PERKEY	UNITSHPD	UNITORDD	DOLLARS	NETWGHTSHPD
ALAN ZIFF	118	200801	258	264	3297.1	4177.48
ALAN ZIFF	118	200802	206	206	2743.48	4164.42
ALAN ZIFF	118	200803	276	284	3376.37	5797.85
ALAN ZIFF	118	200804	293	302	3884.35	5408.16

SALESREP	DISTRICT	REGION
ALAN ZIFF	Northwest	West
ANDREW TAYLOR	UpperMidWest	Central
ANN JOHNSON	Yankee	East
ANNE WILLIAMS	Florida	East

TYPECODE	ITEMTYPE
100	Baking
101	Beef
102	Beverage
103	Bread

MONTHCODE	YEAR	MONTH_IN_YEAR	MONTHNAME
200801	2008	1	January
200802	2008	2	February
200803	2008	3	March
200804	2008	4	April

ABC Example

ABC has already built its logical dimension hierarchies. Now ABC wants to add sources that contain pre-computed aggregations. You need to specify the level of aggregation for each source by using logical levels. The database administrator for ABC has already created the necessary aggregate tables:

- Sales facts aggregated to the sales rep, product type, and month levels
- Sales Rep aggregate with one row for each sales representative, which can be considered an aggregation of the Customer dimension to the sales rep level
- Product type aggregate with one row for each product type, which can be considered an aggregation of the Product dimension to the type level
- Month aggregate with one row for each year and month combination, which can be considered an aggregation of the Time dimension to the month level

Aggregate dimension tables are sometimes called *upper level attribute* tables because they contain only the attributes at the upper levels of the dimension.

Steps to Implement Aggregate Navigation

1. Import tables.
2. Create joins.
3. Create the fact logical table source and mappings.
4. Specify the fact aggregation content.
5. Specify content for the fact detail source.
6. Create the dimension logical table source and mappings.
7. Specify the dimension aggregation content.
8. Specify content for the dimension detail source.
9. Test results for levels stored in aggregates.
10. Test results for data above or below levels.

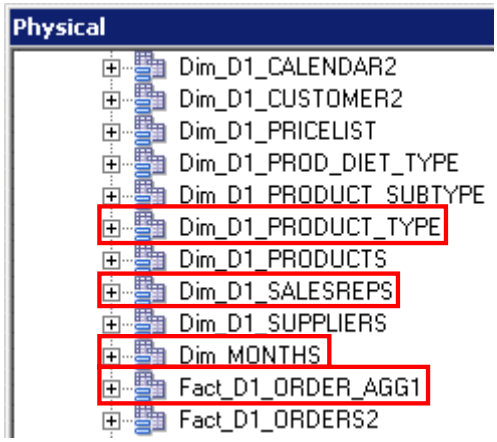
Steps to Implement Aggregate Navigation

These are the high-level steps to implement aggregate navigation. Each step is covered in detail in subsequent slides.

Most of these steps can be automated through the use of the Aggregate Persistence Wizard, which is presented in detail later in this lesson.

1. Import Tables and Create Aliases

Import fact and dimension aggregates and create aliases.



1. Import Tables and Create Aliases

Use known techniques to import the fact and dimension aggregate tables to the Physical layer and create aliases. You are importing both aggregate fact and aggregate dimension tables because you need to create logical dimension sources at the same level of detail as the fact sources.

2. Create Joins

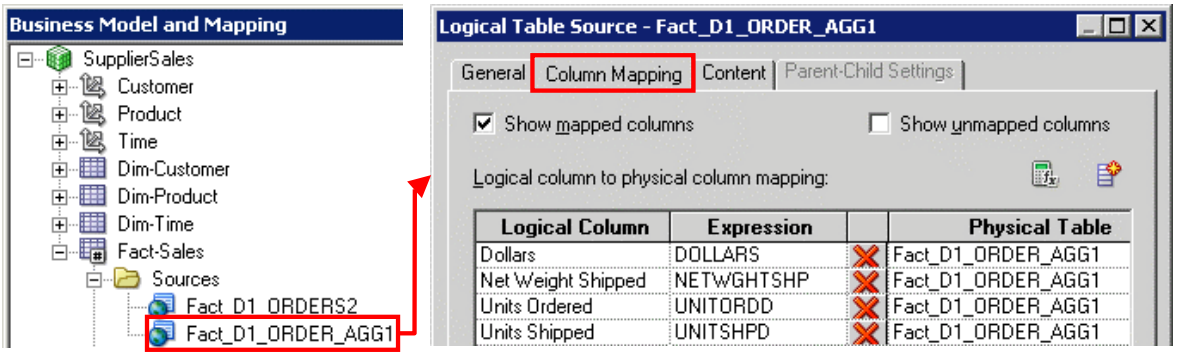
Use the Physical Diagram to create joins between the aggregate fact table alias and the aggregate dimension table aliases.

2. Create Joins

Use known techniques to create physical joins.

3. Create the Fact Logical Table Source and Mappings

Create the new aggregate logical table source in the existing logical fact table and map the columns.



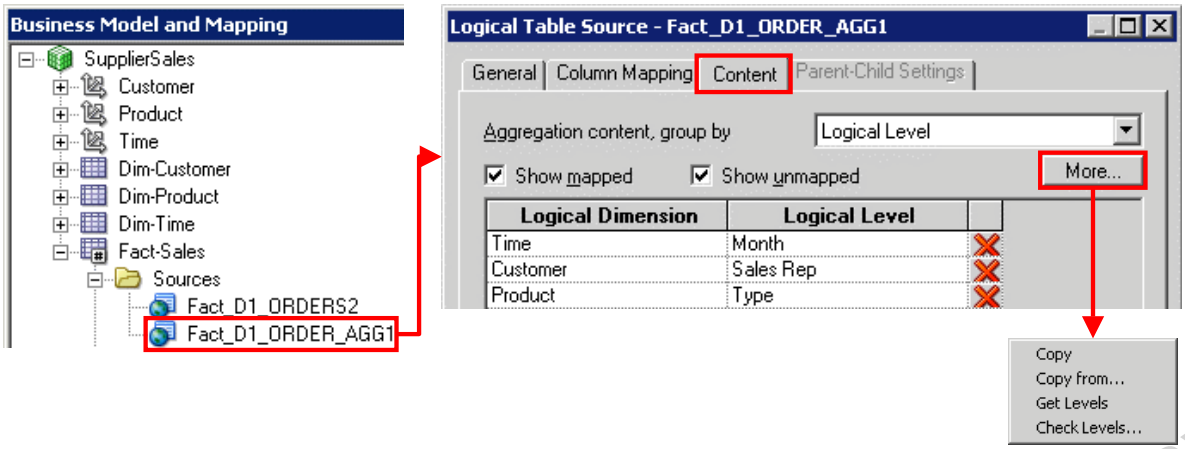
3. Create the Fact Logical Table Source and Mappings

Use known techniques to create a new logical table source within the current logical fact table that points to the aggregate table. For example, drag physical columns from the fact aggregate physical table to corresponding logical columns in the business model to map existing logical columns to the new logical table source. In this example, the new logical table source is named `Fact_D1_ORDER_AGG1`.

Note that these four columns now map to both the `Fact_D1_ORDERS2` table and the `Fact_D1_ORDER_AGG1` table. In the next step, you configure the model to choose the appropriate table during a query based on how content is specified on the Content tab.

4. Specify the Fact Aggregation Content

Specify the aggregation content of the aggregate fact logical table source so that Oracle BI Server knows what level of data is stored in the aggregate tables.



4. Specify the Fact Aggregation Content

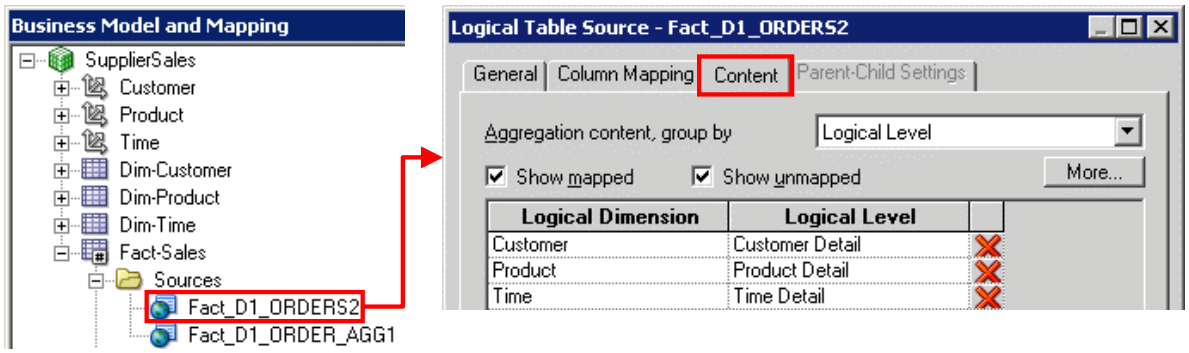
Use the Content tab of the Logical Table Source dialog box to specify the aggregation content of the new aggregate logical table source. You set aggregation content for the fact table to the corresponding levels in the dimension hierarchies. Later, when a user queries against a particular level, Oracle BI Server will know that it should access the aggregate tables instead of the detail tables. In a subsequent step, you set similar levels for the dimension table aggregate sources.

For example, if a user queries for total sales by Month by Sales Rep, the server accesses the `Fact_D1_ORDER_AGG1` aggregate fact table and the corresponding aggregate dimension tables, `Dim_MONTHS` and `Dim_D1_SALESREP`. If a user queries for a level lower than the levels specified here, for example Day instead of Month, or Customer instead of Sales Rep, the server accesses the detail tables (`Fact_D1_ORDERS2`, `Dim_D1_CALENDAR2`, `Dim_D1_CUSTOMER2`). If a user queries for a higher level (Year instead of Month, District instead of Sales Rep), the aggregate tables are used as well, because whenever a query is run against a logical level or above, the aggregate tables are used.

Hint: Click the More button in Logical Table Source dialog box to have the server help determine the levels.

5. Specify Content for the Fact Detail Source

Set the levels of the fact detail source to the lowest in the hierarchies.



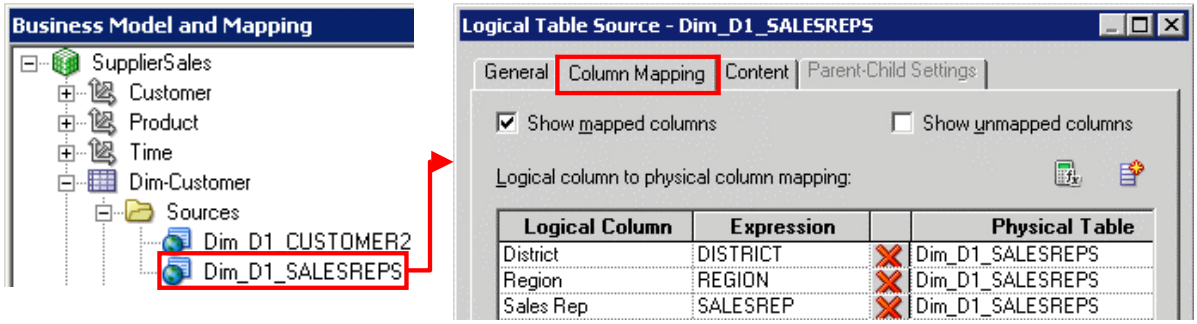
5. Specify Content for the Fact Detail Source

It is considered a best practice to set the levels of the fact detail source to the lowest in the hierarchies. This is because you want the server to access the detail tables when queries are against levels lower than those specified for the aggregate tables.

It is also good practice to specify the content of all sources for documentation purposes. This helps prevent another administrator from interpreting the lack of an aggregation content statement as an inadvertent omission of information.

6. Create the Dimension Logical Table Source and Mappings

Create the new aggregate logical table source in the existing logical dimension tables and map the columns.



6. Create the Dimension Logical Table Source and Mappings

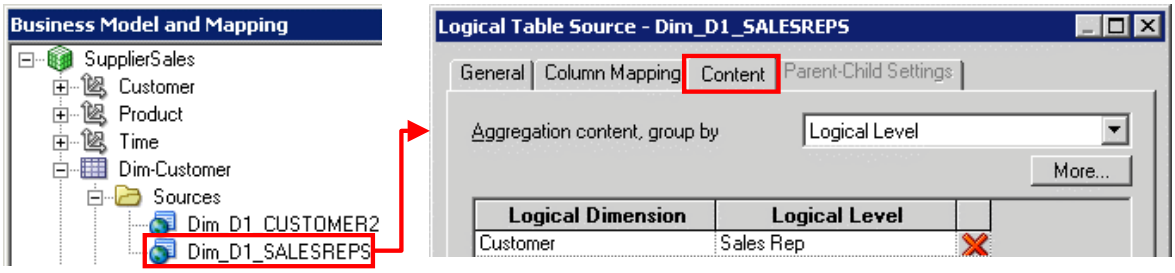
Use known techniques to create a new logical table source within the current logical dimension tables that points to the aggregate tables.

This example shows the `Dim-Customer` logical dimension table. A new logical table source named `Dim_D1_SALESREP` is added. It maps to the `Dim_D1_SALESREP` physical aggregate table. The logical columns District, Region, and Sales Rep now map to two physical tables, `Dim_D1_CUSTOMER2` and `Dim_D1_SALESREP`.

In the ABC example, you need to create similar mappings for all three logical dimension tables (not shown here).

7. Specify the Dimension Aggregation Content

Specify the aggregation content of the new logical table source so that Oracle BI Server knows what level of data is stored in the aggregate tables.



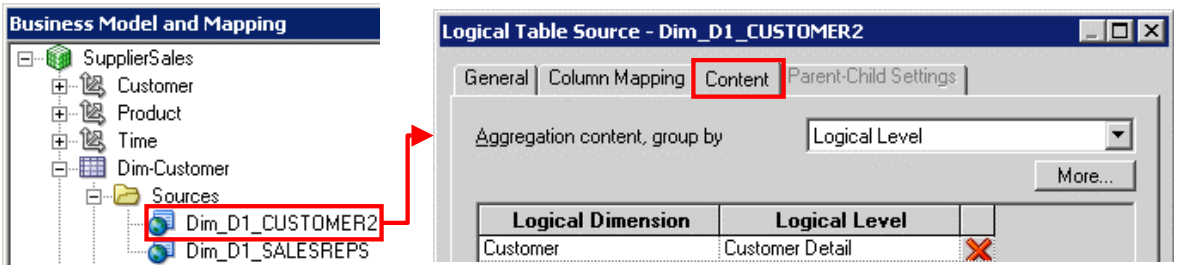
7. Specify the Dimension Aggregation Content

Specify the aggregation content for the aggregate logical table source for the `Dim-Customer` table so that Oracle BI Server knows what level of data is stored in the aggregate tables. Recall that the `Dim_D1_SALESREPS` table contains data at the Sales Rep level within the Customer hierarchy.

In the ABC example, you need to set content levels for all three aggregate logical table sources (not shown here).

8. Specify Content for the Dimension Detail Source

Set the levels of the dimension detail source to the lowest in the hierarchies.



8. Specify Content for the Dimension Detail Source

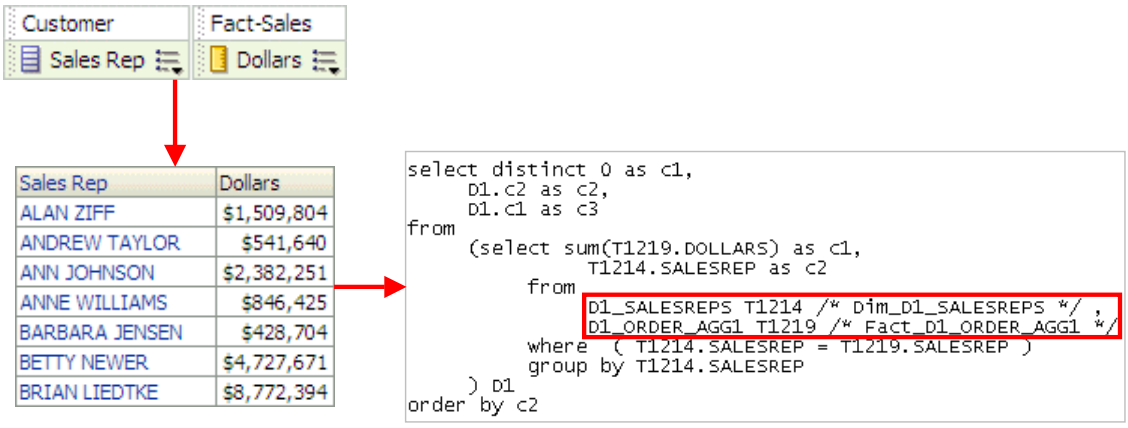
It is good practice to set the levels for the dimension detail logical table source to the lowest in the hierarchies. This is because you want the server to access the detail tables when queries are against levels lower than those specified for the aggregate tables.

It is also good practice to specify the content of all sources for documentation purposes, because another administrator could interpret the lack of an aggregation content statement as an inadvertent omission of information.

In the ABC example, you need to set content levels for all three detail logical table sources (not shown here).

9. Test Results for Levels Stored in Aggregates

Run analyses and inspect the query log to ensure that the aggregate tables are accessed as expected.

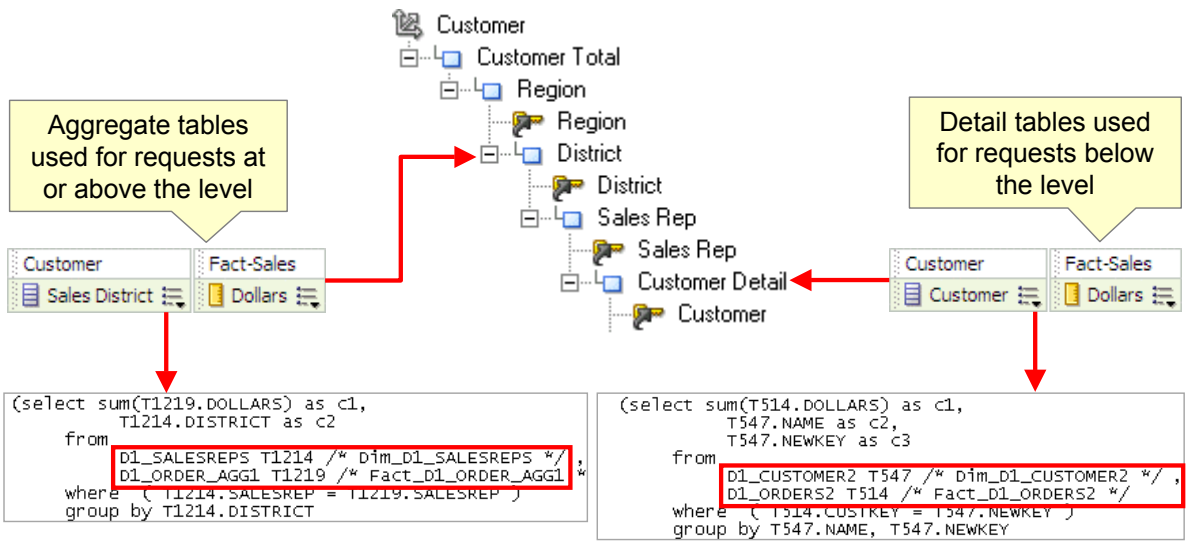


9. Test Results for Levels Stored in Aggregates

In this example, the analysis requests data at the levels stored in the aggregates.

As expected, the Sales Rep data is retrieved from the dimension aggregate, `Dim_D1_SALESREPS`, and the dollars data is retrieved from the fact aggregate, `Fact_D1_ORDER_AGG1`.

10. Test Results for Data Above or Below Levels

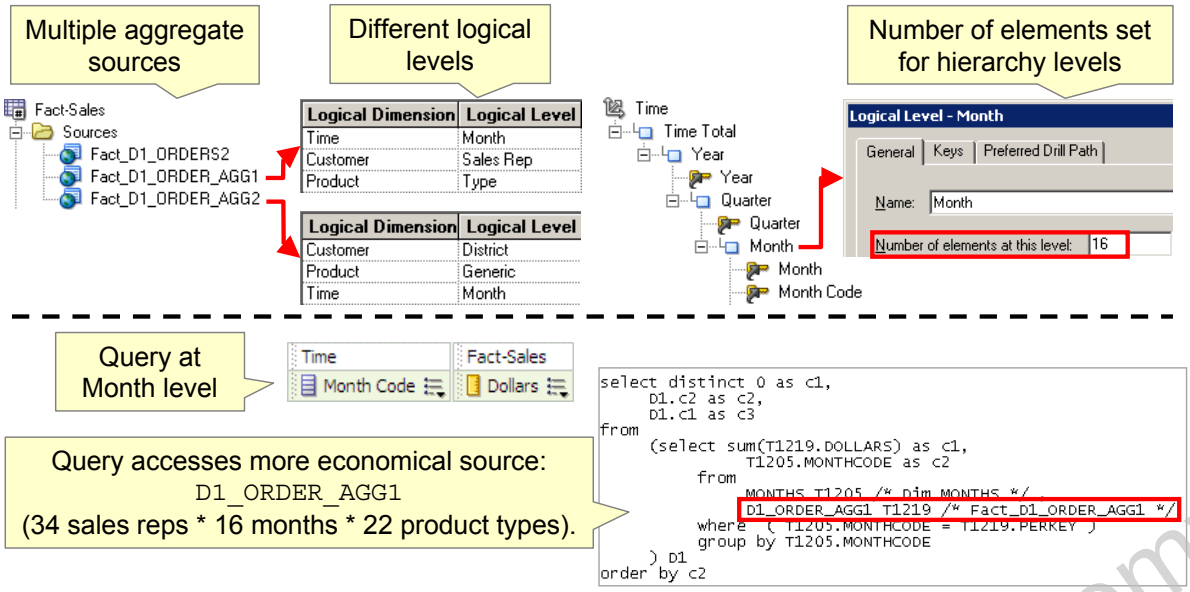


10. Test Results for Data Above or Below Levels

In this example, the analysis requests data at levels above or below those stored in the aggregates. When data is requested for Sales District, which is above the Sales Rep level, the `Dim_D1_SALESREPS` and `Fact_D1_ORDER_AGG1` aggregate tables are still used. When data is requested for Customer, which is below the Sales Rep level, the `Dim_D1_CUSTOMER2` and `Fact_D1_ORDERS2` detail tables are used.

Setting the Number of Elements

Used by OBI Server when there are two or more aggregate sources that can be accessed by a query



Setting the Number of Elements

The number of elements is used by Oracle BI Server when picking aggregate sources. Setting the number of elements is necessary only when there are two or more aggregate sources that could be accessed by an Oracle BI query.

Aggregate fact sources are accessed based on a combination of the fields selected as well as the number of elements of the levels in the logical dimensions to which they map. The number does not have to be exact, but ratios of numbers from one logical level to another should be accurate.

The screenshot shows only the number of elements set for the Month level (16). In the practice for this lesson, you set the number of elements for all levels in all of the logical dimension hierarchies. By adjusting the number of elements, you can alter the aggregate fact source selected by Oracle BI Server.

Setting the Number of Elements (continued)

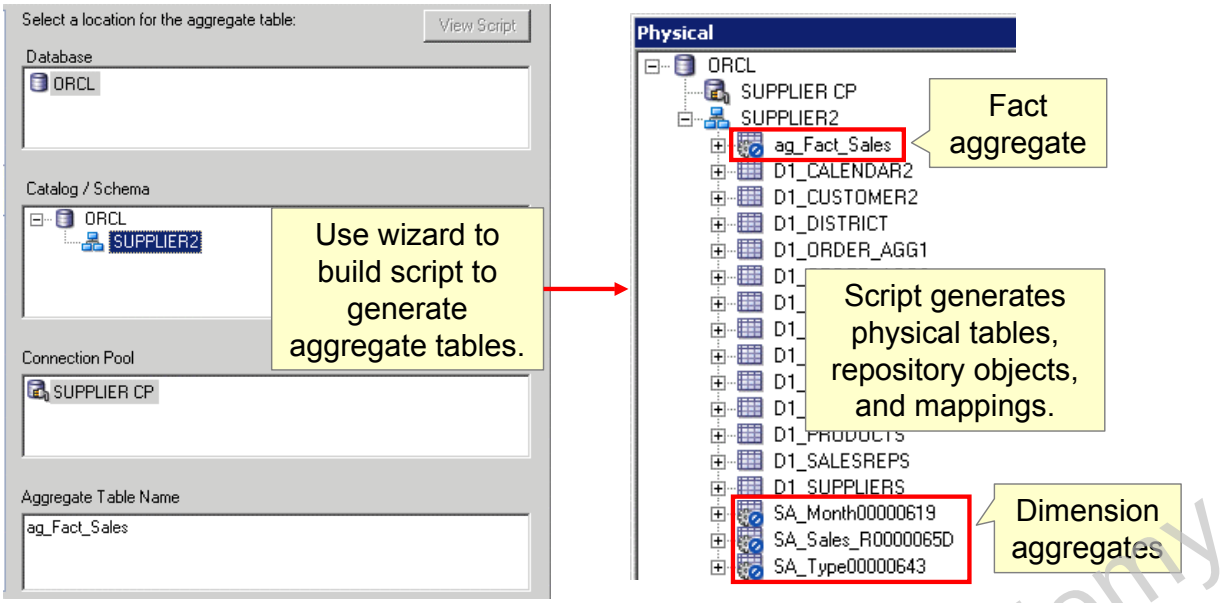
In this example, there are two aggregate sources for the Fact-Sales logical table: `Fact_D1_ORDER_AGG1` and `Fact_D1_ORDER_AGG2`. Logical levels are set differently for each aggregate source; however, both sources have logical levels set to Month for the Time logical dimension.

Despite the fact that both aggregate sources have their aggregation content set to the Month level for the Time logical dimension, the query uses `Fact_D1_ORDER_AGG1`. This is because Oracle BI Server determines that it is more economical to access `Fact_D1_ORDER_AGG1` instead of `Fact_D1_ORDER_AGG2` based on the levels specified in the dimension hierarchies.

For example, to access `Fact_D1_ORDER_AGG2`, it calculates 35,712 potential rows (12 districts * 16 months * 186 generic products). To access `Fact_D1_ORDER_AGG1`, it calculates 11,968 potential rows (34 sales reps * 16 months * 22 product types). Therefore, `Fact_D1_ORDER_AGG1` is the more economical aggregate source.

Aggregate Persistence Wizard

Automates the creation of physical aggregate tables and their corresponding objects in the repository



Aggregate Persistence Wizard

The traditional process of creating aggregates for Oracle BI Server queries is manual. It can be tedious, requiring complicated data definition language (DDL) and data manipulation language (DML) scripts to be written for creating tables in the databases involved. Additionally, these aggregated tables need to be mapped into the repository metadata to be available for queries. This is a time-consuming and, possibly, error-prone process.

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. The Aggregate Persistence Wizard enables the administrator to automate the creation of physical aggregate tables and their corresponding objects in the repository.

Aggregate Persistence Wizard Steps

1. Open the Aggregate Persistence Wizard.
2. Specify a file name and location.
3. Select the business model and measures.
4. Select dimensions and levels.
5. Select the connection pool, container, and name.
6. Review the aggregate definition.
7. View the complete aggregate script.
8. Verify that the script is created.
9. Execute the aggregate creation job.
10. Verify aggregates in the Physical layer.
11. Verify aggregates in the BMM layer.
12. Verify aggregates in the database.
13. Verify your work.

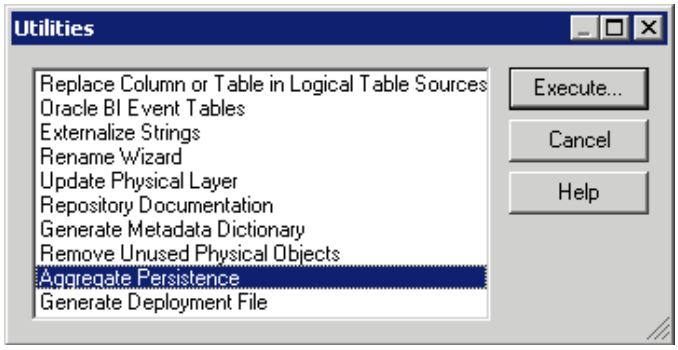
ORACLE

Aggregate Persistence Wizard Steps

This slide lists the steps for building aggregates by using the Aggregate Persistence Wizard. Each step is presented in detail in the following slides.

1. Open the Aggregate Persistence Wizard

Select Tools > Utilities > Aggregate Persistence and click the Execute button.

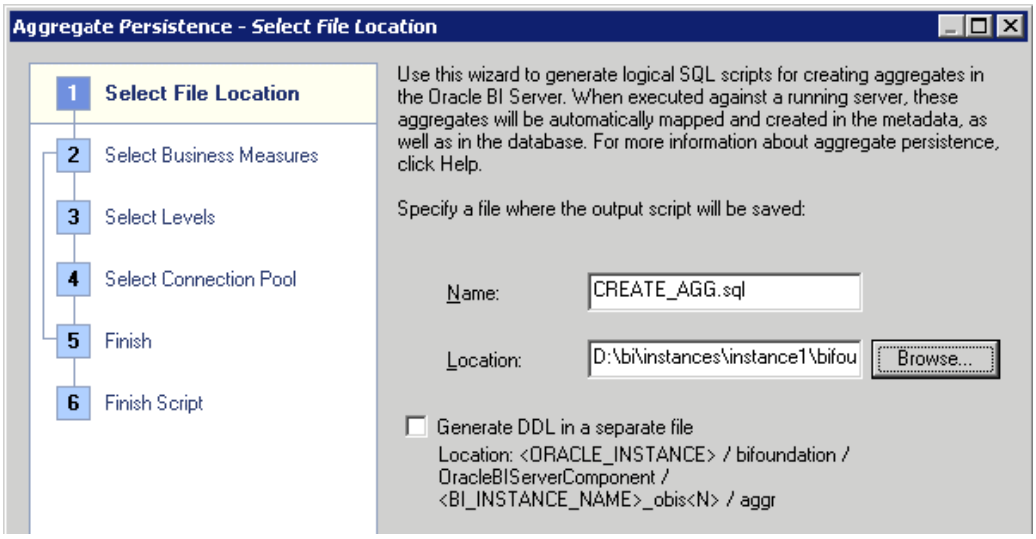


1. Open the Aggregate Persistence Wizard

Select Tools > Utilities > Aggregate Persistence and click the Execute button to open the Aggregate Persistence Wizard.

2. Specify a File Name and Location

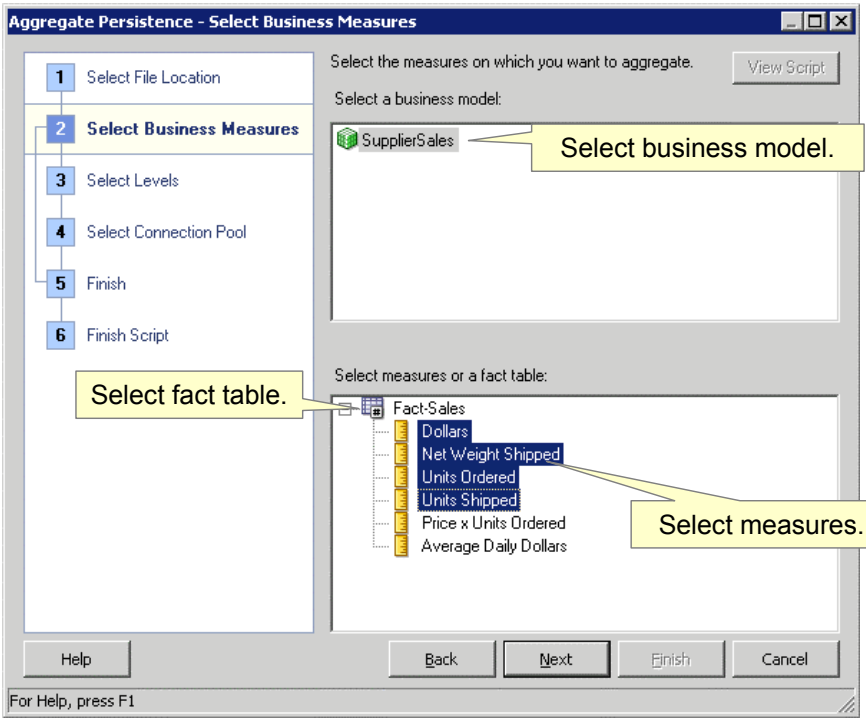
Specify a file and location where the output script should be saved.



2. Specify a File Name and Location

The wizard helps you generate logical SQL scripts for creating aggregates. This file stores the aggregate specifications and is updated if more aggregates are specified. When the script is executed, the aggregates are automatically mapped and created in the metadata as well as in the database.

3. Select the Business Model and Measures



3. Select the Business Model and Measures

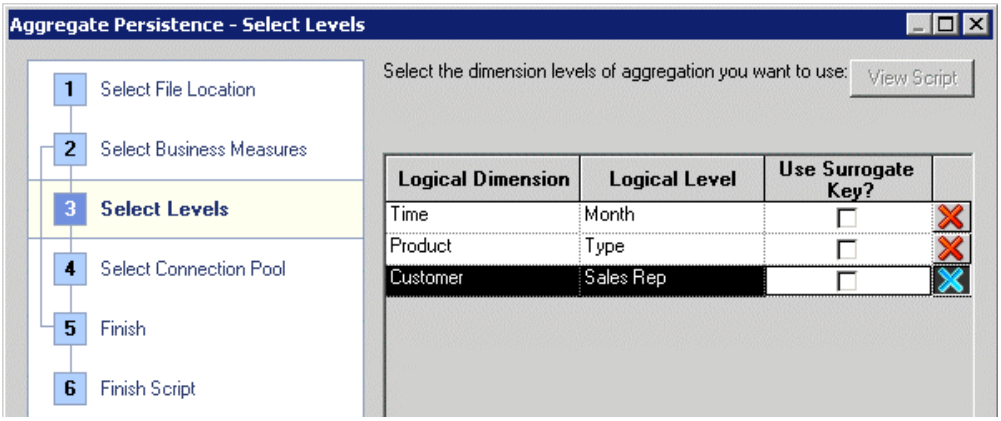
In the top pane of the Select Business Measures screen, select the business model. When there are multiple business models, only one can be selected.

In the bottom pane, select the fact table. When there are multiple fact tables, only one fact table can be selected.

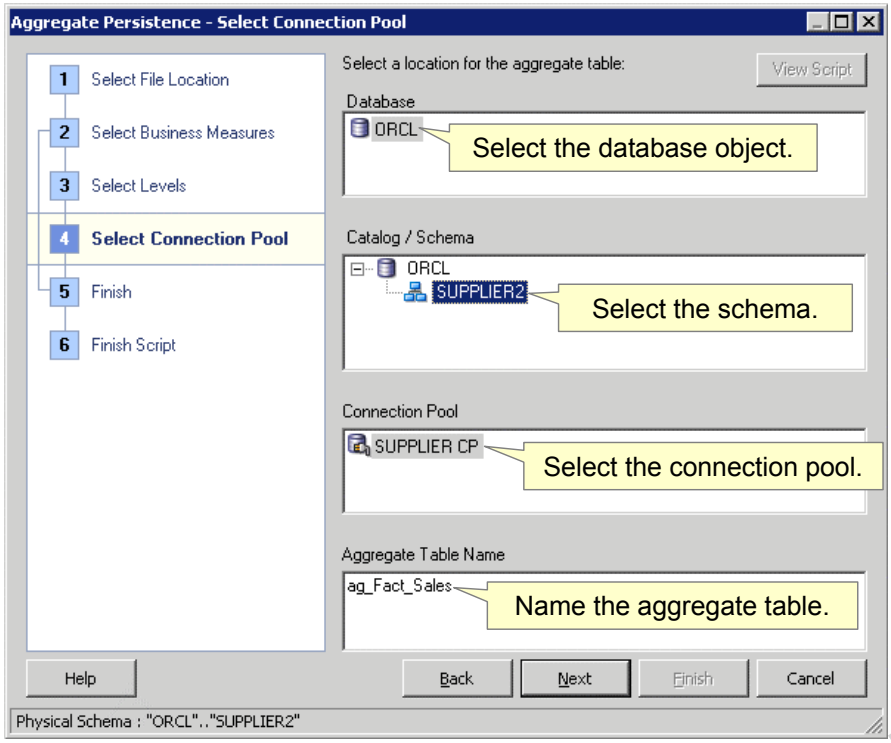
Expand the fact table and select the desired measures.

4. Select Dimensions and Levels

Select corresponding aggregate dimensions and levels.



5. Select the Connection Pool, Container, and Name



5. Select the Connection Pool, Container, and Name

In the top pane of the Select Connection Pool screen, select the repository database object.

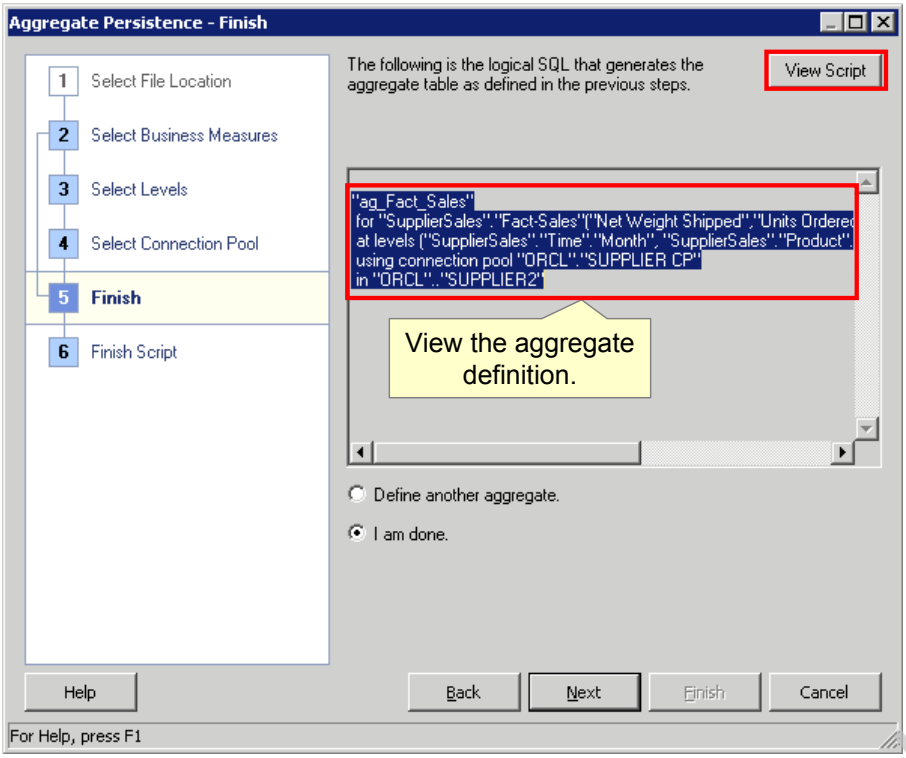
In the second pane, expand the database object and select the desired schema. In this example, there is only one schema, `SUPPLIER2`.

In the third pane, select the connection pool. In this example, there is only one, `SUPPLIER CP`.

In the Aggregate table name field, accept the default name or create a new name for the aggregate table. In this example, the default name `ag_Fact_Sales` is used.

Note: The parameters provided here are for output, which could be to a different database than where the detail tables reside.

6. Review the Aggregate Definition

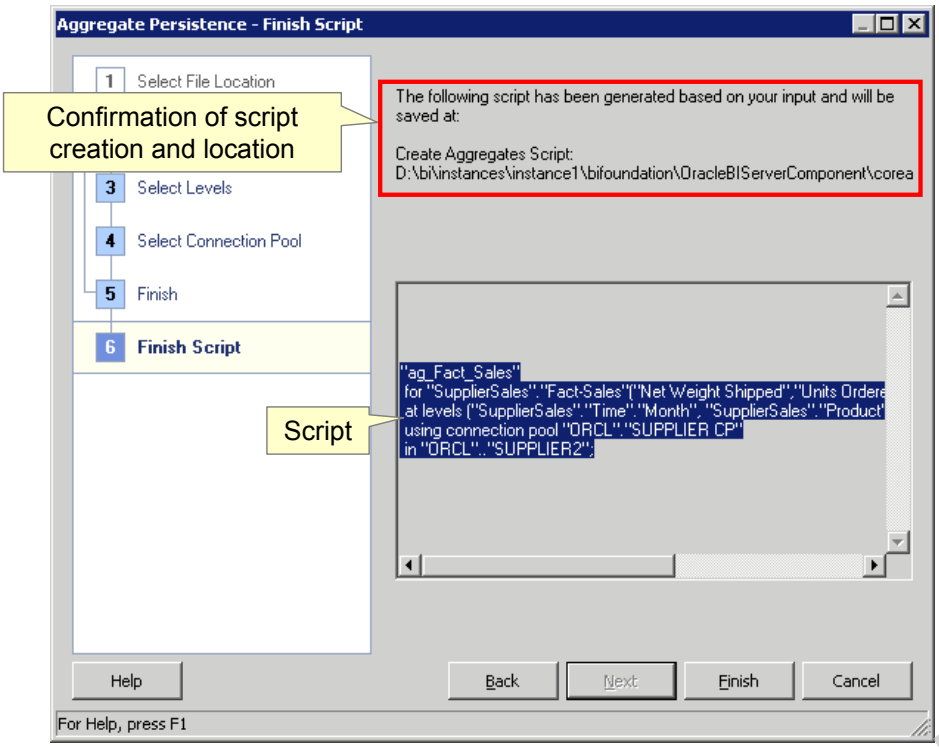


6. Review the Aggregate Definition

The screen displays the script that generates the aggregate tables based on the parameters defined in the previous steps.

Click the View Script button to view, search, or copy the script. Here you can define another aggregate or select the “I am done” option.

7. View the Complete Aggregate Script

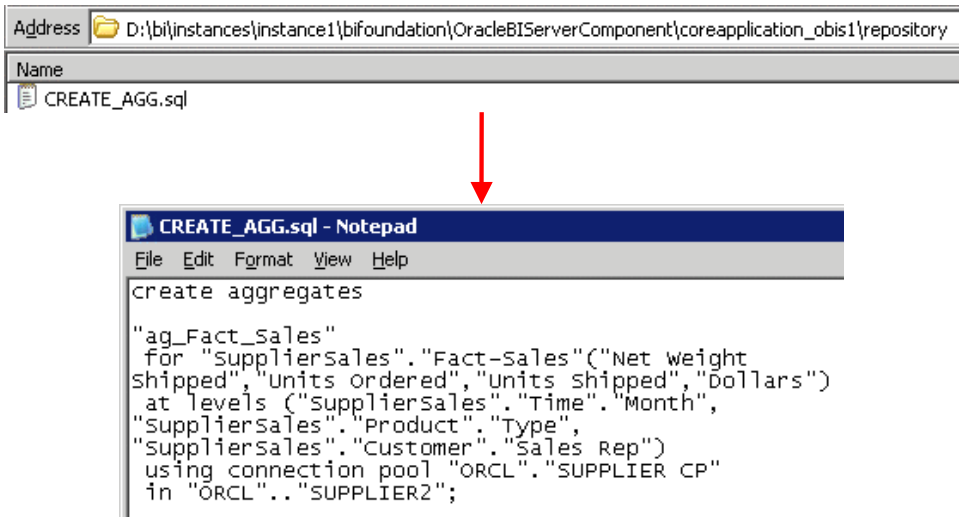


7. View the Complete Aggregate Script

When you have finished, the wizard displays the Complete Aggregate Script dialog box, confirming that the script has been generated and stored in the location identified in an earlier step.

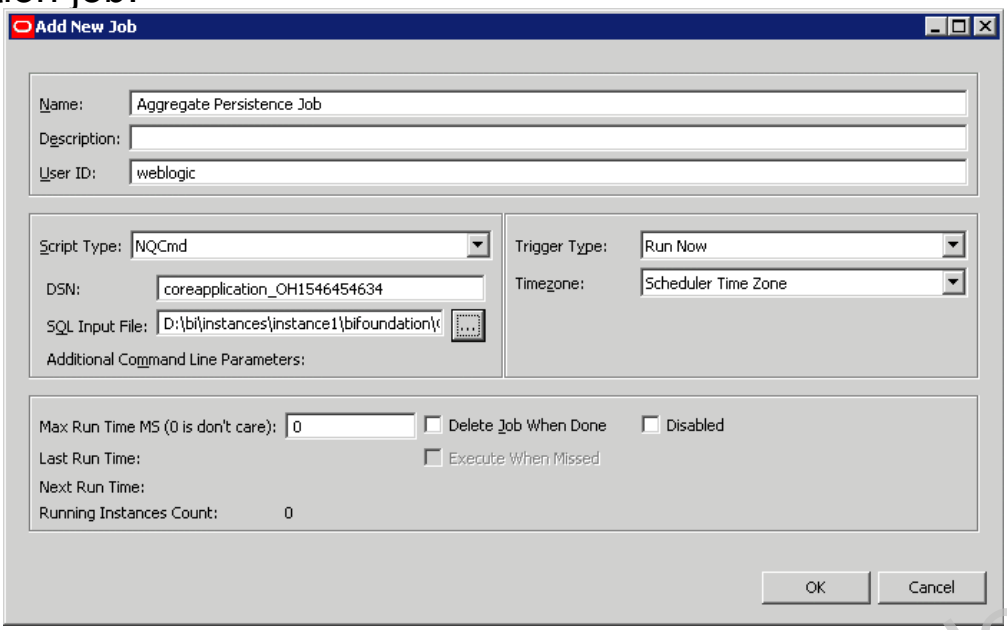
8. Verify That the Script Is Created

Navigate to the directory where the file was saved and verify that the script was created as expected.



9. Execute the Aggregate Creation Job

Use the Job Manager to create and execute the aggregate creation job.

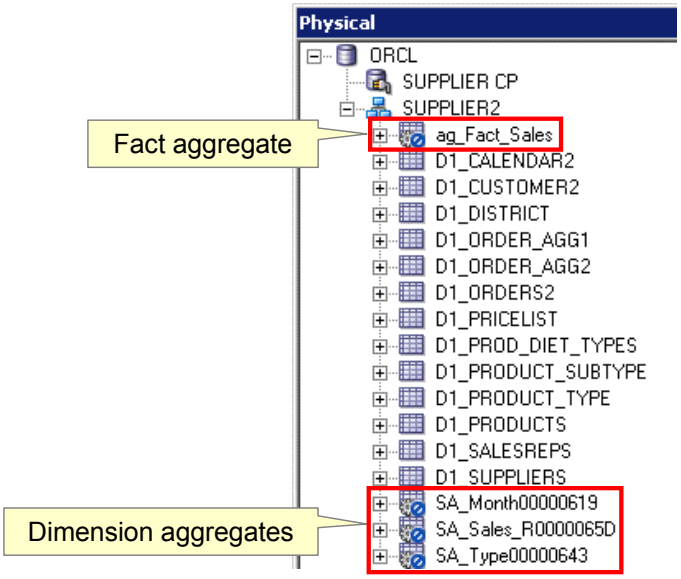


9. Execute the Aggregate Creation Job

You need to start and run the Job Manager in online mode. This is one method for running the aggregate persistence script. You can also use the Issue SQL screen in Presentation Services Administration to issue SQL directly against the Oracle BI Server.

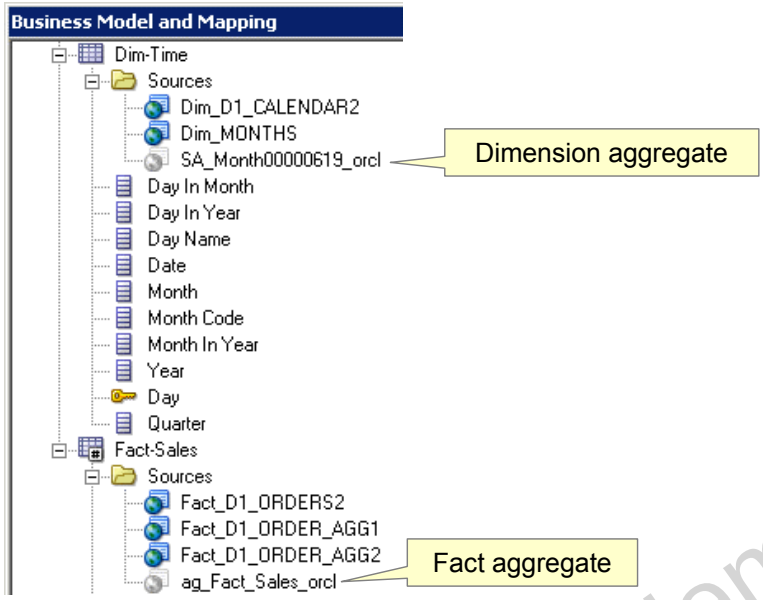
10. Verify Aggregates in the Physical Layer

Verify that the aggregates are created in the Physical layer of the repository as expected.



11. Verify Aggregates in the BMM Layer

Verify that the aggregates are created in the Business Model and Mapping layer of the repository as expected.



12. Verify Aggregates in the Database

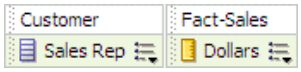
Verify that the aggregates are created in the database.

```
SQL> select table_name from user_tables where table_name like 'SA_%'
 or table_name like 'AG_%';

TABLE_NAME
------------------------------
AG_FACT_SALES
SA_TYPE00000643
SA_SALES_R00000065D
SA_MONTH00000619
```

13. Verify Your Work

- Activate the aggregate tables:

- Run an analysis:



- Check the log and verify that the aggregate tables are accessed as expected:

```
select T912.Sales_Rep0000020D as c1,
     sum(T931.Dollars00000268) as c2
from
     SA_SalesRe0000043C T912,
     ag_SalesFacts T931
where  ( T912.Sales_Rep0000020D = T931.Sales_Rep0000020D )
group by T912.Sales_Rep0000020D
order by c1
```

Troubleshooting Aggregate Navigation

If aggregate navigation is not working, the cause might be one of the following:

- Aggregation content is not specified correctly for one or more sources.
- Aggregate dimension sources are not physically joined to aggregate fact table sources at the same level.
- A dimensional source does not exist at the same level as a fact table source.
- Aggregate dimension sources do not contain a column that maps to the primary key of the dimension hierarchy level.
- The number of elements is not specified correctly for dimension hierarchy levels.

Troubleshooting Aggregate Navigation

The slide lists some of the common reasons that aggregate navigation fails to work. Use this as a troubleshooting list to help identify causes and solutions. This list is applicable regardless of the method used to define aggregation navigation.

Considerations

Using aggregates comes with a price:

- Additional time is required to build and load these tables.
- Additional storage is necessary.

Build only the aggregates you need:

- Look at query patterns and build aggregates to speed up common queries that require summarized results.
- Ensure that enough data is combined to offset the cost of building aggregates.
- Monitor and adjust to account for changing query patterns.

Summary

In this lesson, you should have learned how to:

- Describe aggregate tables and their purpose in dimensional modeling
- Model aggregate tables
- Set the number of elements for a hierarchy level
- Use the Aggregate Persistence Wizard to create aggregates

Practice 10-1 Overview: Using Aggregate Tables

This practice covers the following topics:

- Importing aggregate tables
- Creating keys and joins for aggregate tables
- Creating logical table sources for aggregate tables
- Specifying aggregate content

Practice 10-1 Overview: Using Aggregate Tables

ABC wants to add sources that contain pre-computed aggregations. You need to specify the level of aggregation for each source using logical levels. The database administrator for ABC has already created the necessary aggregate tables:

- `MONTHS` contains one row for each year and month combination, which can be considered an aggregation of the Period dimension to the Month level.
- `D1_ORDER_AGG1` contains sales facts aggregated to the Customer.Sales Rep level, Product.Type level, and Period.Month level.
- `D1_PRODUCT_TYPE` is already part of your model and contains one row for each product type, which can be considered an aggregation of the Product dimension to the Type level.

Practice 10-2 Overview: Setting the Number of Elements

This practice covers setting the number of elements for logical dimension levels and observing the results.

Practice 10-2 Overview: Setting the Number of Elements

In this practice, you set the number of elements for logical dimension levels and observe the results. The number of elements is used by Oracle BI Server when picking aggregate sources. Setting the number of elements is necessary only when there are two or more aggregate sources that can be accessed by an Oracle BI query.

Aggregate fact sources are accessed based on a combination of the fields selected as well as the number of elements of the levels in the logical dimensions to which they map. The number does not have to be exact, but ratios of numbers from one logical level to another should be accurate. By adjusting the number of elements, you can alter the aggregate fact source selected by Oracle BI Server.

Practice 10-3 Overview: Using the Aggregate Persistence Wizard

This practice covers using the Aggregate Persistence Wizard to automate the creation of aggregate tables and their corresponding objects in the repository.

Practice 10-3 Overview: Using the Aggregate Persistence Wizard

The traditional process for creating aggregates for Oracle BI Server queries is manual. It can be tedious, requiring complicated DDL and DML scripts to be written for creating tables in the databases involved. Additionally, these aggregated tables need to be mapped into the repository metadata to be available for queries. This is a time-consuming and, possibly, error-prone process.

The Aggregate Persistence Wizard enables an administrator to automate the creation of aggregate tables and their corresponding objects in the repository. Recall that your repository contains an aggregate table called `D1_ORDER_AGG1`, which contains sales fact data aggregated at the month and product type levels, and corresponding Product Type and Months dimension aggregate tables. In this practice, you use the Aggregate Persistence Wizard to create similar aggregate tables and the corresponding metadata.

11

Using Partitions and Fragments

Objectives

After completing this lesson, you should be able to:

- Identify reasons for segmenting data
- Describe techniques to model partitions
- Implement a value-based partition
- Implement a fact-based partition

Business Challenge

- Data is often partitioned into multiple physical sources for a single logical table.
- Organizations need to seamlessly and efficiently access and process data from multiple sources to satisfy user requests.
- Business applications must “know” where to go for specific types of data under specific conditions.

Business Challenge

Data is often partitioned into multiple physical sources for a single logical table in a business model. 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 contains.

For example, there are situations in which data may be fragmented or partitioned. When individual sources at a given level contain information for a portion or fragment of the domain, an application needs to know the content of the sources to pick the appropriate source for the query. This should have no impact on the end-user experience. How and where the data is accessed should be completely transparent to the end user.

Business Solution: Oracle BI Server

- The Oracle BI repository can be configured so that Oracle BI Server handles navigation to the appropriate source.
- Oracle BI Server seamlessly and efficiently accesses and processes data from multiple sources to satisfy user requests.

Business Solution: Oracle BI Server

When there are multiple sources, the metadata can be built so that Oracle BI Server handles navigation to the appropriate source. Oracle BI Server can seamlessly access and process data from multiple sources efficiently to satisfy user requests.

For example, as you saw in the lesson titled “Using Aggregates,” a database administrator can create an aggregate table to improve performance. The metadata must be configured correctly so that Oracle BI Server knows when it is appropriate to access the aggregate table instead of the detailed table when satisfying a user request.

Partition

- Is a database element that contains part of the data for a fact or a dimension
- Combines with other data fragments as necessary
- May be:
  - Fact based
  - Value based
  - Level based
  - Complex

Partition

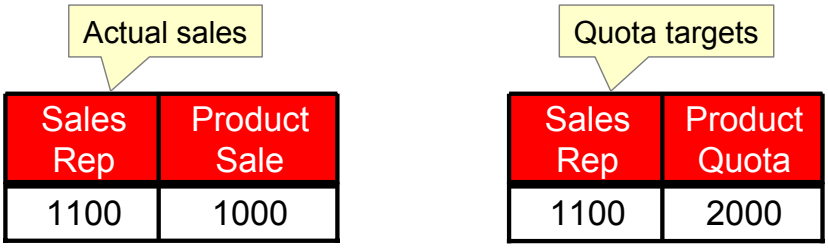
A partition typically contains a subset of the data for a fact or dimension. A partition is the division of a database or its elements into distinct independent parts. Database partitioning is done for reasons of manageability, performance, or availability.

Partitioning can be done by building separate smaller databases or by splitting larger selected elements into smaller ones (for example, splitting one large table into many smaller tables). When a user requests data, it may be necessary to consolidate data from different partitions to complete the request.

The different partition types—fact based, value based, level based, and complex—are discussed in detail in the following slides.

Partitioning by Fact

- Data is partitioned by fact when different fact data is stored in different tables.
- Example: Actual sales versus quota targets



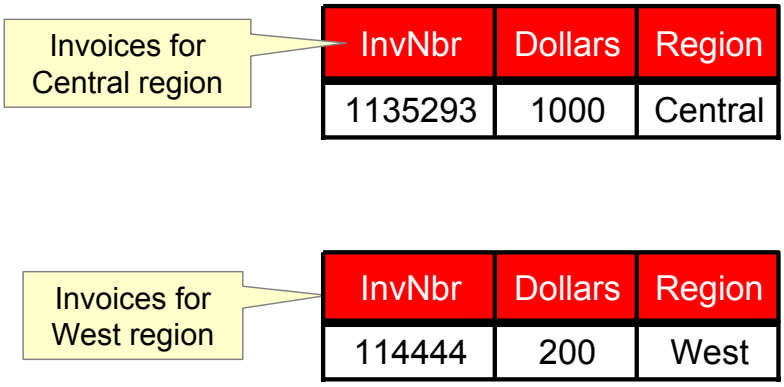
Partitioning by Fact

Data can be stored in such a way that different facts are in different tables. For example, companies usually store sales quotas in tables separate from the actual sales history. Inventory data is also commonly found stored in a database separate from sales history.

This is sometimes referred to as *vertical partitioning* because a theoretical all-encompassing fact table can be thought to be sliced "vertically," with different columns going to different fact tables.

Partitioning by Value

- Data is partitioned by value when the data is split into separate tables according to the values of the data.
- Example: Invoice data is stored separately for each region.



Partitioning by Value

Data can also be partitioned into separate tables according to the values of the data.

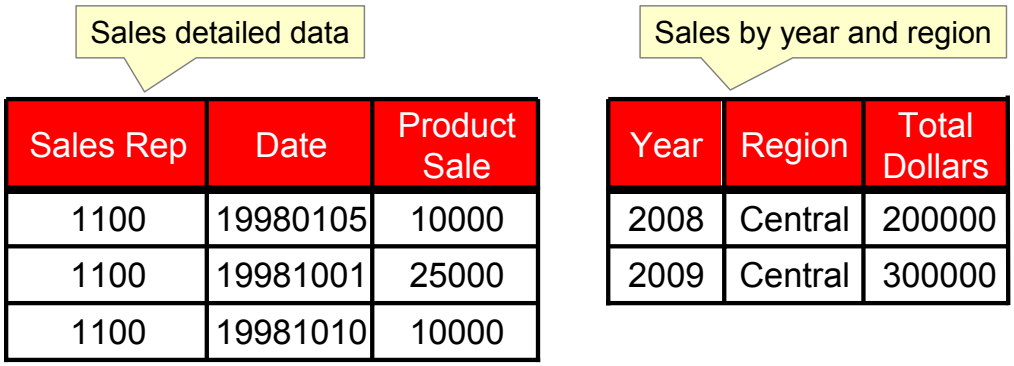
Examples:

- East region sales data may be in a different table from West region sales data.
- Inventory data may be in separate tables segregated by warehouse.
- The data for the current year may not be in the same table as the data for prior years, or data could be separated by month, so that the last three years of data are in 36 separate tables.

This type of partitioning creates complexity in the query environment. Not only is the table population increased, thereby creating another query navigation issue, but any user who seeks to answer a question such as “What is the total number of widgets in inventory?” has to query multiple tables and then consolidate the results. One of the important benefits of Oracle BI Server is that it can do this type of navigation and consolidation automatically, preserving a simple logical model of the data with which users interact.

Partitioning by Level

- Data is partitioned by level when the same facts are stored in separate tables at different levels of aggregation.
- Example: Detailed sales data is summarized and stored by year and region.



Sales Rep	Date	Product Sale
1100	19980105	10000
1100	19981001	25000
1100	19981010	10000

Year	Region	Total Dollars
2008	Central	200000
2009	Central	300000

Partitioning by Level

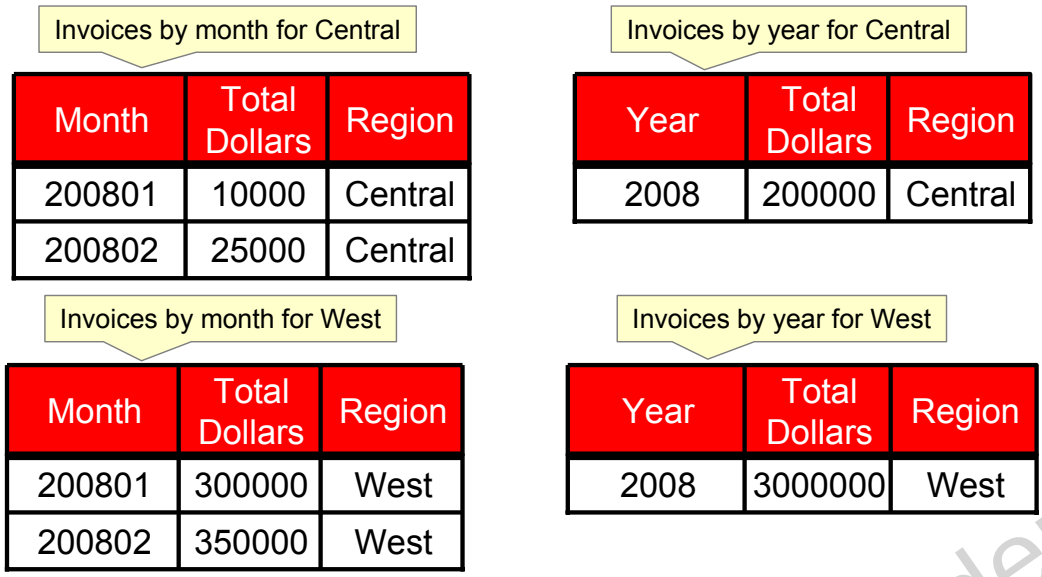
Data can also be partitioned by level. This occurs when the same facts are stored in separate tables at different levels of aggregation. You saw examples of this in the previous lesson. The use of aggregate tables is a common practice in data warehouse design because it is a very effective tool for improving query performance.

Theoretically, however, the number of aggregate tables in a data warehouse can get very large. As such, data warehouse designers prefer to follow an incremental approach to creating aggregations. They look for the best performance, creating the aggregate tables that are going to be used most frequently and that offer the most data compression (that is, where the row counts decrease the most).

Of course, aggregate fact tables invite aggregate dimension tables. The result is that while introducing aggregate tables goes a long way toward improving query performance, it also complicates the overall query environment. One of the important benefits of Oracle BI Server is that it is "aggregate aware." It writes SQL to use an efficient aggregate table automatically, while preserving a simple logical model of the data for the user to interact with. The user, therefore, remains insulated from the actual aggregate navigation process and sees only the query results.

Complex Partitioning

- Data is partitioned using more than one technique.
- Example: Invoice sales data is partitioned by value and level.



Complex Partitioning

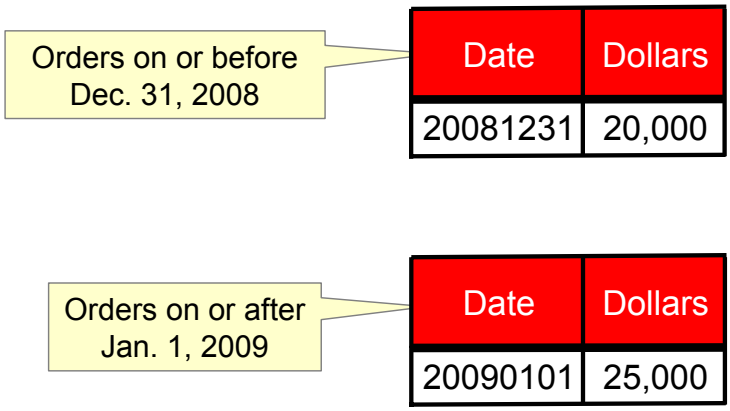
Partitioning strategies can be mixed. For example, data can be partitioned by level and value so that at any given intersection of levels, multiple aggregate tables may exist. Moreover, at any given intersection of levels, not all values might be stored.

For example, at a Brand, Month, and Market aggregate level, just the important brands might be stored. Certain queries at this level can use the aggregate tables; other queries, with different constraints, have to use more detailed tables.

Oracle BI Server also navigates complex partitions for the user.

ABC Example: Value Based (Order Date)

Replace the current, single source for order data with two value-based partitions.



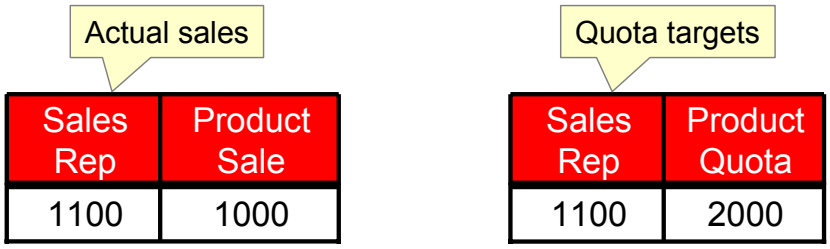
ORACLE

ABC Example: Value Based (Order Date)

ABC wants to implement separate partitions for order data. One partition stores historical data for orders on or before December 31, 2008. The other partition stores recent data for orders on or after January 1, 2009.

ABC Example: Fact Based (Quota)

Enable users to query for actual sales data and quota data in a single query.



ABC Example: Fact Based (Quota)

ABC sets sales quotas for its sales organization. These quotas are set at the regional level by quarter. In addition, each of the regional sales quotas is broken down by product type.

Quota numbers are stored in an Excel workbook. You incorporate the quota numbers in the business model and create business measures to report variance from quota and percentage of quota.

Implementation Steps

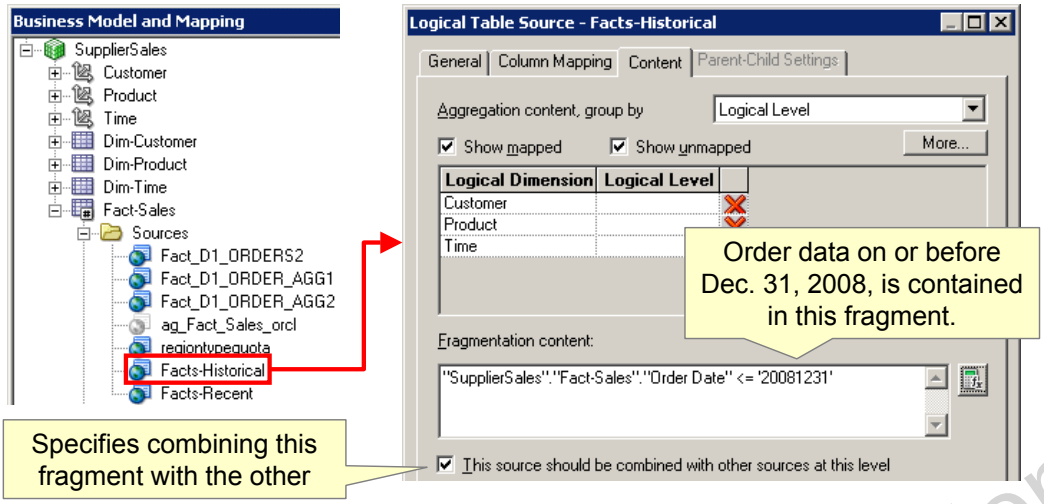
1. Import physical sources.
2. Create physical joins.
3. Add sources to the Business Model and Mapping layer.
4. Specify fragmentation content. New step
5. Test the results.

Implementation Steps

Most of the steps listed here should be familiar by now. The only new step is specifying fragmentation content, which is discussed in detail in the next slide.

Specify Fragmentation Content

- Use the Expression Builder to define the type of content that the fragment contains.
- Set the flag to specify whether to combine this fragment with other data at this level.



Specify Fragmentation Content

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 contains. Describe the content in terms of logical columns, using the “Fragmentation content” edit box on the Content tab of the Logical Table Source window. Click the button next to the “Fragmentation content” box to open the Expression Builder to build the expression.

This is similar to the steps used to define aggregates. For aggregates, you specified the content by using the level indicator. Here you use an expression to serve the same purpose—to identify what data the source contains. In this example, the `Facts-Historical` table contains data for orders on or before December 31, 2008. In a request for order data, this source may need to be combined with data from other sources at this level. In this example, that would be the `Facts-Recent` table, which contains the remaining order data.

Summary

In this lesson, you should have learned how to:

- Identify reasons for segmenting data
- Describe techniques to model partitions
- Implement a value-based partition
- Implement a fact-based partition

Practice 11-1 Overview: Modeling a Value-Based Partition

This practice covers the following topics:

- Creating a partition
- Defining fragmentation content for a value-based partition

Practice 11-1 Overview: Modeling a Value-Based Partition

ABC wants to store its fact data in two separate partitions, one for recent data and one for historical data. Each partition contains the same columns. Only the data values are different. The historical data partition stores invoice fact data up to and including December 31, 2008. The recent data partition stores invoice fact data after December 31, 2008.

Practice 11-2 Overview: Modeling a Fact-Based Partition

This practice covers modeling a fact-based partition for quota data.

Practice 11-2 Overview: Modeling a Fact-Based Partition

ABC sets sales quotas for its sales organization. These quotas are set at the regional level by quarter. In addition, each of the regional sales quotas is broken down by product type.

Quota numbers are stored in an Excel workbook. The `quota.xls` workbook is stored on your machine. You incorporate the quota numbers in the business model and create business measures to report variance from quota and percent of quota.

Practice 11-3 Overview: Using the Calculation Wizard to Create Derived Measures

This practice covers creating measures for the quota partition.

Practice 11-3 Overview: Using the Calculation Wizard to Create Derived Measures

Use the Calculation Wizard to create two derived measures: "Variance from Quota" and "Percent of Quota."

12

Using Repository Variables

Objectives

After completing this lesson, you should be able to:

- Create session variables
- Create repository variables
- Create initialization blocks
- Implement a dynamic repository variable

Variables

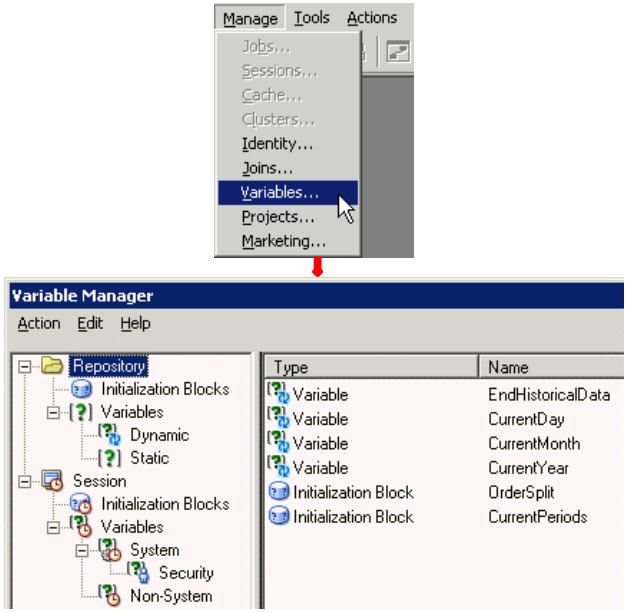
- Contain values in memory that are used by Oracle BI Server during its processing
- Are created and managed using the Variable Manager feature in the Administration Tool
- Consist of two classes:
  - Session variables
  - Repository variables

Variables

You can use variables in a repository to streamline administrative tasks and modify metadata content dynamically to adjust to a changing data environment. A repository variable has a single value at any point in time. Repository variables can be used instead of literals or constants in the Expression Builder in the Administration Tool. Oracle BI Server substitutes the value of the repository variable for the variable itself in the metadata.

Variable Manager

Is a utility in the Administration Tool that is used to define variables



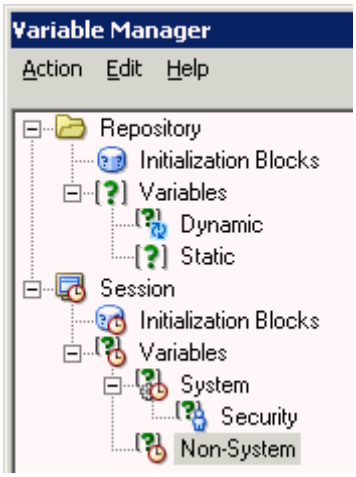
Variable Manager

You can use the Variable Manager to define variables. The Variable Manager dialog box 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. The left pane is divided into repository variables and initialization blocks, and session variables and initialization blocks. Select Manage > Variables to access the Variable Manager.

Variable Types

Repository:
- Static
- Dynamic

Session:
- System
- Non-system



Variable Types

There are two classes of variables: repository variables and session variables.

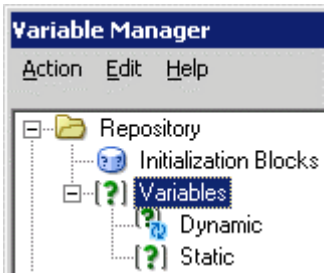
- A repository variable has a single value at any point in time. There are two types of repository variables: static and dynamic. Repository variables are represented by a question-mark icon (**?**).
- Session variables are created and assigned a value when each user logs on. There are two types of session variables: system and non-system. System and non-system variables are represented by a question-mark icon (**?**).

Initialization blocks are used to initialize dynamic repository variables, system session variables, and non-system session variables. The icon for an initialization block is a cube labeled with the letter *i*.

Each of these components is discussed in detail in the following slides.

Repository Variables

- Persist from the time Oracle BI Server is started until it is shut down
- Can be used instead of literals or constants in the Expression Builder in the Administration Tool
- Exist in two forms:
  - Static
  - Dynamic

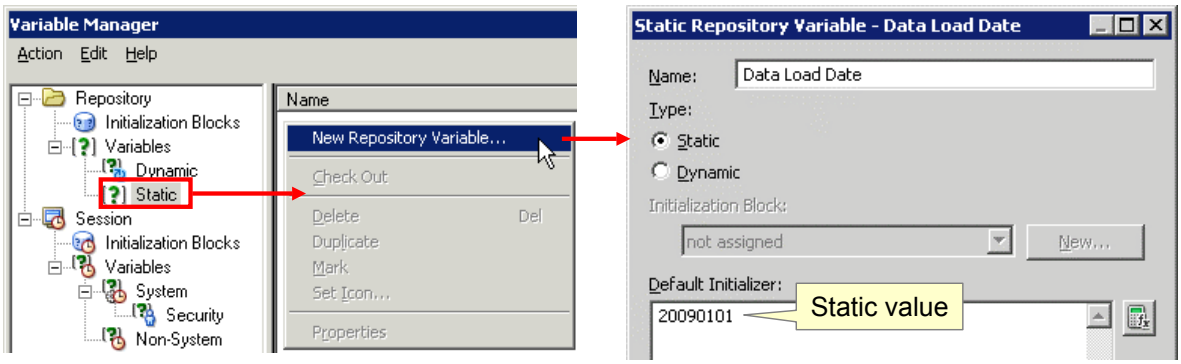


Repository Variables

A repository variable has a single value at any point in time. Repository variables can be used instead of literals or constants in the Expression Builder in the Administration Tool. Oracle BI Server substitutes the value of the repository variable for the variable itself in the metadata. There are two types of repository variables: static and dynamic. These are discussed in detail in the slides that follow.

Static Repository Variables

- Are repository variables whose values are constant and do not change while Oracle BI Server is running
- Have values that are initialized in the Static Repository Variable dialog box



Static Repository Variables

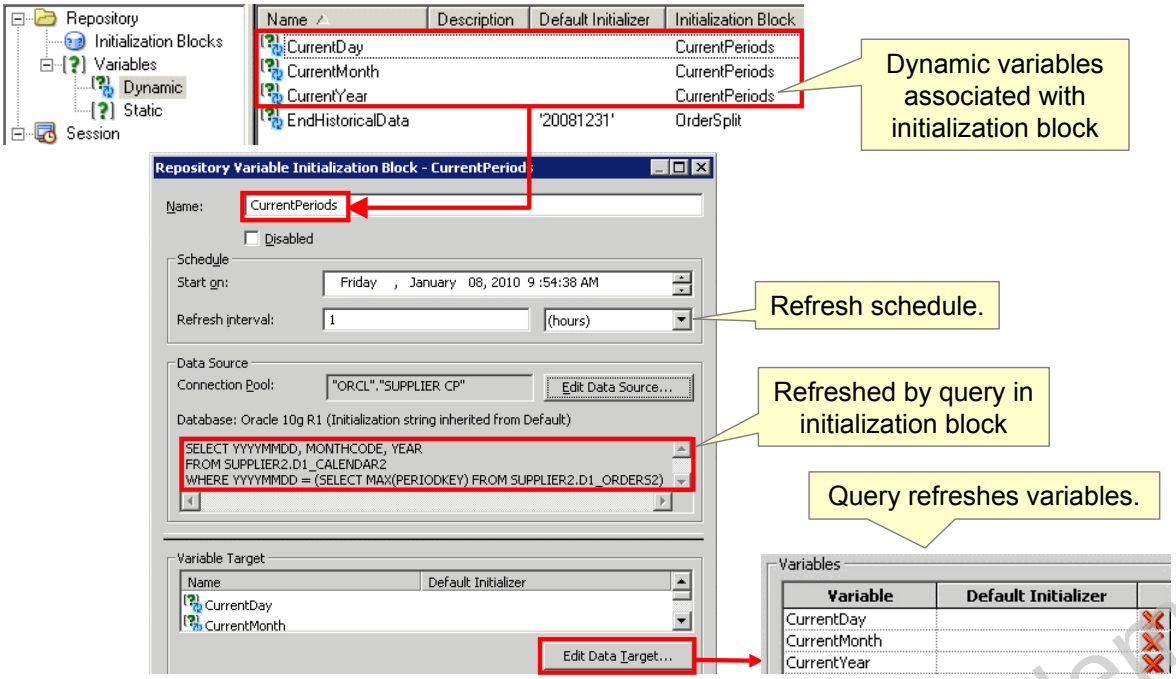
The value of a static repository value is initialized in the Default Initializer field in the Static Repository Variable dialog box. This value does not change until an Oracle BI Server administrator decides to change it. Values are initialized once and remain constant. An example of a static value is the last time data was loaded (to represent how current the analytical information is).

To create a new static repository variable:

1. Select Repository > Variables > Static in the left pane.
2. Right-click in the right pane and select New Repository Variable.
3. Enter variable values in the dialog box.

Dynamic Repository Variables

Have values that are refreshed by data returned from queries in initialization blocks



Dynamic Repository Variables

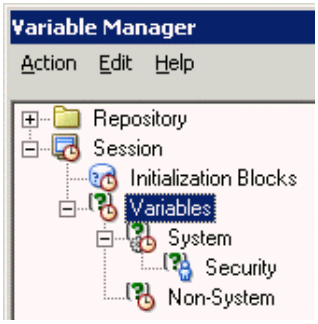
You initialize dynamic repository variables in the same way as static variables, but the values are refreshed by data returned from queries. When defining a dynamic repository variable, you create an initialization block or use a preexisting one that contains a SQL query. You also set up a schedule that Oracle BI Server follows to execute the query and refresh the value of the variable periodically.

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. In this example, the query returns three columns (`YYYYMMDD`, `MONTHCODE`, `YEAR`) that refresh three variables: `CurrentDay`, `CurrentMonth`, and `CurrentYear`.

A common use of these variables is to set column filters in analyses. For example, to filter a column on the value of the dynamic repository variable `CurrentMonth`, set the column filter to the variable `CurrentMonth`.

Session Variables

- Persist only while a user's session is active
- Receive values when users establish their sessions
- Exist in two forms:
  - System
  - Non-system



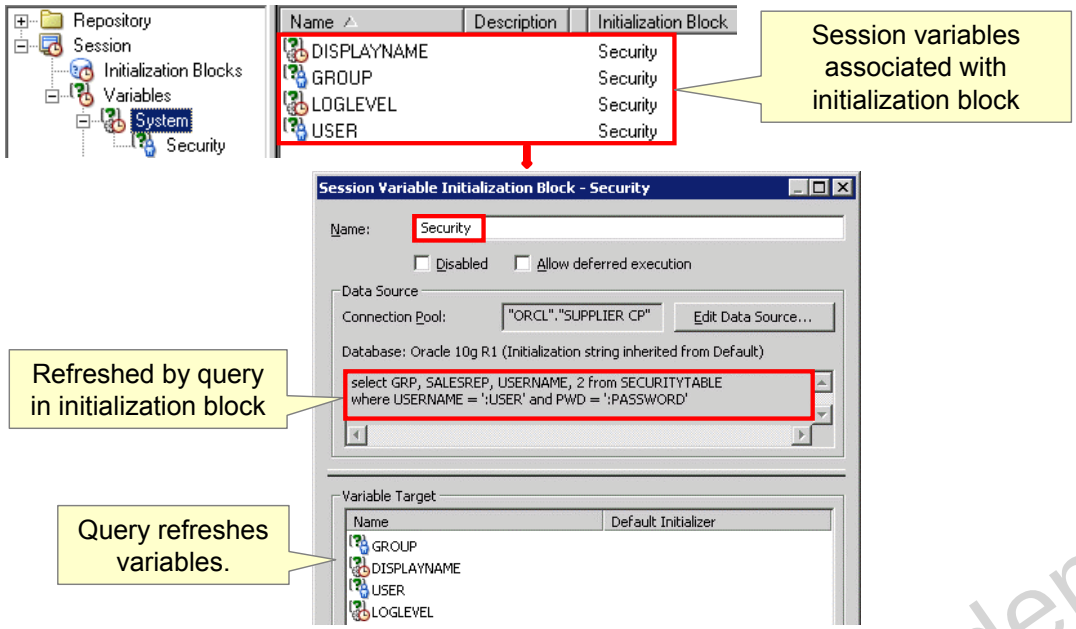
Session Variables

Session variables are like dynamic repository variables in that they 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, 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 Oracle BI Server. Each instance of a session variable can be initialized to a different value. A session is an instance of a user running the client application. The session starts when a user opens the application and ends when the user closes it. The types of session variables are discussed in detail in the following slides.

System Session Variables

Are predefined session variables used by Oracle BI Server for specific purposes, such as authenticating users



System Session Variables

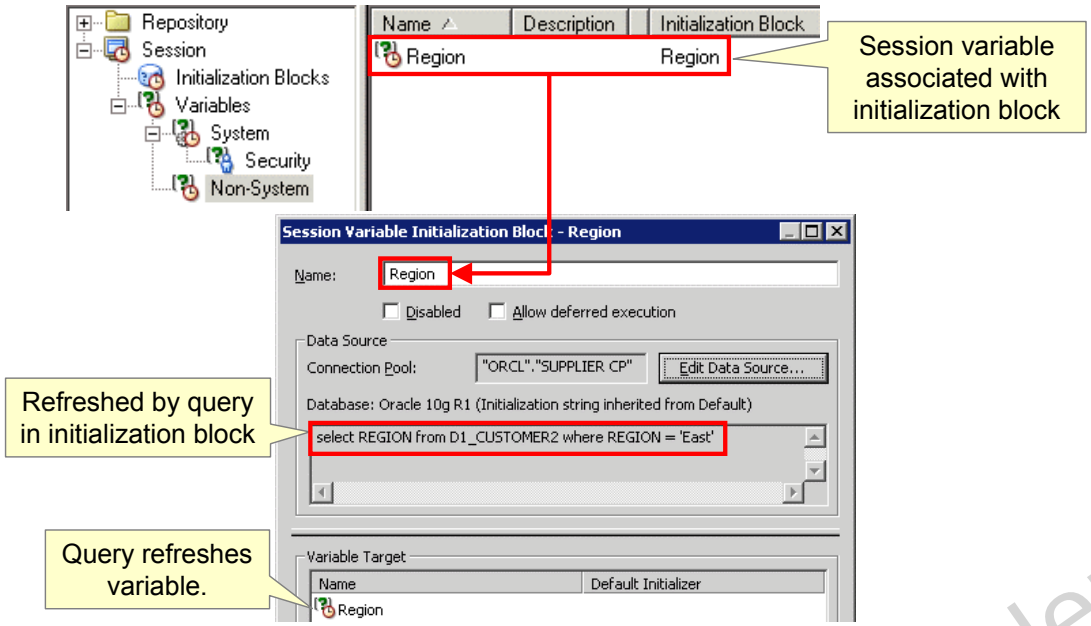
System session variables are session variables that Oracle BI Server and Oracle BI Presentation Services use for specific purposes. System session variables have reserved names, which cannot be used for other kinds of variables. An example is `USER`, which holds the value that the user entered as a logon name. Another example is `GROUP`, which contains the group that a user belongs to.

System session variables are primarily used when authenticating users against external sources such as database tables or Lightweight Directory Access Protocol (LDAP) servers. If a user is authenticated successfully, session variables can be used to set filters and permissions for that session.

In this example, the initialization block query requests values for four columns in a security table based on user name and password. The values are used to refresh the `GROUP`, `DISPLAYNAME`, `USER`, and `LOGLEVEL` variables.

Non-System Session Variables

Are application-specific variables that are created by the implementation team



Non-System Session Variables

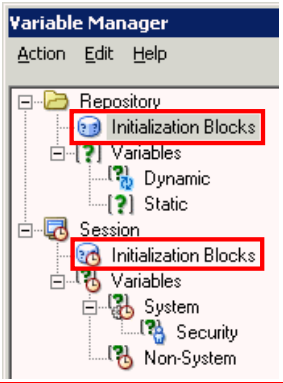
Unlike system session variables, which have reserved names and are used for specific purposes, non-system session variables can be created by the administrator to serve a purpose for a specific application. A common use for non-system session variables is setting user filters.

For example, you can define a non-system variable called `Region` that is initialized to the name of the user's sales region. You can then set a security filter for all members of a group that enables them to see only data pertinent to their region.

In this example, an initialization block named `Region` populates the variable `Region` with the value `East`.

Initialization Blocks

- Are used to initialize system and non-system session variables, as well as dynamic repository variables
- Specify SQL to be run to populate one or more variables by accessing data sources
- Are invoked at Oracle BI Server startup and are periodically rerun to refresh values for dynamic variables according to an established schedule



ORACLE

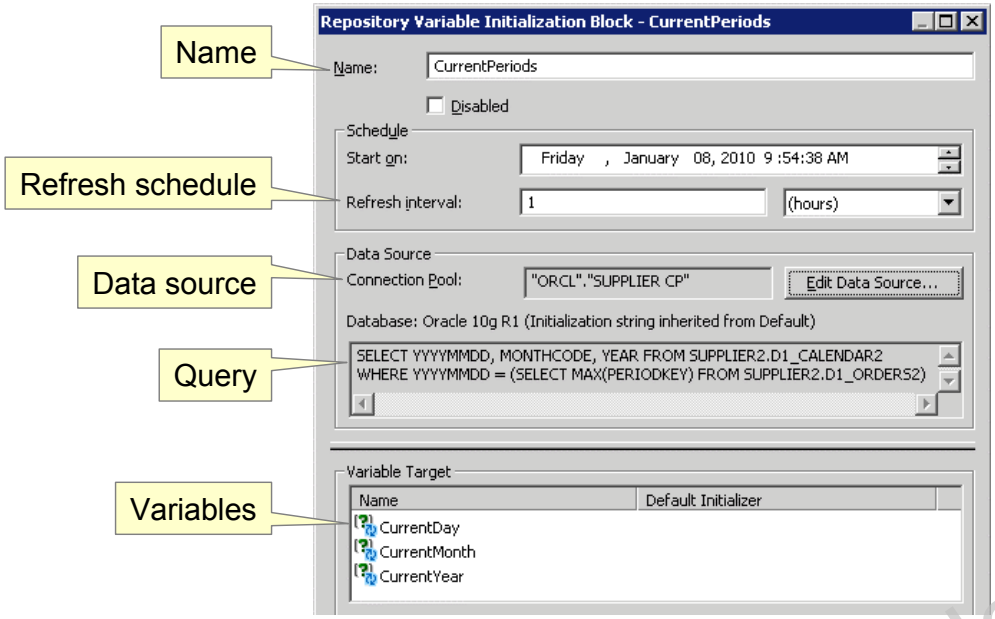
Initialization Blocks

Initialization blocks are used to initialize dynamic repository variables, system session variables, and non-system session variables. An initialization block contains the SQL that is executed to initialize or refresh the variables associated with that block. Initialization blocks are invoked at Oracle BI Server startup and are periodically rerun to refresh values for dynamic variables according to an established schedule.

Initialization blocks are related to repository or session variables.

Initialization Block: Example

Determine the latest dates contained in the source data and store it in variables:

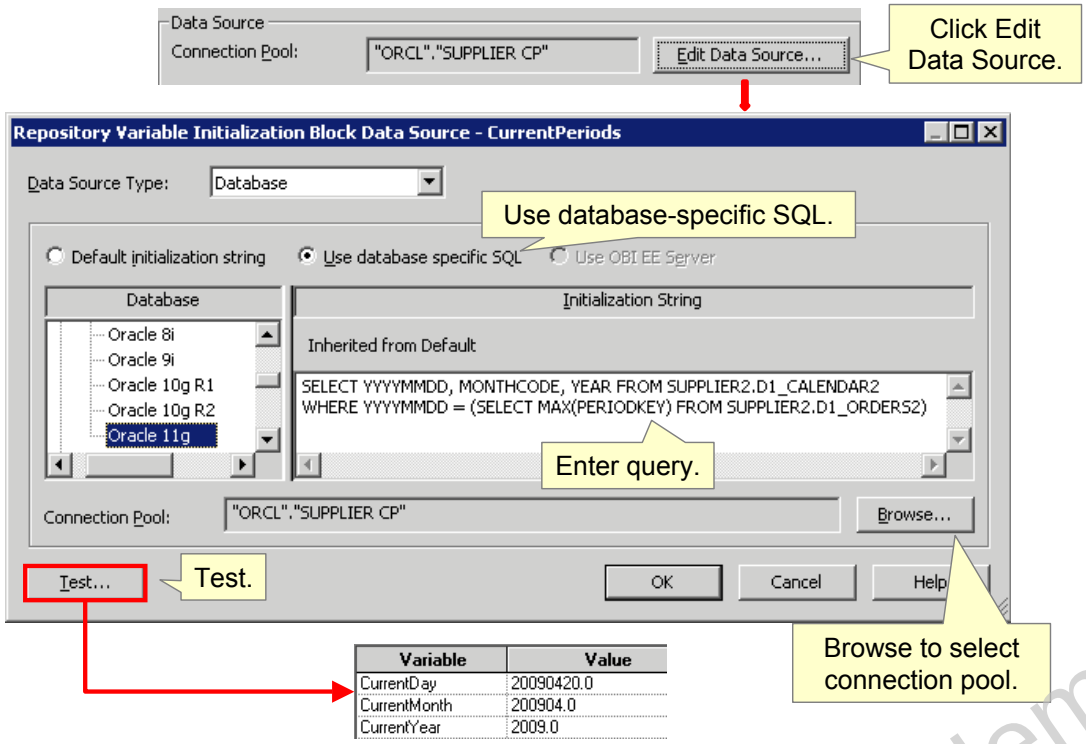


Initialization Block: Example

This slide shows an example presented earlier in this lesson in more detail.

- The name of this initialization block is `CurrentPeriods`.
- The initialization block is scheduled to refresh every hour.
- The data source is the data source identified in the `SUPPLIER` CP connection pool.
- The SQL queries the `D1_CALENDAR2` table for the latest Day, Month, and Year data based on the most recent period key in the `D1_ORDERS2` table, and then populates the `CurrentDay`, `CurrentMonth`, and `CurrentYear` variables.

Initialization Block Example: Edit Data Source



Initialization Block Example: Edit Data Source

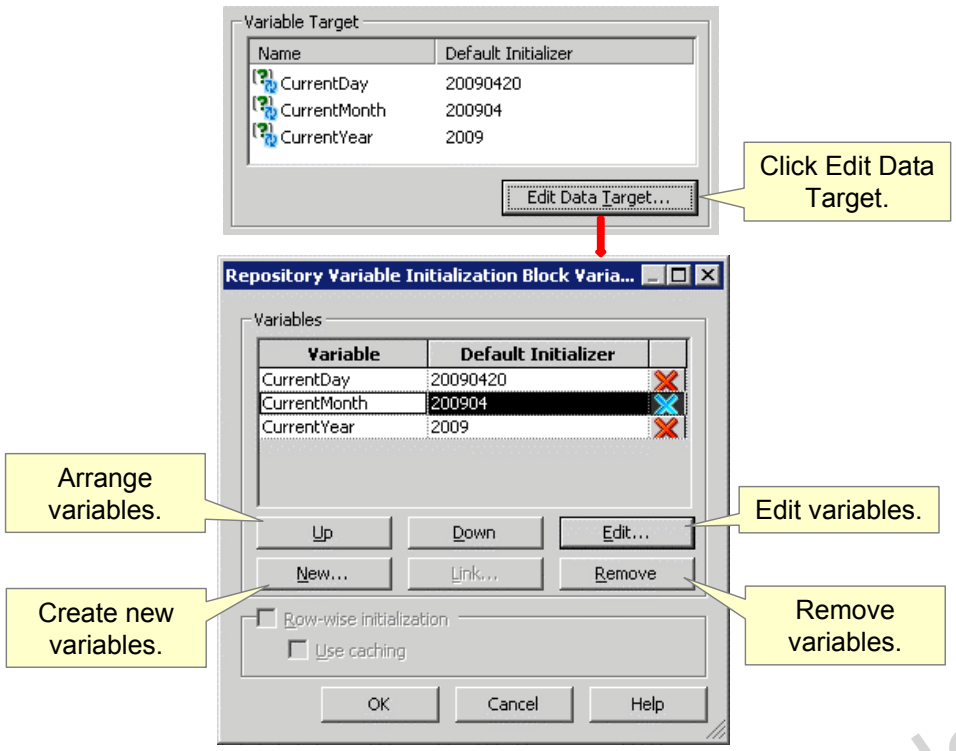
This slide shows the Repository Variable Init Block Data Source dialog box. Click the Edit Data Source button in the Initialization Block dialog box to display this dialog box, where you can do the following:

- Enter a default query or enter the query using database-specific SQL.
- Click the Test button to test the query.
- Click the Browse button to select the connection pool.

The SQL must reference physical tables that can be accessed using the connection pool specified in the Connection Pool field. 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.

When you create SQL and submit it directly to the database, bypassing Oracle BI Server (for example, when creating initialization blocks), you should test the SQL using the Test button. If the SQL contains an error, the database returns an error message.

Initialization Block Example: Edit Data Target



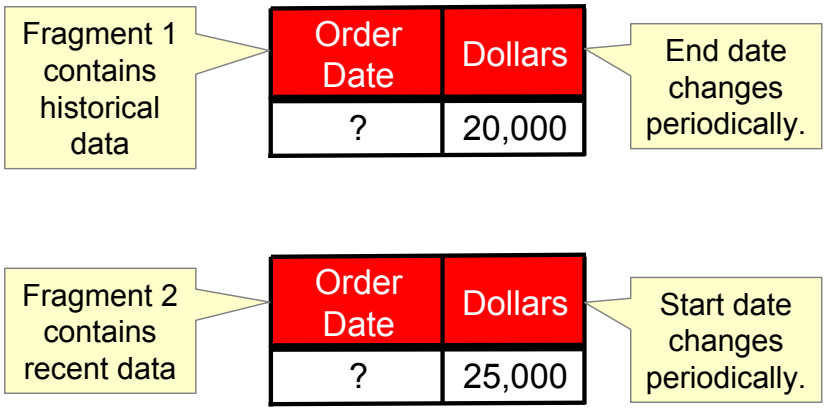
Initialization Block Example: Edit Data Target

This slide shows the Initialization Block Variable Target dialog box. Click the Edit Data Target button in the Initialization Block dialog box to display this dialog box, where you can do the following:

- Create new variables.
- Use Up and Down buttons to rearrange variable order. The order of the variables must match the order of the corresponding columns in the initialization block SQL query.
- Remove variables.
- Edit variables.

ABC Example

Use a variable to dynamically determine which set of order data is in which partition and use it to specify the fragmentation content instead of providing a hard-coded formula.



ABC Example

ABC has 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 box.

Without using dynamic repository variables, you should describe the content of the source containing recent data with a static expression such as `SupplierSales."Fact-Sales"."Order Date" > '20081231'`, as you did in the practice for the previous lesson ("Using Partitions and Fragments"). This content statement becomes invalid as new data is added to the recent source and older data is moved to the historical source. This assumes that the data is refreshed periodically and, therefore, the contents in each fragment may be different. For example, at the end of Q1 2009, the Q1 2009 order data is moved from recent fragment to historical fragment.

To accurately reflect the new content, you must modify the fragmentation content description manually. Dynamic repository values can be set up to do this automatically. To use a dynamic repository variable, you must also set up an initialization block to refresh the variable on a continuing basis.

Implementation Steps

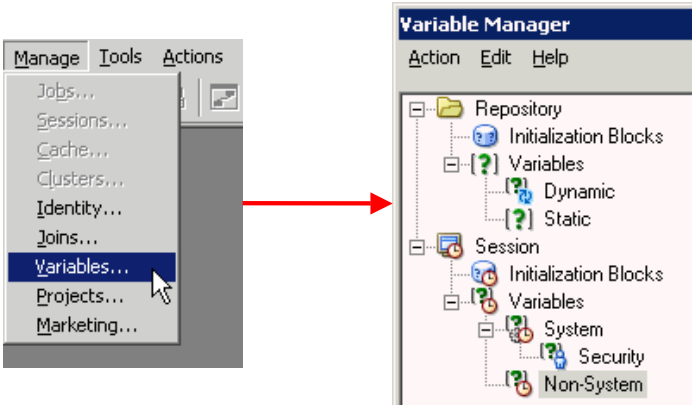
1. Open the Variable Manager.
2. Create an initialization block.
3. Edit the data source.
4. Edit the data target.
5. Test the initialization block query.
6. Use the variable to determine content.
7. Verify the initialization.
8. Verify your work.

Implementation Steps

This slide lists the high-level steps for implementing the ABC example. Each step is discussed in detail in subsequent slides.

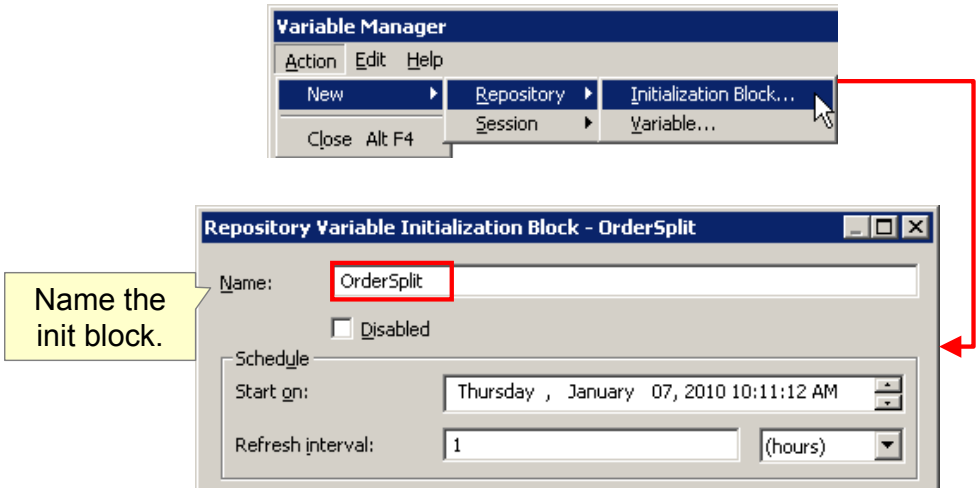
1. Open the Variable Manager

In the Administration Tool, select Manage > Variables.



2. Create an Initialization Block

In the Variable Manager, select Action > New > Repository > Initialization Block to open the Repository Variable Init Block dialog box.

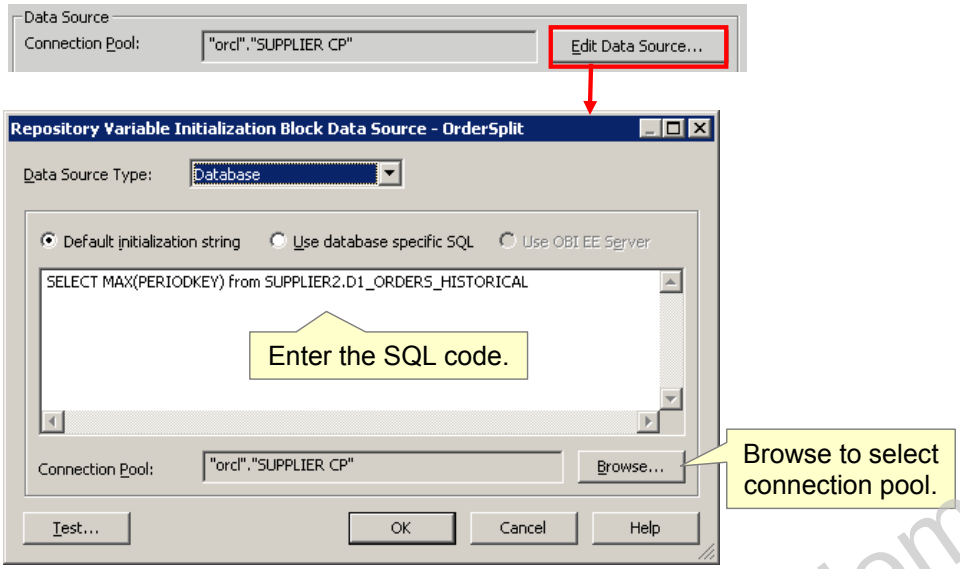


2. Create an Initialization Block

Enter `OrderSplit` as the name for the repository variable initialization block.

3. Edit the Data Source

Click the Edit Data Source button to navigate to the Repository Variable Init Block Data Source dialog box.



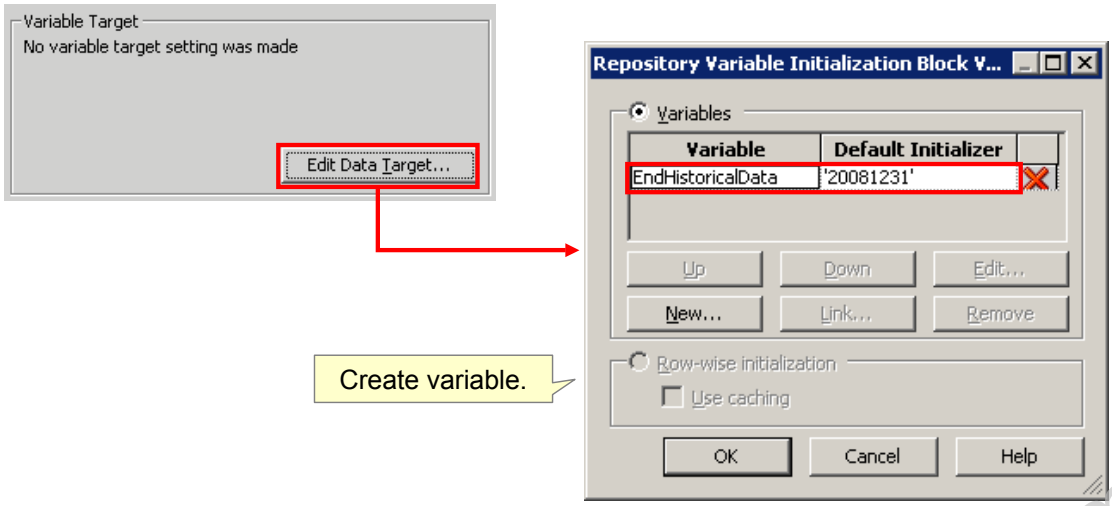
3. Edit the Data Source

Enter the SQL code in the Default Initialization String field to capture the most recent period key from the `D1_ORDERS_HISTORICAL` table.

Browse to select the `SUPPLIER CP` connection pool for the `SUPPLIER2` schema.

4. Edit the Data Target

Click the Edit Data Target button to navigate to the Repository Variable Init Block Variable Target dialog box.

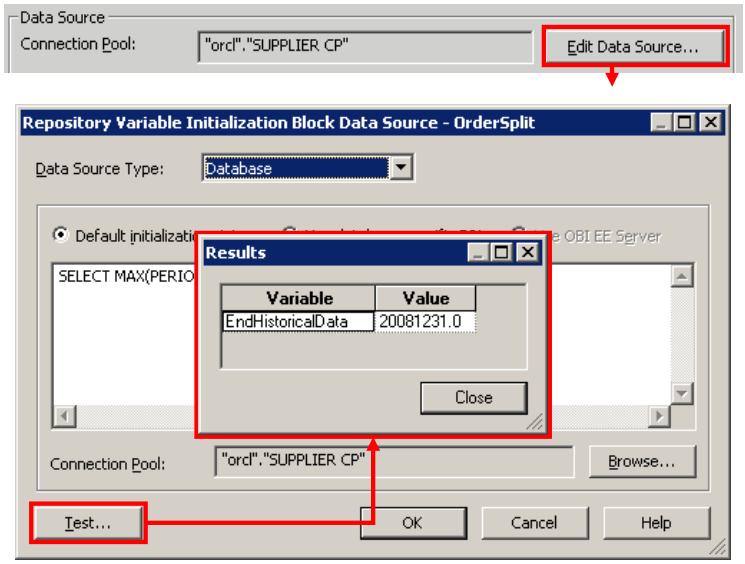


4. Edit the Data Target

Create a variable called `EndHistoricalData`, which will be refreshed by the initialization block SQL. Specify `'20081231'` as the default initializer. For repository variables, when you open a repository in online mode, the value shown in the Default Initializer column of the Initialization Block Variables tab is the current value of that variable as known to Oracle BI Server.

5. Test the Initialization Block Query

Click the Edit Data Source button to navigate to the Repository Variable Init Block Data Source dialog box.

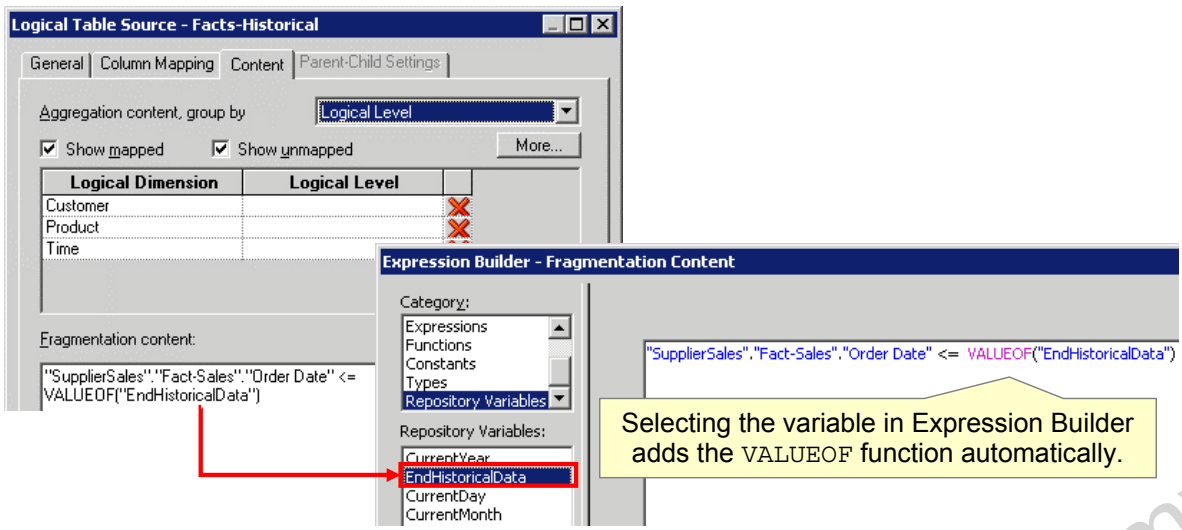


5. Test the Initialization Block Query

Return to the Repository Variable Initialization Block Data Source dialog box and click the Test button to test the initialization block query.

6. Use the Variable to Determine Content

Use the variable to dynamically determine the content in a fragmentation expression.



6. Use the Variable to Determine Content

Repository variables are available in the Repository Variables folder in the Expression Builder. Use the Expression Builder to add the variable to a formula in the Fragmentation content field of the `Facts-Historical` and `Facts-Recent` logical table sources.

Only `Facts-Historical` is shown here. This expression returns all orders in which the order date is less than or equal to the value of `EndHistoricalData`, which is the most recent period key from the `D1_ORDERS_HISTORICAL` table.

`Facts-Recent` returns all orders in which the order date is greater than the value of `EndHistoricalData`.

7. Verify the Initialization

Check the query log to verify that the variable is initialized and is used properly on Oracle BI Server startup.

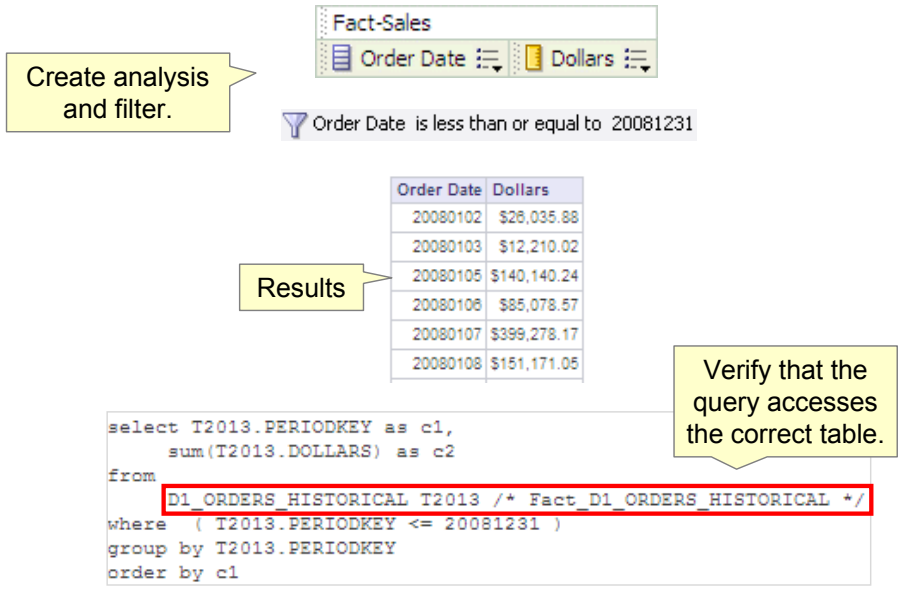
```
---------------------- An initialization block named 'OrderSplit', on behalf
of a Repository Variable, issued the following SQL query:

    SELECT MAX(PERIODKEY) from SUPPLIER2.D1_ORDERS_HISTORICAL

Returned 1 rows.  Query status: Successful Completion
```


7. Verify the Initialization

If you stop and restart the Oracle BI Server component, the server automatically executes the SQL in repository variable initialization blocks, reinitializing the repository variables.

8. Verify Your Work

Test your work to verify that an analysis accesses the correct fragment.



Summary

In this lesson, you should have learned how to:

- Create session variables
- Create repository variables
- Create initialization blocks
- Implement a dynamic repository variable

Practice 12-1 Overview: Creating Dynamic Repository Variables

This practice covers the following topics:

- Creating a repository variable initialization block
- Creating a dynamic repository variable
- Testing an initialization block
- Using a variable to dynamically determine fragmentation content

Practice 12-1 Overview: Creating Dynamic Repository Variables

ABC has implemented separate partitions for order data. However, instead of hard coding the data in the partitions, when the partitions are reloaded, historical data is loaded into the first partition and recent data is loaded into the second partition. You create an initialization block and a dynamic repository variable, and you then use the variable to determine how the data is split between the partitions.

Practice 12-2 Overview: Using Dynamic Repository Variables as Filters

This practice covers the following topics:

- Creating a repository variable initialization block
- Creating a dynamic repository variable
- Testing an initialization block
- Using dynamic repository variables as filters in analyses

Practice 12-2 Overview: Using Dynamic Repository Variables as Filters

Rather than creating hard-coded column filters such as `Year = 2009` in analyses, ABC wants to use variables that always return the current year, current month, and current day. You create these dynamic repository variables and then use them in column filters in analyses.

13

Modeling Time Series Data

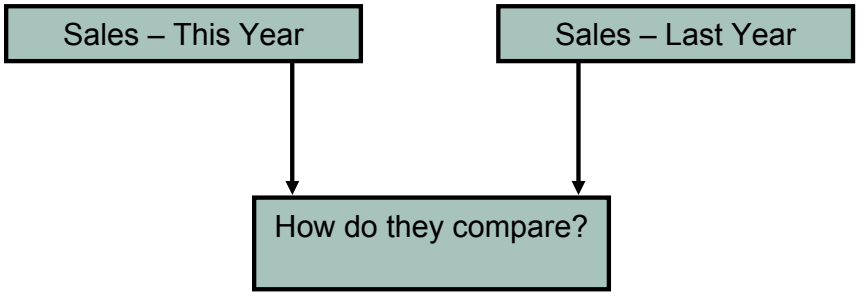
Objectives

After completing this lesson, you should be able to:

- Describe the use of time comparisons for business analysis
- Implement time comparison measures in the business model by using Oracle BI time series functions

Time Comparisons

Provide the ability to compare business performance with prior time periods



Time Comparisons

The ability to compare business performance with previous time periods is fundamental to understanding a business. Time comparisons enable businesses to analyze data that spans multiple time periods, providing a context for the data. Here is an example:

Question: What are the sales so far this year?

Answer: $50 million

Question: Is that good?

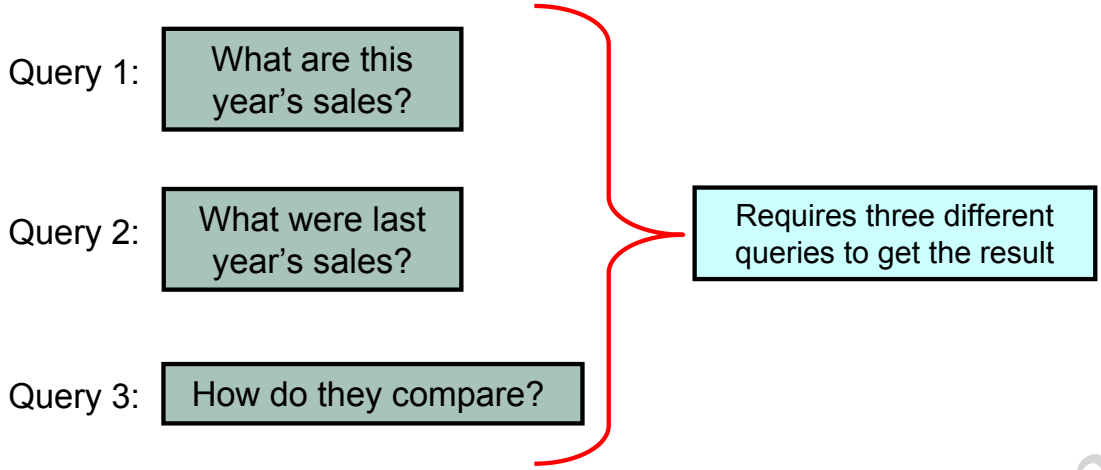
You need to ask three separate questions to answer the question, “Is that good?”

- What are this year’s sales?
- What were last year’s sales?
- How do they compare?

A business needs to make these comparisons across multiple time periods. For example, how do current sales compare to sales a year ago, a quarter ago, a month ago, and so on?

Business Challenge: Time Comparisons

There is no direct way in SQL to make time comparisons.



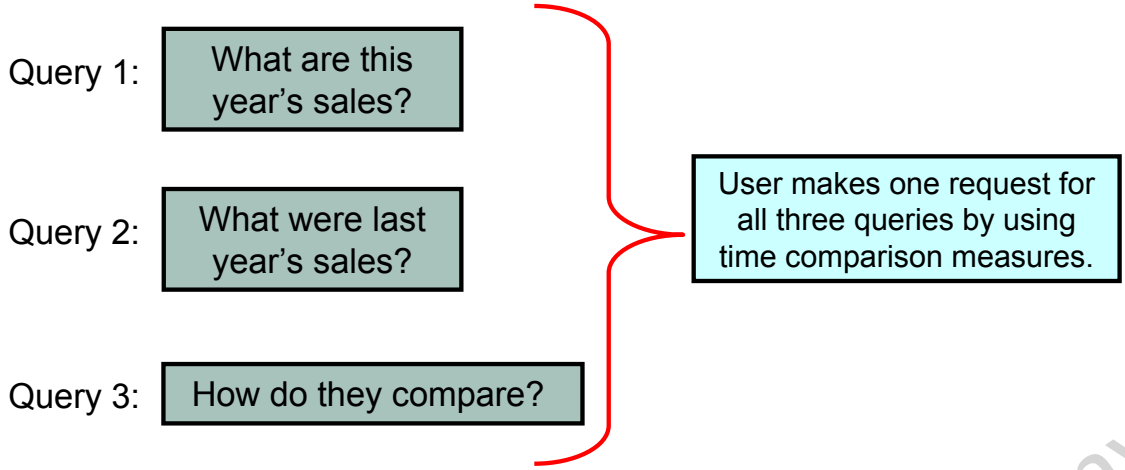
Business Challenge: Time Comparisons

SQL was not designed to make straightforward comparisons over time. There is no direct way in SQL to make time comparisons. For example, to compare this year's sales to last year's sales, you must run three separate queries:

- What are this year's sales?
- What were last year's sales?
- How do this year's sales compare to last year's sales?

Oracle BI Solution: Model Time Comparisons

Use the Oracle BI repository to model time series data.

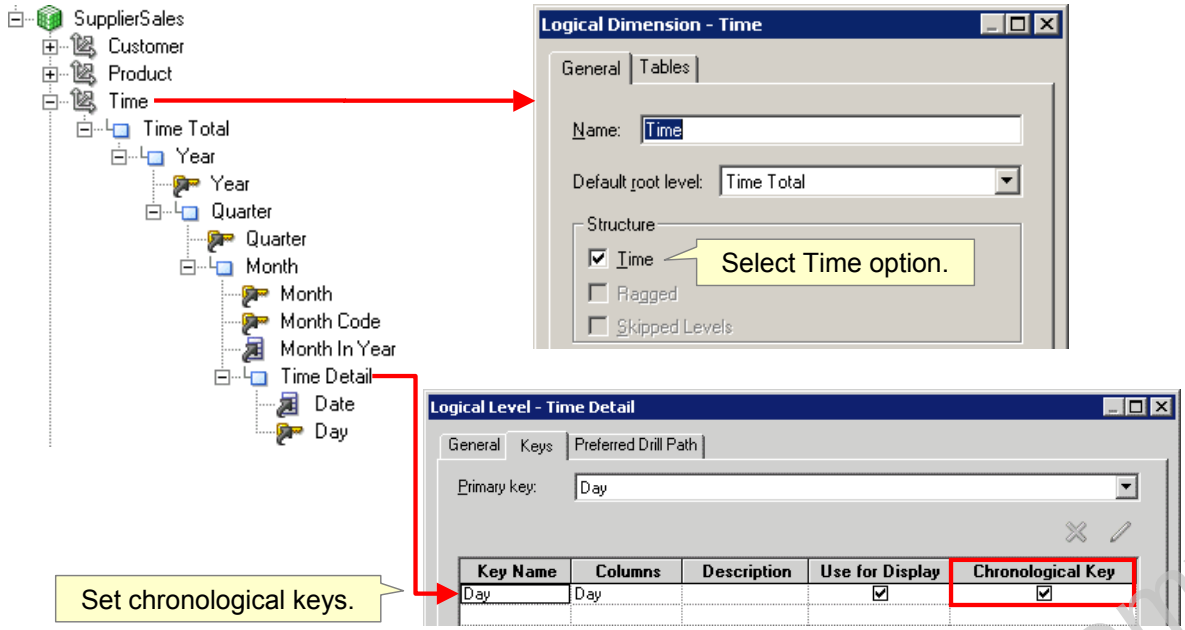


Oracle BI Solution: Model Time Comparisons

The solution is to model time series data in the Oracle BI repository. This enables users to make one request for the desired result. Oracle BI Server runs multiple queries in parallel to get the results. The queries that run in the background to support the time measure are transparent to the user.

Time Dimensions

Set up time dimensions based on the period table.



Time Dimensions

Because SQL does not provide a direct way to make time comparisons, you must model time series data in the Oracle BI repository.

The first step is to 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. The time series functions are described in more detail on the next slide.

Compared to modeling an ordinary dimension, the time dimension requires just two additional steps: selecting the Time option in the Logical Dimension dialog box and designating a chronological key for every level of the dimension hierarchy.

At query time, 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.

Time Series Functions

Oracle BI Server provides AGO, TODATE, and PERIODROLLING functions for time series comparisons:

- AGO
  - Calculates aggregated value as of some time period shifted from the current time
- TODATE
  - Aggregates a measure attribute from the beginning of a specified time period to the currently displayed time
- PERIODROLLING
  - Performs an aggregation across a specified set of query grain periods, rather than within a fixed time series grain

Time Series Functions

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.

Time series functions include AGO, TODATE, and PERIODROLLING. These functions let you use the Expression Builder to call a logical function to perform time series calculations instead of aliasing physical tables and modeling logically. The time series functions perform calculations based on the calendar tables in your data warehouse, rather than calculations based on standard SQL date manipulation functions.

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.

The TODATE function is used to aggregate a measure attribute from the beginning of the specified time period to the currently displayed time. For example, the TODATE function can calculate "Year to Date" sales on a quarterly, monthly, or weekly basis.

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."

Time Series Grains

Several different grains can be used in a time query:

- Query grain
  - Lowest time grain of the request
- Time Series grain
  - Grain at which the aggregation or offset is requested for both `AGO` and `TODATE` functions
- Storage grain
  - Grain of the source

	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

Time Series Grains

- **Query grain:** The lowest time grain of the request. In the example shown in the slide, 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 example shown in the slide, the time series grain is Quarter. Time series queries are valid only if the time series grain is at the query grain or longer. 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 the slide 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.

Time Series Grains (continued)

`Dollars Qago` is an example of the `AGO` function. It compares dollars to dollars a quarter ago.

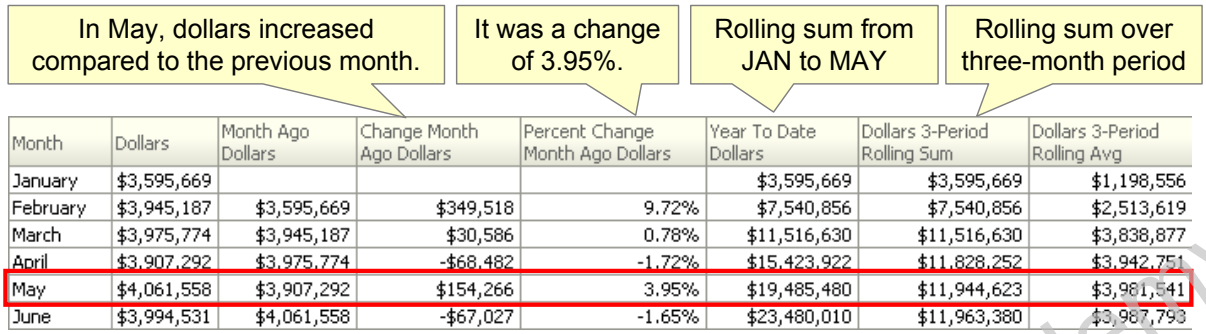
`Dollars QTD` is an example of the `TODATE` function. It accumulates dollars from the beginning of each quarter to the end of each quarter.

`Dollars 3-Period Rolling Sum` and `Dollars 3-Period Rolling Avg` are examples of the `PERIODROLLING` function. For example, for `Dollars 3-Period Rolling Sum`, the three-month rolling sum starts two periods in the past and includes the current period. That is, for the month 2008/07, the rolling sum includes 2008/05, 2008/06, and 2008/07.

ABC Example

ABC wants to implement the following time series measures:

- Month Ago Dollars
- Change Month Ago Dollars
- Percent Change Month Ago Dollars
- Year To Date Dollars
- Dollars 3-Period Rolling Sum
- Dollars 3-Period Rolling Avg



Month	Dollars	Month Ago Dollars	Change Month Ago Dollars	Percent Change Month Ago Dollars	Year To Date Dollars	Dollars 3-Period Rolling Sum	Dollars 3-Period Rolling Avg
January	$3,595,669				$3,595,669	$3,595,669	$1,198,556
February	$3,945,187	$3,595,669	$349,518	9.72%	$7,540,856	$7,540,856	$2,513,619
March	$3,975,774	$3,945,187	$30,586	0.78%	$11,516,630	$11,516,630	$3,838,877
April	$3,907,292	$3,975,774	-$68,482	-1.72%	$15,423,922	$11,828,252	$3,942,751
May	$4,061,558	$3,907,292	$154,266	3.95%	$19,485,480	$11,944,623	$3,981,541
June	$3,994,531	$4,061,558	-$67,027	-1.65%	$23,480,010	$11,963,380	$3,987,793

ABC Example

ABC wants to implement time series measures using the `AGO`, `TODATE`, and `PERIODROLLING` time series functions. Users can then build analyses with columns that show this month's dollar performance compared with last month's and calculate the change and percentage of change. Users can also display a running total of dollars on a month-to-month basis, a three-period rolling sum, and a three-period rolling average.

The slide shows an example of the results from an analysis and highlights calculations for the month of May. In May, dollars increased by $154,266 from the previous month. This was an increases of 3.95%. Since the beginning of the year, ABC has had $19,485,480 in sales. For the three-month period up to and including May (March, April, May), ABC has earned $11,944,623. The average monthly amount over this three-month period is $3,981,541.

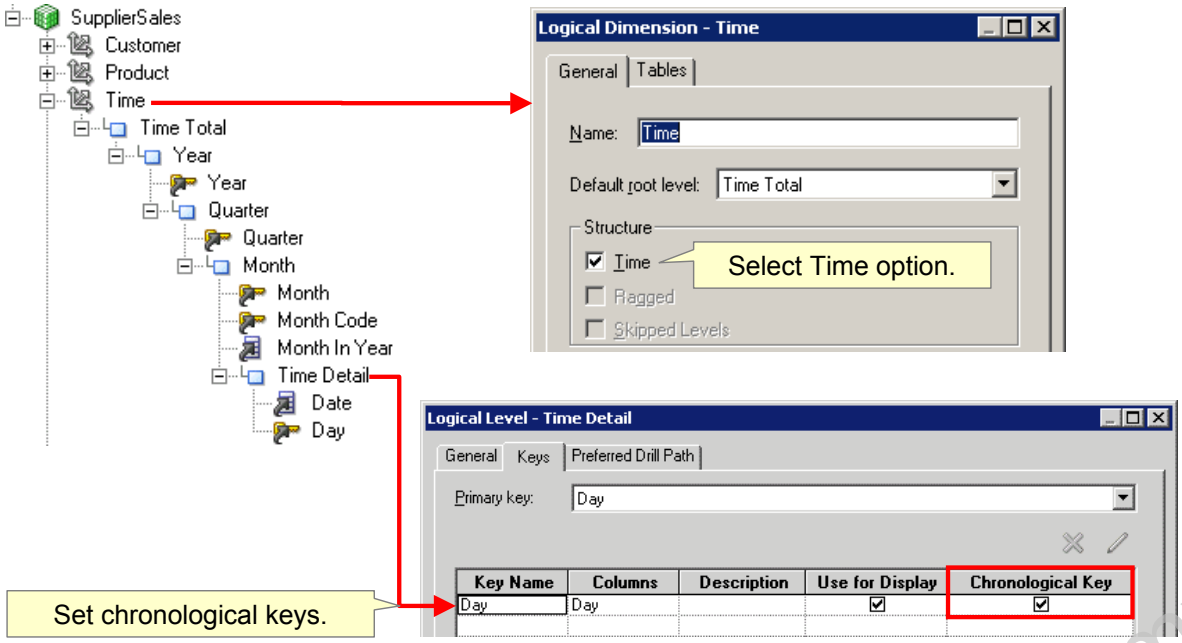
Steps to Model Time Series Data

1. Identify a time dimension and chronological keys.
2. Create a measure by using the `AGO` function.
3. Use a column with the `AGO` function to create additional measures.
4. Create a measure by using the `TODATE` function.
5. Create a measure by using the `PERIODROLLING` function.
6. Add new measures to the Presentation layer.
7. Test the results.

Steps to Model Time Series Data

These are the high-level steps to model time series data for the ABC example. Each step is covered in detail in the following slides.

1. Identify a Time Dimension and Chronological Keys



1. Identify a Time Dimension and Chronological Keys

As we have seen, compared to modeling an ordinary dimension, the time dimension requires just two additional steps: selecting the Time option in the Logical Dimension dialog box and designating a chronological key for every level of every dimension hierarchy. Only logical dimensions with the Time option selected can be used as the time dimension for the time series functions `AGO`, `TODATE`, and `PERIODROLLING`.

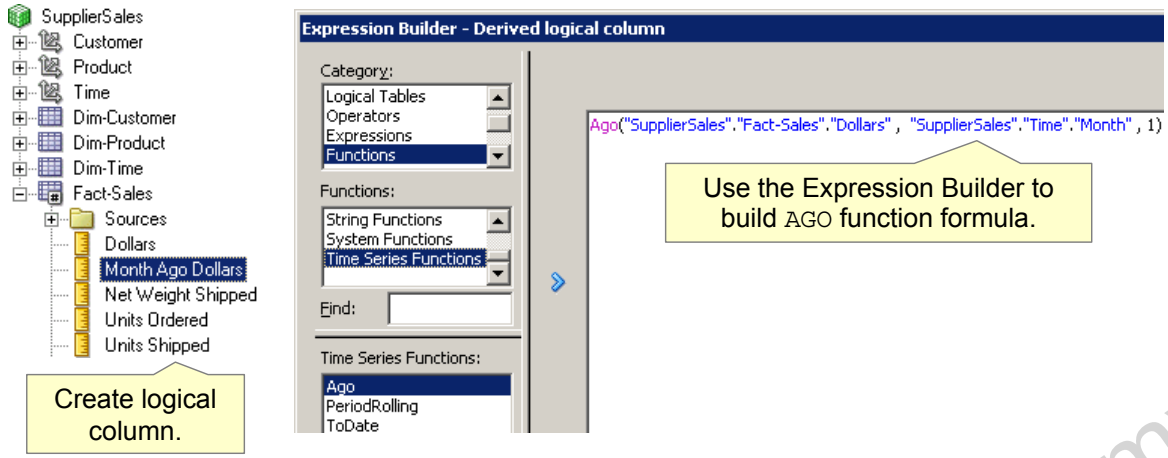
Designate a chronological key for every level of each dimension hierarchy. This key must meet the requirements of being sequential (the members have a natural order), cardinal (all members are spaced the same distance apart at a given level, such as day or month), and complete (no members missing). Oracle 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 allows 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. As with any level key, be sure the key is unique at its level. In this example, the Day column is set as the chronological key for the Time Detail level.

2. Create a Measure by Using the AGO Function

Use the Expression Builder to build a measure by using the AGO function with the following form:

```
Ago(<<Measure>>,<<Level>>,<<Number of Periods>>)
```



2. Create a Measure by Using the AGO Function

Create a new logical column (`Month Ago Dollars` in this example), and then use the Expression Builder to build the AGO function formula.

The AGO function offsets the time dimension to display data from a past period. In the Expression Builder, the AGO function has the following template:

```
Ago(<<Measure>>, <<Level>>, <<Number of Periods>>)
```

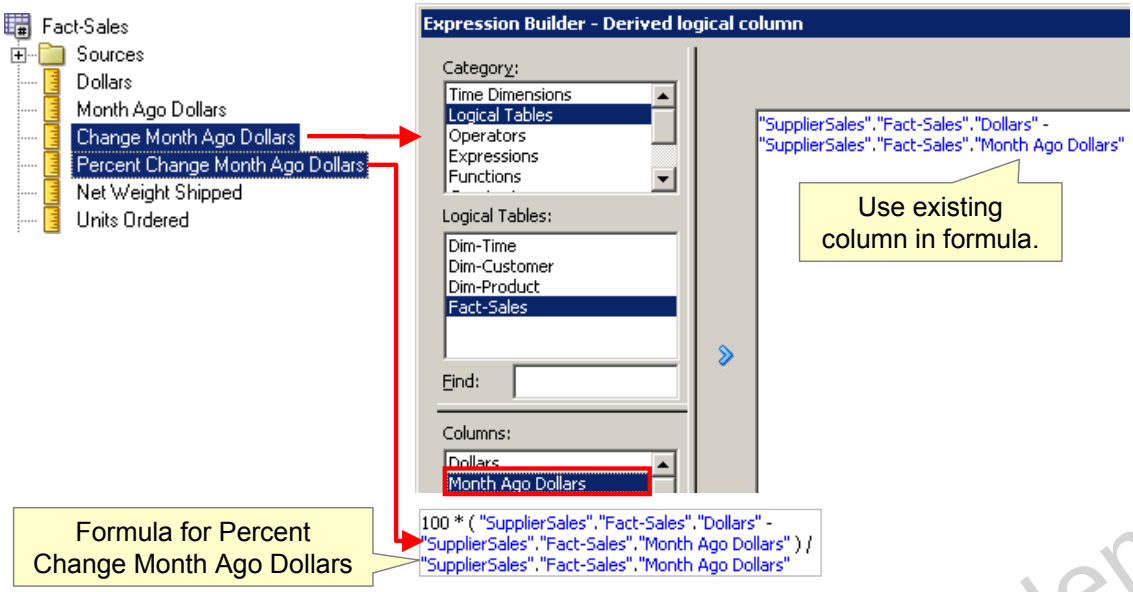
`<<Measure>>` represents the logical measure column from which you want to derive. In this example, you select the Dollars measure from your existing logical fact table.

`<<Level>>` is the optional time series grain you want to use. In this example, you select the Month level from the Time logical dimension.

`<<Number of Periods>>` is the size of the offset, measured in the grain you provided in the `<<Level>>` argument. In this example, the `<<Level>>` is Month and the `<<Number of Periods>>` is 1, so the function displays dollars from one month ago.

3. Use a Column with the AGO Function to Create Additional Measures

Use the existing column with the AGO function to build Change and Percent Change measures.



3. Use a Column with the AGO Function to Create Additional Measures

Use the existing column with the AGO function to build additional measures that use the AGO function in their calculation, such as Change Month Ago Dollars and Percent Change Month Ago Dollars.

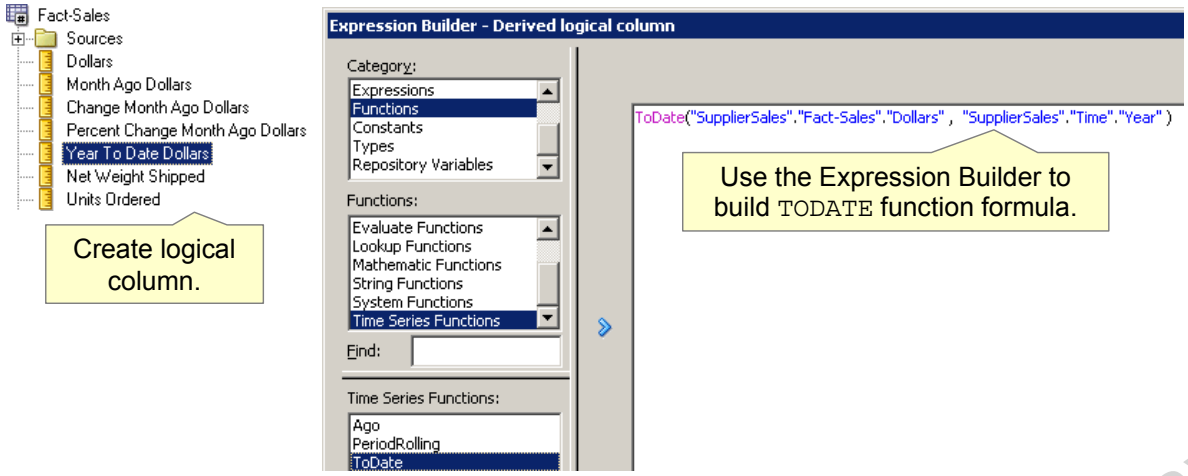
In the examples in the slide, the Month Ago Dollars measure is used to calculate the change in dollars between the current month and the previous month, and the percentage change in dollars between the current month and the previous month.

Note: You can use the Calculation Wizard to create these measures.

4. Create a Measure by Using the TODATE Function

Use the Expression Builder to build a measure by using the TODATE function with the following form:

`ToDate(<<Measure>>,<<Level>>)`



4. Create a Measure by Using the TODATE Function

Create a new logical column (`Year To Date Dollars` in this example) and then use the Expression Builder to build the TODATE function formula.

The TODATE function accumulates the measure from the beginning of the time series grain period to the current displayed query grain period. 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 select the Dollars measure from your existing logical fact table.

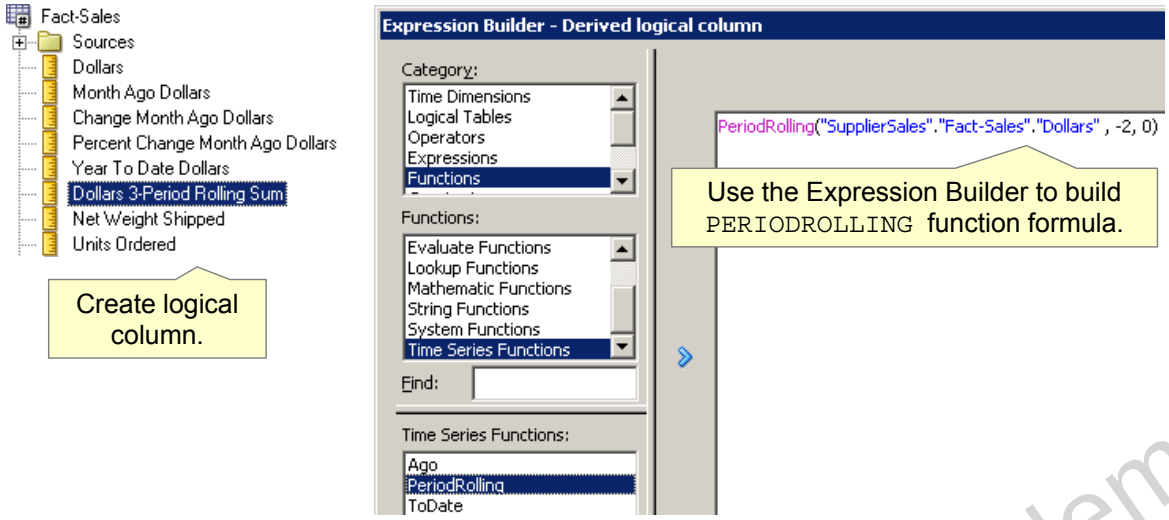
`<<Level>>` is the time series grain you want to use. In this example, you select Year from your time dimension.

In this example, the TODATE function calculates the Dollars measure from the beginning of the year to a specified time. For example, in a query with month as the time criteria, the Year To Date Dollars measure returns dollars calculated from the beginning of the year to the specified month.

5. Create a Measure by Using the PERIODROLLING Function

Use the Expression Builder to build a measure by using the PERIODROLLING function with the following form:

PeriodRolling(<<Measure>>, <<integer>>, <<integer>>)



5. Create a Measure by Using the PERIODROLLING Function

Create a new logical column (Dollars 3-Period Rolling Sum in this example), and then use the Expression Builder to build the PERIODROLLING function formula. The PERIODROLLING function lets you perform an aggregation across a specified set of query grain periods, rather than within a fixed time series grain. Note that because this function has no time series grain, the length of the rolling sequence is determined by the query grain. For example, the measure Dollars 3-Period Rolling Sum sums the last three months if the query grain is Month, but sums the last three years if the query grain is Year.

The PERIODROLLING function has the following template:

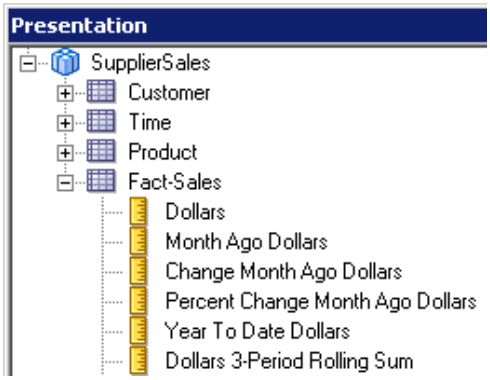
PeriodRolling(<<Measure>>, <<integer>>, <<integer>>)

<<Measure>> represents the logical measure column from which you want to derive. To create the Dollars 3-Period Rolling Sum measure, select the Dollars measure from your existing logical fact table.

<<integer>> and <<integer>> identify the first period and last period used in the rolling aggregation, respectively. Each integer is the relative number of periods from the displayed period. In this example, if the query grain is month, the three-month rolling sum starts two periods in the past and includes the current period. That is, for the 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 in the formula are –2 and 0.

6. Add New Measures to the Presentation Layer

Add new time series measures to the Presentation layer so that users can include them in query criteria.



6. Add New Measures to the Presentation Layer

Use known techniques to add the new time series measures to the Presentation layer so that users can include the measures in query criteria. In this example, five new measures have been added:

- Month Ago Dollars
- Change Month Ago Dollars
- Percent Change Month Ago Dollars
- Year To Date Dollars
- Dollars 3-Period Rolling Sum

7. Test the Results

Create analyses and verify the results.

Month	Dollars	Month Ago Dollars	Change Month Ago Dollars	Percent Change Month Ago Dollars	Year To Date Dollars	Dollars 3-Period Rolling Sum
January	$3,595,669				$3,595,669	$3,595,669
February	$3,945,187	$3,595,669	$349,518	9.72%	$7,540,856	$7,540,856
March	$3,975,774	$3,945,187	$30,586	0.78%	$11,516,630	$11,516,630
April	$3,907,292	$3,975,774	-$68,482	-1.72%	$15,423,922	$11,828,252
May	$4,061,558	$3,907,292	$154,266	3.95%	$19,485,480	$11,944,623
June	$3,994,531	$4,061,558	-$67,027	-1.65%	$23,480,010	$11,963,380
July	$4,062,212	$3,994,531	$67,681	1.69%	$27,542,222	$12,118,301
August	$4,242,611	$4,062,212	$180,399	4.44%	$31,784,833	$12,299,354
September	$3,810,263	$4,242,611	-$432,348	-10.19%	$35,595,096	$12,115,086
October	$4,596,372	$3,810,263	$786,109	20.63%	$40,191,468	$12,649,246
November	$3,655,169	$4,596,372	-$941,203	-20.48%	$43,846,637	$12,061,804
December	$3,997,616	$3,655,169	$342,448	9.37%	$47,844,253	$12,249,157

Summary

In this lesson, you should have learned how to:

- Describe the use of time comparisons for business analysis
- Implement time comparison measures in the business model by using the time series functions

Practice 13-1 Overview: Creating Time Series Comparison Measures

This practice covers using the Oracle BI `AGO`, `TODATE`, and `PERIODROLLING` functions to build time series comparison measures.

Practice 13-1 Overview: Creating Time Series Comparison Measures

You use the Oracle BI time series functions to create new measures to enable users to compare current dollar performance to previous time periods. You then add the new measures to the presentation catalog and test them using analyses.

14

Modeling Many-to-Many Relationships

Objective

After completing this lesson, you should be able to use a bridge table to model a many-to-many relationship in an Oracle BI repository.

Business Challenge and Solution

- Dimensional star schemas are ideal for modeling a business when one-to-many relationships exist between the dimension tables and fact tables.
- Challenge: It is often necessary to model many-to-many relationships between dimension tables and fact tables.
- Solution: Use a bridge table to model many-to-many relationships.

Bridge Table

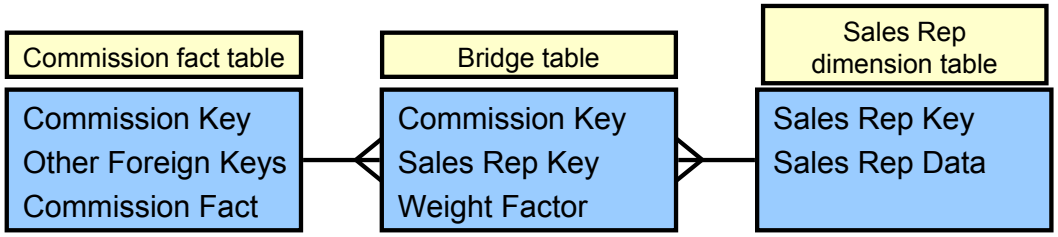
- Resolves many-to-many relationships between dimension tables and fact tables
- Stores multiple records corresponding to a dimension
- Contains a weight factor column representing the ratio of the many-to-many relationship
  - For example, if two sales representatives are associated with a given sales commission, the weight factor for each representative would be 0.50.
  - Multiplying the weight factor by the commission amount yields each sales representative's share of the commission.
  - More complex weight factors can be used (for example, 0.50, 0.25, 0.25) as long as the sum of all factors is 1.

Bridge Table

When you need to model many-to-many relationships between dimension tables and fact tables, you can create a bridge table that resides between the fact and dimension tables. A bridge table stores multiple records corresponding to that dimension.

ABC Example

- Each sales representative may participate in many deals that pay commission.
- Each deal may include many sales representatives who split the commission.
- A bridge table is required to model this many-to-many relationship between the commission fact table and the sales representative dimension table.



ABC Example

In the example for this lesson, you model a bridge table to resolve a many-to-many relationship between a commission fact table and a sales representative dimension table.

Steps to Model a Bridge Table

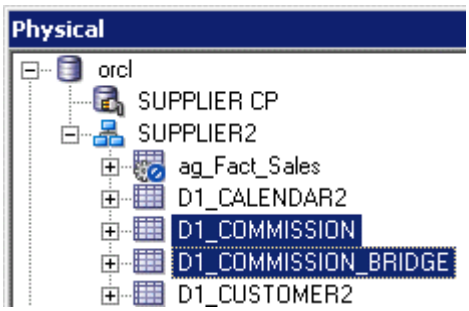
1. Import tables.
2. Create the physical model.
3. Create the logical model.
4. Map the bridge table.
5. Create a calculation measure.
6. Map objects to the Presentation layer.
7. Verify the results.

Steps to Model a Bridge Table

This slide identifies the steps needed to model a bridge table in an Oracle BI repository. Each step is presented in detail in the following slides.

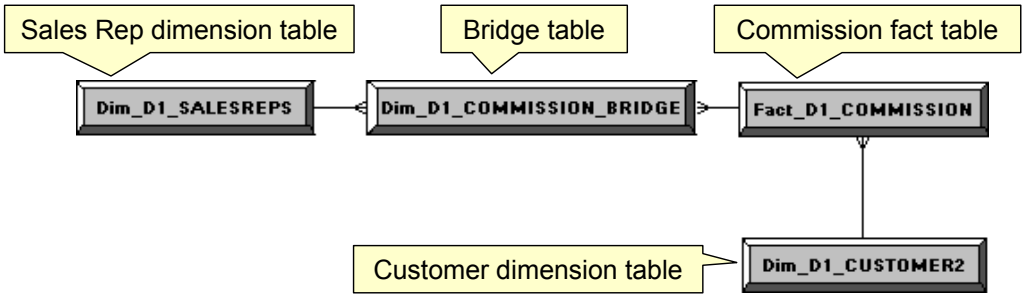
1. Import Tables

Use known techniques to import the commission fact and commission bridge tables to the Physical layer.



2. Create the Physical Model

Use the bridge table to model a many-to-many relationship between the commission fact and the sales representative dimension in the Physical layer.



2. Create the Physical Model

Create aliases and build the physical model.

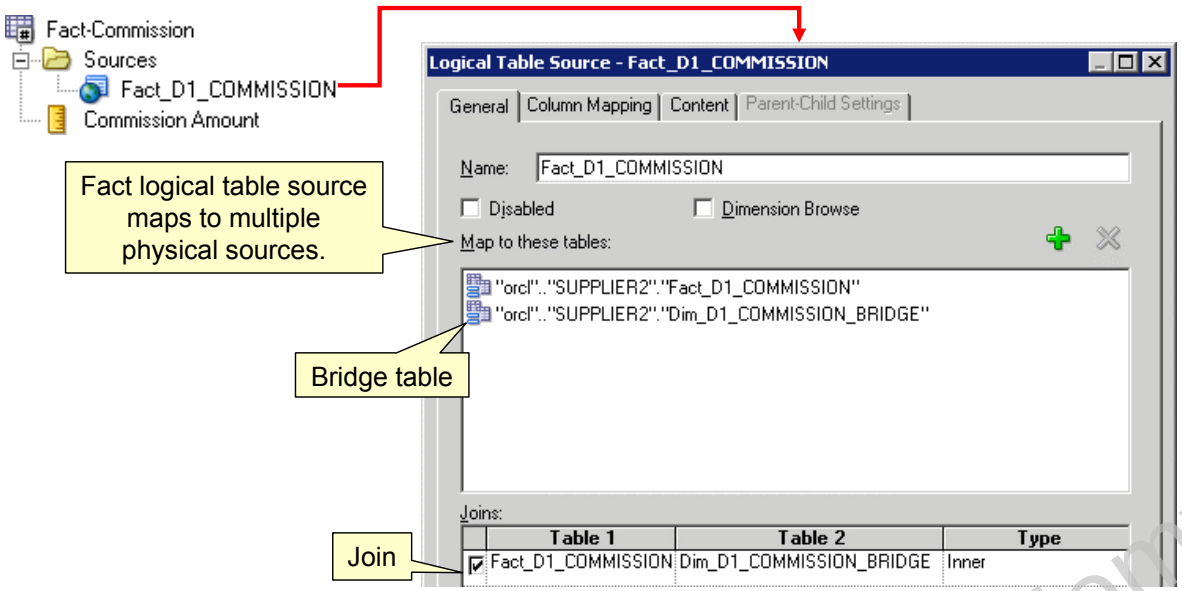
In this example, notice the one-to-many relationships between `Fact_D1_COMMISSION`, `Dim_D1_SALESREPS` and the `Dim_D1_COMMISSION_BRIDGE` bridge table.

3. Create the Logical Model

Use known techniques to build the logical model in the Business Model and Mapping layer.

4. Map the Bridge Table

Map the fact logical table source to the bridge table.



4. Map the Bridge Table

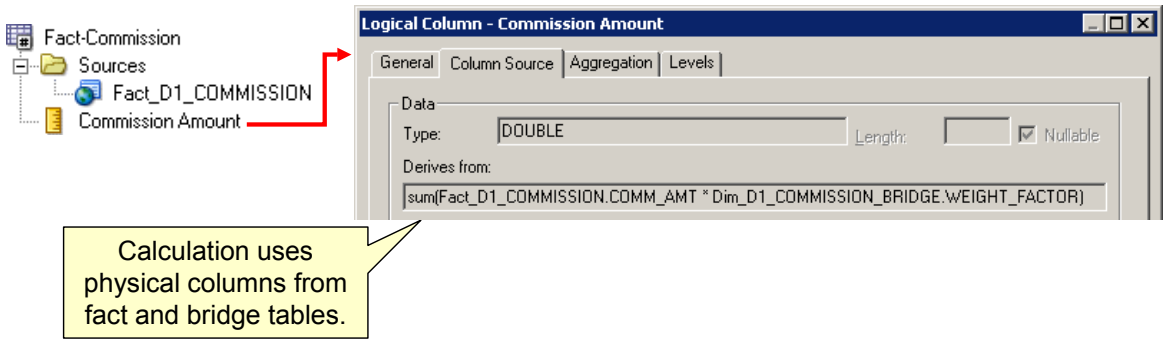
Mapping the fact logical table source to the bridge table avoids the need for snowflaking in the BMM layer.

In the example, two tables are added to one logical table source in the BMM layer. If the bridge table had been added to the business model as a separate logical table, it would have been necessary to join the tables in a logical snowflaked schema.

Using this technique of mapping the logical table source to the bridge table enables you to build models that use only star schemas.

5. Create a Calculation Measure

Use physical columns to create a measure that calculates "commission amount" × "weight factor."

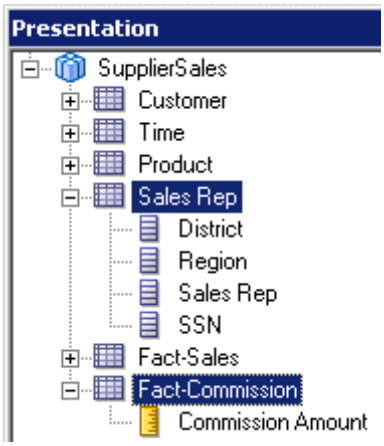


5. Create a Calculation Measure

In this example, you use physical columns to create a measure that calculates the sum of the commission amount in the commission fact table multiplied by the weight factor in the bridge table.

6. Map Objects to the Presentation Layer

Use known techniques to map objects to the Presentation layer.



7. Verify the Results

Use an analysis and the query log to verify the results.

Customer	Sales Rep	Commission Amount
Mayflower Cuisinier	ALAN ZIFF	$22,000
	ANDREW TAYLOR	$22,000

Commission is split between sales reps based on weight factor.

```
select T547.NAME as c1,
     T547.NEWKEY as c2,
     T1214.SALESREP as c3,
     sum(T2069.COMM_AMT * T2075.WEIGHT_FACTOR) as c4
from
     D1_SALESREPS T1214 /* Dim_D1_SALESREPS */ ,
     D1_CUSTOMER2 T547 /* Dim_D1_CUSTOMER2 */ ,
     D1_COMMISSION T2069 /* Fact_D1_COMMISSION */ ,
     D1_COMMISSION_BRIDGE T2075 /* Dim_D1_COMMISSION_BRIDGE */
where  ( T547.NEWKEY = T2069.CUST_KEY and T1214.SALESREP = T2075.SALESREP
group by T547.NAME, T547.NEWKEY, T1214.SALESREP
order by c1, c2, c3
```

Calculation

Bridge table accessed

Summary

In this lesson, you should have learned how to use a bridge table to model a many-to-many relationship in an Oracle BI repository.

Practice 14-1 Overview: Modeling a Bridge Table

In this practice, you model a bridge table to resolve a many-to-many relationship between a fact table and a dimension table.

Practice 14-1 Overview: Modeling a Bridge Table

An ABC sales representative may participate in many deals that pay commission. Additionally, each deal may include many sales representatives so that each sales representative receives a percentage of the commission. You need to model this many-to-many relationship in the repository.

There is a `D1_COMMISSION` fact table with commissions paid per invoice. `D1_COMMISION_BRIDGE` is the bridge table used to create a many-to-many relationship between the `D1_COMMISSION` fact table and the `D1_SALESREP` dimension table. `D1_COMMISION_BRIDGE` includes a weight factor to calculate the weighted distribution of commissions among sales teams.

You import the tables and model them in the repository.

15

Localizing Oracle BI Metadata

Objective

After completing this lesson, you should be able to configure Oracle Business Intelligence metadata to support multilingual environments.

Business Challenges and Solution

Challenges:

- Companies require multilingual support for global deployments of Oracle Business Intelligence.
- End users need to make decisions based on applications and data presented in their own language.

Solution:

- Add multilingual support to Oracle BI.

Business Challenges and Solution

Companies often require multilingual support for their deployment of Oracle BI. At the minimum, end users need to be able to view applications and data in their own language. Oracle BI provides the ability to localize both data and repository metadata.

Oracle BI Multilingual Support

Requires three types of configurations:

- Repository metadata, such as presentation folders

Focus of this lesson

- Database data, such as product names

Focus of next lesson

- Report and dashboard metadata, such as chart labels

Covered in a separate course

Oracle BI Multilingual Support

Multilingual support requires three types of configurations. The focus of this lesson is the localization of Oracle BI repository metadata. The focus of the next lesson, “Localizing Oracle BI Data,” is the localization of Oracle BI data. The localization of reports and dashboards is covered in a separate course: *Oracle BI 11g R1: Create Analyses and Dashboards*.

Configuring Oracle BI Metadata

Involves configuring names and descriptions of repository metadata:

- Uses a translation table to store language-specific names and descriptions for metadata
- Uses initialization blocks and variables to select the preferred language of the user and the corresponding names and descriptions

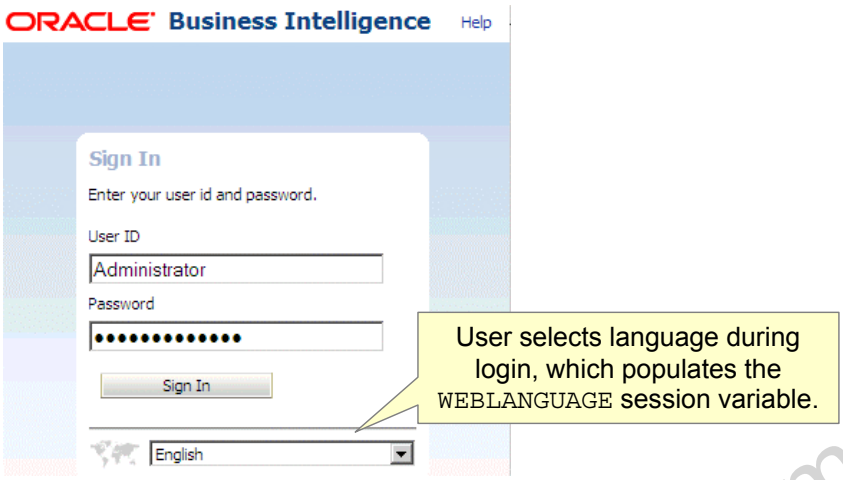
Configuring Oracle BI Metadata

Configuring Oracle BI repository metadata requires the use of a translation table to store language-specific names and descriptions, as well as initialization blocks and variables. Initialization blocks and variables are used in conjunction with the translation table to select the user's preferred language to display the correct metadata names and descriptions.

WEBLANGUAGE Session Variable

Is populated with a two-character, lowercase language code when a user selects a language during login

- Example: `en` for English



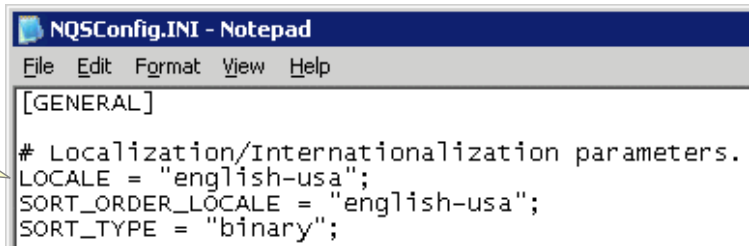
WEBLANGUAGE Session Variable

The `WEBLANGUAGE` session variable is populated with a two-character, lowercase language code when a user selects a language during login. This variable is not populated until the user logs in, so there is no way to test it from within the Oracle BI Administration Tool.

LOCALE Configuration Setting

- Specifies how data is returned from the server
- Localizes message strings, such as names of days and months

Default LOCALE is specified in the NQSConfig.ini file.



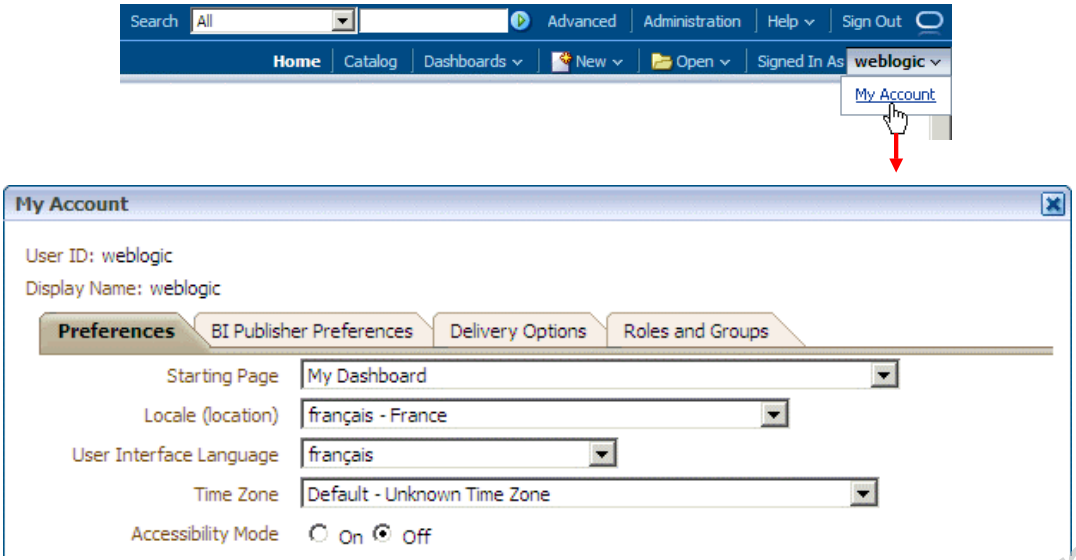
ORACLE

LOCALE Configuration Setting

The LOCALE configuration setting specifies how data is returned from the Oracle BI Server. It localizes message strings , such as the names of days and months. The default LOCALE is specified in the General section of the NQSConfig.ini file.

Changing Localization Preferences

Users can change localization preferences on the My Account page.

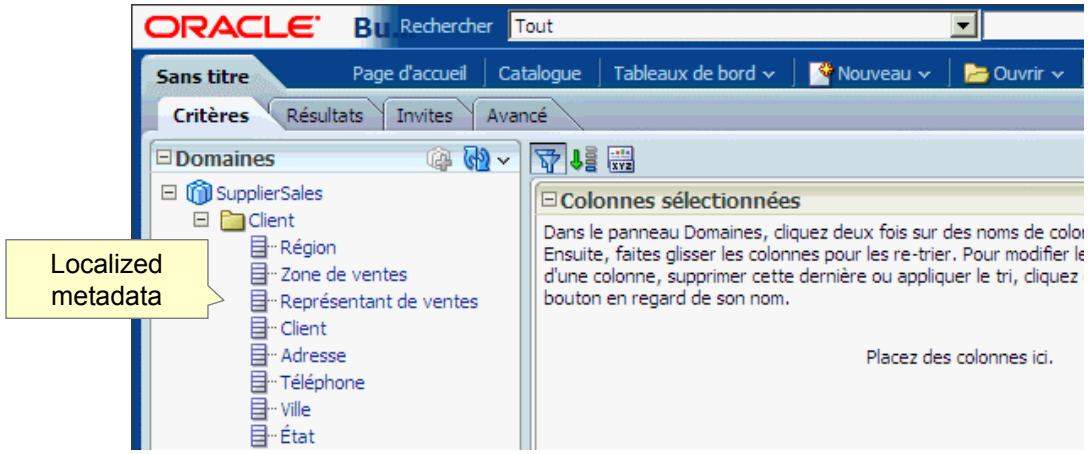


Changing Localization Preferences

Users can easily change their localization preferences by selecting Settings > My Account in the Oracle BI user interface.

ABC Example

Localize metadata to the French language.



ABC Example

In the example for this lesson, you localize Oracle BI repository metadata to the French language. In this example, the Customer table and its columns are localized.

Steps to Localize Metadata

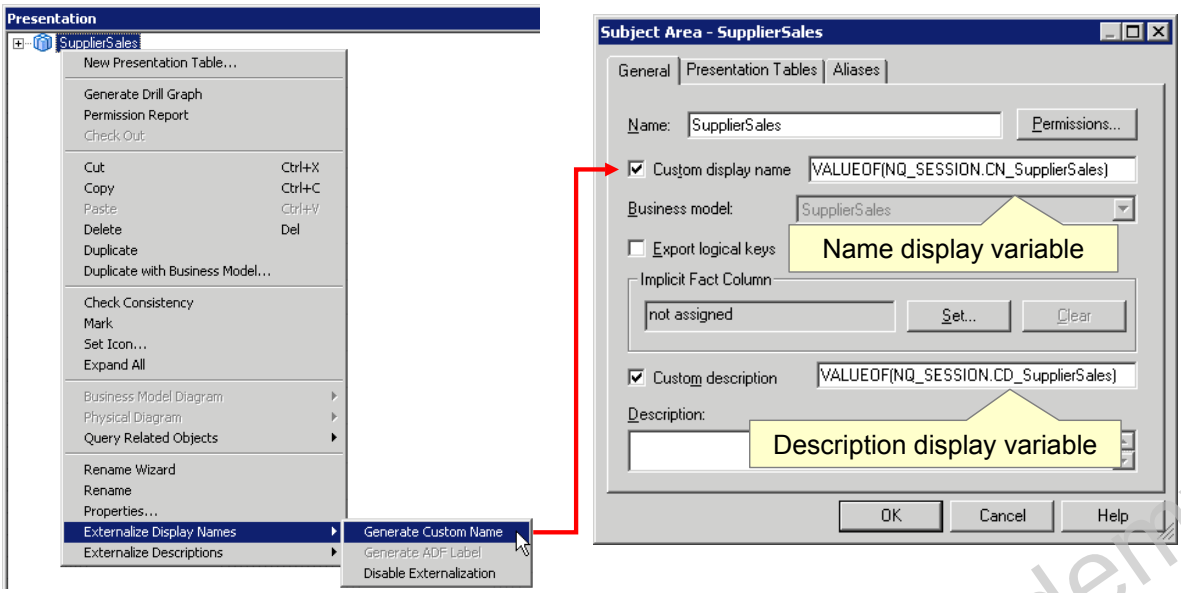
1. Externalize metadata objects.
2. Run the Externalize Strings utility.
3. Modify the translation file.
4. Load the translation table.
5. Import the translation table.
6. Create an initialization block.
7. Modify My Account preferences.
8. Verify the translations.

Steps to Localize Metadata

This slide identifies the steps to localize Oracle BI repository metadata. Each step is presented in detail in the following slides.

1. Externalize Metadata Objects

To set display variables, select Externalize Display Names or Externalize Descriptions for each Presentation layer object that requires translation.



1. Externalize Metadata Objects

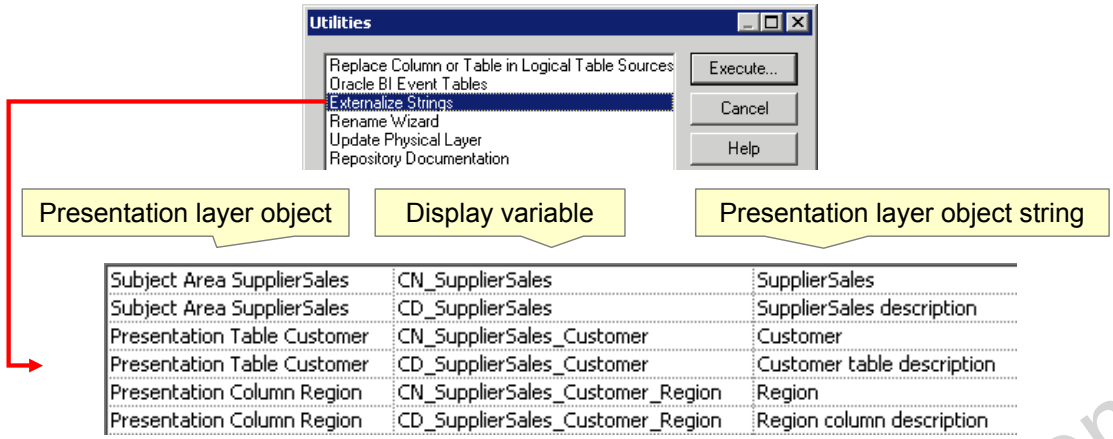
To set display variables, right-click any Presentation layer object, such as a subject area, presentation table, or presentation column. Then select either Externalize Display Names > Generate Custom Names or Externalize Descriptions > Generate Custom Descriptions to externalize strings.

Selecting one of these right-click externalization options automatically selects the Custom display name or Custom description options in the Properties dialog box for the selected object and all of its child objects. For example, if you right-click a subject area and select one of the externalization options, the externalization flag is set on all presentation tables, columns, hierarchies, and levels within that subject area.

The object display name and description values change to variables, which are populated by an initialization block and displayed in a query user interface. In this example, the `SupplierSales` subject area display name is now populated by the session variable `CN_SupplierSales`, and the `SupplierSales` description is populated by the session variable `CD_SupplierSales`.

2. Run the Externalize Strings Utility

- Exports the names of the Presentation layer objects to a file in comma-separated value, tab-delimited, or XML format
- Externalizes the Presentation layer strings to enable setup of local language translations for them



2. Run the Externalize Strings Utility

After you select Externalize Display Names and Externalize Descriptions, you can run the Externalize Strings utility. The Externalize Strings utility is primarily for use by translators to translate Presentation layer catalogs, tables, columns, and their descriptions into other languages. You can save these text strings to a comma-separated, tab-delimited, or XML external file with ANSI, Unicode, and UTF-8 coding options.

The screenshot shows the name of the Presentation layer object, the display variable (see previous slide), and the Presentation layer object string.

3. Modify the Translation File

- Modify the file generated by the Externalize Strings utility to include language translations:
  - Add a row for each language and display variable.
- Include required columns needed for translation table:
  - `LANG_ID` identifies the language.
  - `MSG_NUM` identifies the display variable to be translated.
  - `MSG_TEXT` contains the translated text.

Add row for each language and display variable.

METADATA_OBJECT	MSG_NUM	MSG_TEXT	LANG_ID
Presentation Table Customer	CN_SupplierSales_Customer	Customer	en
Presentation Table Customer	CN_SupplierSales_Customer	Client	fr
Presentation Column Region	CN_SupplierSales_Customer_Region	Region	en
Presentation Column Region	CN_SupplierSales_Customer_Region	Région	fr
Presentation Column Sales District	CN_SupplierSales_Customer_Sales_District	Sales District	en
Presentation Column Sales District	CN_SupplierSales_Customer_Sales_District	Zone de ventes	fr
Presentation Column Sales Rep	CN_SupplierSales_Customer_Sales_Rep	Sales Rep	en
Presentation Column Sales Rep	CN_SupplierSales_Customer_Sales_Rep	Représentant de ventes	fr

3. Modify the Translation Table

After running the Externalize Strings utility to generate a file, you modify the file to include language translations. You must add a row for each language and display variable. For example, if there are 70 display variables and two languages, 140 rows are required. In this example, there are one English row and one French row for each display variable. If there were three languages, 210 rows would be required, and so on.

The translation table has three required columns:

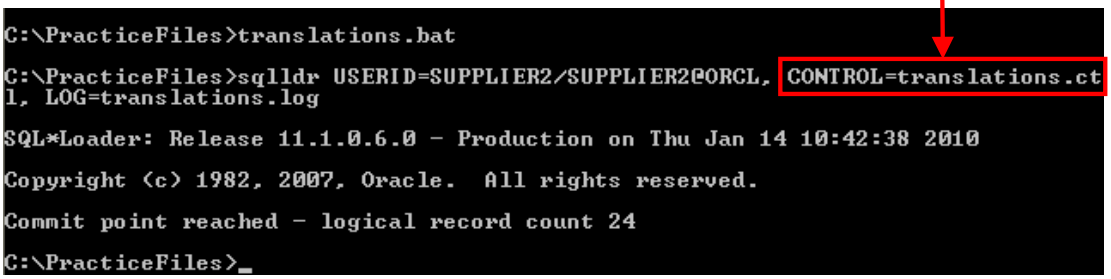
- **`LANG_ID`:** Identifies the language. It is typically a two-character, lowercase code. In this example, `fr` is used to indicate a French translation.
- **`MSG_NUM`:** Identifies the display variable to translate. One `MSG_NUM` row is required for each presentation catalog name, presentation catalog description, table name, table description, column name, and column description. The syntax is `CX_[LABEL NAME]`, where `X` is `N` (for name) or `D` (for description). In the example in the slide, `CN_SupplierSales` is the display variable for the `SupplierSales` presentation catalog name and `CD_SupplierSales` is the display variable for the `SupplierSales` presentation catalog description.
- **`MSG_TXT`:** Contains the translated text

In the example in the slide, there is one additional, nonrequired column, `METADATA_OBJECT`, which is used to help identify each object.

4. Load the Translation Table

Use SQL Loader to load the translation file into an existing table.

```
load data
infile 'translations.csv'
badfile 'translations.bad'
discardfile 'translations.dsc'
into table TRANSLATIONS
FIELDS TERMINATED BY ','
(METADATA_OBJECT, MSG_NUM, MSG_TEXT, LANG_ID)
```



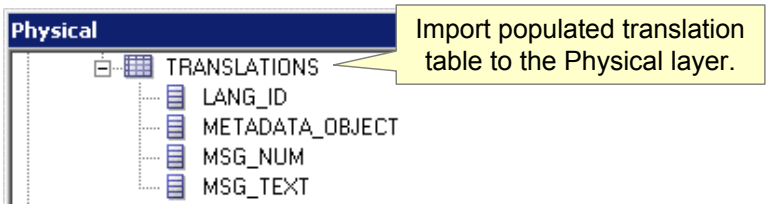
4. Load the Translation Table

For Oracle database, use SQL Loader to load the `.csv` file into an existing table. In this example, a table named `TRANSLATIONS` has already been created for you in the `SUPPLIER2` schema, and a translation batch file, `translations.bat`, has already been created to load the file.

The screenshot shows an example of running the `translations.bat` file, which starts the SQL Loader utility and uses a control file to load the data from the `TRANSLATIONS.csv` file into the `TRANSLATIONS` table in the `SUPPLIER2` schema in the database.

5. Import the Translation Table

Import the translation table to the Physical layer to give Oracle BI Server access to the translation information.

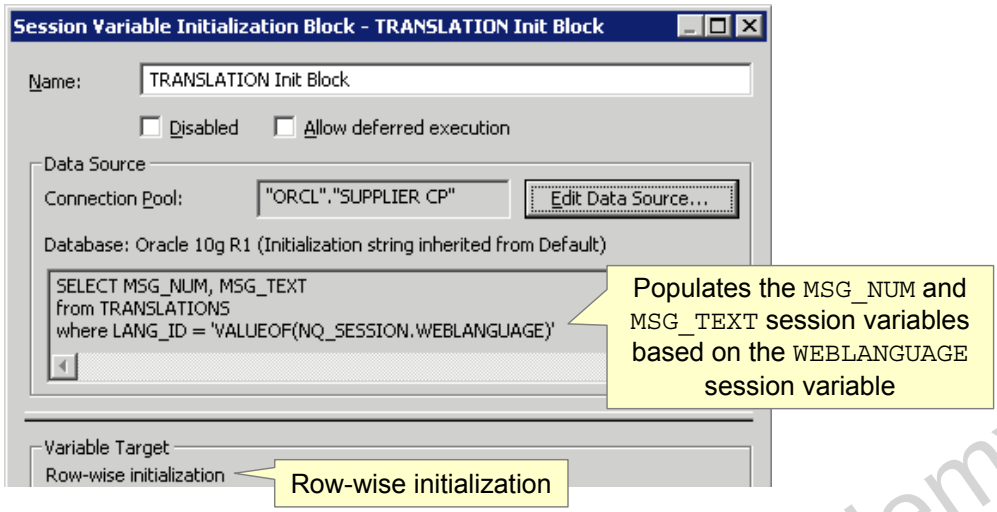


5. Import the Translation Table

Use known techniques to import the translation table to the Physical layer of the repository. This is an optional step.

6. Create an Initialization Block

Create a row-wise initialization block that populates session variables with values from the translation table based on the value of WEBLANGUAGE.



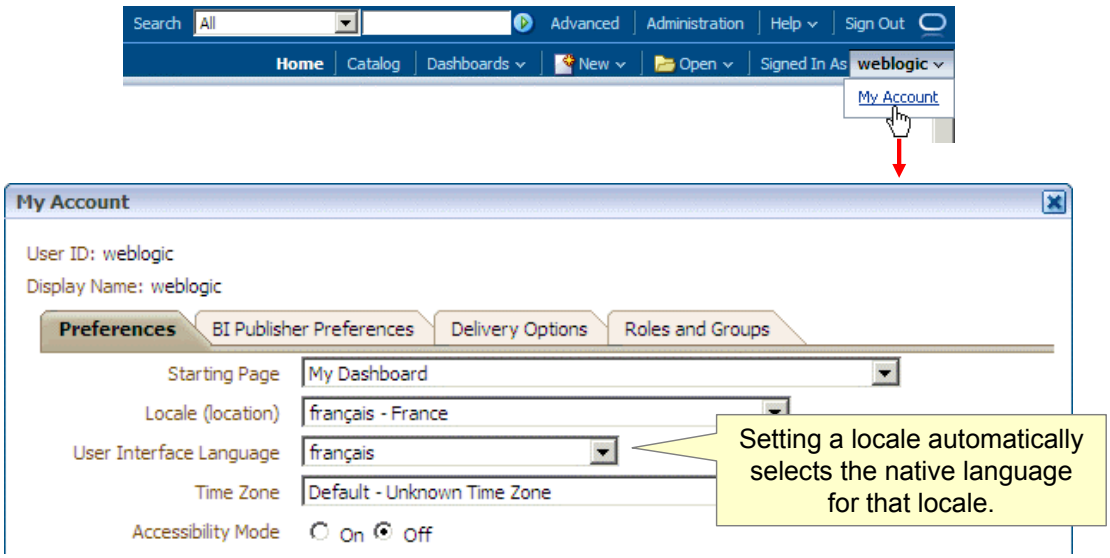
6. Create an Initialization Block

Create an initialization block that assigns the value of the WEBLANGUAGE session variable to a variable. WEBLANGUAGE is a session variable that is initialized automatically, based on the language selected when a user logs in. WEBLANGUAGE is set to the language of the user's browser when a user first logs in to an integrated Oracle BI application. For example, if a user with a browser language set to French logs in to Oracle Business Intelligence for the first time, the value for WEBLANGUAGE is French, and the metadata is translated to French.

The row-wise initialization option allows you 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 enter in the Session Variable Initialization Block Data Source dialog box. In this example, the initialization block populates the MSG_NUM and MSG_TEXT session variables by using data from the TRANSLATIONS table.

7. Modify My Account Preferences

Set locale or language on the My Account page.

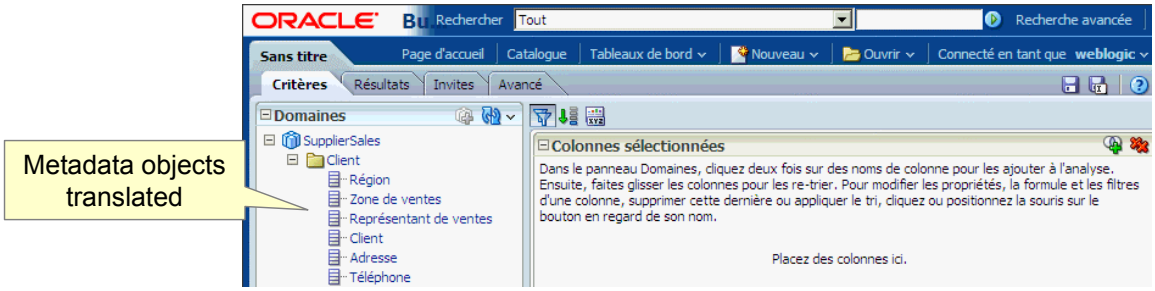


7. Modify My Account Preferences

To verify results, change the localization preferences on the My Account page.

8. Verify the Translations

- Verify that objects are translated as expected.



- Check `NQQuery.log` and verify that the initialization block issued a SQL query.

```
-------------------- An initialization block named
'TRANSLATIONS', on behalf of a Session Variable, issued the
following SQL query:

    SELECT MSG_NUM, MSG_TEXT
FROM TRANSLATIONS
WHERE 'VALUEOF(NQ_SESSION.WEBLANGUAGE)' = LANG_ID

Returned 230 rows.  Query status: Successful Completion
```

ORACLE

8. Verify the Translations

Open Oracle BI and verify that the metadata is translated as expected. Check the `NQQuery.log` file to verify that the initialization block executed the expected query.

Summary

In this lesson, you should have learned how to configure Oracle BI metadata to support multilingual environments.

Practice 15-1 Overview: Localizing Repository Metadata

In this practice, you localize Oracle BI repository metadata.

Practice 15-1 Overview: Localizing Repository Metadata

You want to display metadata in languages other than English. To do this, you perform the following steps.

You first run the Externalize Strings repository utility to export Presentation layer object strings to a file in comma-separated format. You then modify the file to include translated strings. Finally, you load the data from the file into a database table, create initialization blocks to populate session variables, modify localization preferences, and check the results in Oracle BI and in `NQQuery.log`.

16

Localizing Oracle BI Data

Objective

After completing this lesson, you should be able to localize Oracle BI data to support multilingual environments.

Business Challenges and Solution

Challenges:

- Companies require multilingual support for global deployments of Oracle BI.
- Users need to make decisions based on applications and data presented in their own language.

Solution:

- Add multilingual data support to Oracle BI.

Business Challenges and Solution

Companies often require multilingual support for their deployment of Oracle BI. At a minimum, end users need to be able to view applications and data in their own language. Oracle BI provides the ability to localize both data and repository metadata.

Oracle BI Multilingual Data Support

Requires three types of configurations:

- Repository metadata, such as presentation folders

Focus of last lesson

- Database data, such as product names

Focus of this lesson

- Analysis and dashboard metadata, such as chart labels

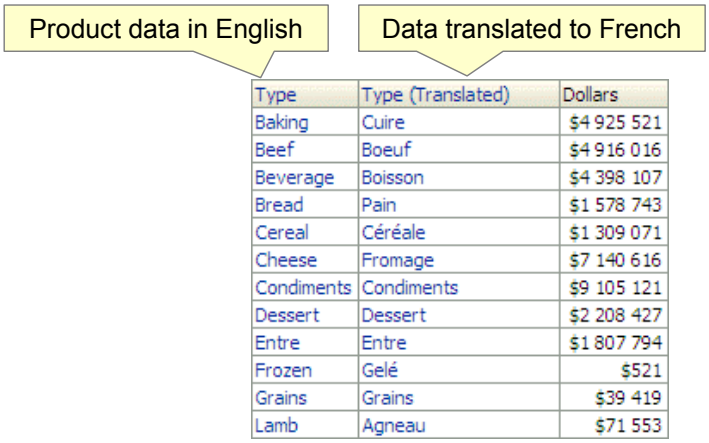
Covered in a separate course

Oracle BI Multilingual Data Support

Multilingual support requires three types of configurations. The focus of the lesson titled "Localizing Oracle BI Metadata" was the localization of Oracle BI repository metadata. The focus of this lesson is the localization of Oracle BI data. The localization of analyses and dashboards is covered in a separate course: *Oracle BI 11g R1: Create Analyses and Dashboards*.

What Is Multilingual Data Support?

Multilingual data support is the ability to display data from database schemas in multiple languages.



Type	Type (Translated)	Dollars
Baking	Cuire	$4 925 521
Beef	Boeuf	$4 916 016
Beverage	Boisson	$4 398 107
Bread	Pain	$1 578 743
Cereal	Céréale	$1 309 071
Cheese	Fromage	$7 140 616
Condiments	Condiments	$9 105 121
Dessert	Dessert	$2 208 427
Entre	Entre	$1 807 794
Frozen	Gelé	$521
Grains	Grains	$39 419
Lamb	Agneau	$71 553

What Is Multilingual Data Support?

You can configure Oracle BI Server to display field information in multiple languages.

Required Translation Tables

- Available language table
  - Provides list of available languages
- Lookup translation table
  - Contains translations

Required Translation Tables

There are two tables required for data translation.

- **Available language table:** Stores a list of languages available for querying against the data. It also defines the language for users when they log in to Oracle BI.
- **Lookup translation table:** Provides functionality similar to the metadata translation table discussed in the previous lesson. This table contains required columns and the language-value translations.

Each of these tables is discussed in more detail in subsequent slides.

Available Language Table

Defines the language for users when they log in to Oracle BI Server.

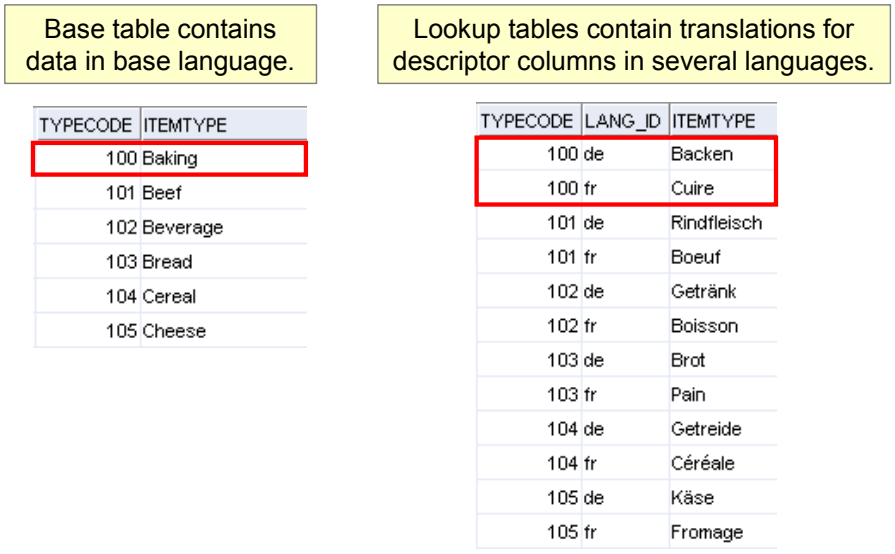
LAN_INT	LANG_ID	USER_NAME
0	en	Administrator
1	fr	Administrator_f
1	de	Administrator_d

Available Language Table

In the example in the slide, `LAN_INT` is set to `0` for English and to `1` for all other languages. Later in this lesson, you learn how to create an initialization block with a session variable that is populated by values in this table. Column names are arbitrary.

Lookup Tables

Multilingual schemas typically store translated fields in separate tables called *lookup tables*.



Lookup Tables

Lookup tables contain translations for descriptor columns in several languages, while the base tables contain the data in the base language. A lookup is when a query joins the base table and lookup table to obtain the translated values for each row in the base table. In the example in the slide, the base table data is in English. In the lookup table, there are two translations—French and German—for each product type.

Descriptor columns provide a textual description for a key column where there is a logical one-to-one relationship between the descriptor column and the key column. In this example, the descriptor column is `ITEMTYPE`, which provides textual descriptions for the `TYPECODE` key column. `LANG_ID` identifies the two-character language code for each row.

Lookup tables might be dense and sparse in nature. A dense lookup table contains translations in all languages for every record in the base table. A sparse lookup table contains translations for only some of the records in the base tables. Sometimes it is also possible for a lookup table to be both dense and sparse.

Assuming a completely dense lookup table, the number of rows in the lookup table for a particular language is equal to the number of rows in the base table.

Designing Translation Lookup Tables

There are two common techniques for designing translation lookup tables in a multilingual schema:

- Lookup table for each base table
- Lookup table for each translated field

Base table contains data in base language.

Key	Code	Description	Category_Code	Category
1	A123	Bread	D45	Groceries
2	B234	Marmalade	D45	Groceries
3	C345	Milk	D45	Groceries

Lookup table for each base table

Key	Language_Key	Description	Category
1	de	Brot	Lebensmittelgeschaft
1	it	Pane	Drogheria
2	de	Marmelade	Lebensmittelgeschaft
2	it	Marmaleta di agrumi	Drogheria
3	de	Milch	Lebensmittelgeschaft
3	it	Latte	Drogheria

Lookup table for each translated field

Field_Key	Value_Key	Language_Key	Translation
Description	A123	de	Brot
Description	A123	it	Pane
Description	B234	de	Marmelade
Description	B234	it	Marmaleta di agrumi
Description	C345	de	Milch
Description	C345	it	Latte
Category	D45	de	Lebensmittelgeschaft
Category	D45	it	Drogheria

Designing Translation Lookup Tables

There is often a separate lookup table for each base table. The lookup table contains a foreign key reference to records in the base table and contains the values for each translated field in a separate column. Assuming a completely dense lookup table, the number of rows in the lookup table for a particular language is equal to the number of rows in the base table.

The alternative approach to having one lookup table for each base table involves a separate lookup table for each translated field. Getting the translated value of each field requires a separate join to a lookup table. In practice, there is often just one physical table that contains translations for multiple fields. When a single table contains translations for multiple fields, you must place a filter on the lookup table to restrict the data to only those values that are relevant to a particular column in the base table.

ABC Example

Translate ABC product data from English to French.

Type	Type (Translated)	Dollars
Baking	Cuire	$4 925 521
Beef	Boeuf	$4 916 016
Beverage	Boisson	$4 398 107
Bread	Pain	$1 578 743
Cereal	Céréale	$1 309 071
Cheese	Fromage	$7 140 616
Condiments	Condiments	$9 105 121
Dessert	Dessert	$2 208 427
Entre	Entre	$1 807 794
Frozen	Gelé	$521
Grains	Grains	$39 419
Lamb	Agneau	$71 553
Non-food	Non-nourriture	$4 547 002
Pasta	Pâtes	$147 409
Pork	Porc	$3 431 672
Poultry	Volaille	$7 202 342
Rice	Riz	$271 889
Seafood	Nourriture de mer	$2 736 436
Snacks	Snacks	$1 482 841
Soup	Soupe	$826 005
Vegetable	Légume	$4 985 949

ABC Example

In the example in this lesson, you translate product data from English to French.

Steps for Localizing Data

1. Create a physical lookup table.
2. Create an available language table.
3. Import tables to the Physical layer.
4. Create a session variable initialization block.
5. Create a logical lookup table.
6. Set keys for the logical lookup table.
7. Create a logical lookup column.
8. Add the logical lookup column to the Presentation layer.
9. Test your work.

Steps for Localizing Data

This slide lists the steps for localizing data. Each step is described in detail in the following slides.

1. Create a Physical Lookup Table

Create a lookup table with translated fields in your physical data source.

Product Type base table

TYPECODE	ITEMTYPE
100	Baking
101	Beef
102	Beverage
103	Bread
104	Cereal
105	Cheese

Separate lookup table for Product Type base table

TYPECODE	LANG_ID	ITEMTYPE
100	de	Backen
100	fr	Cuire
101	de	Rindfleisch
101	fr	Boeuf
102	de	Getränk
102	fr	Boisson
103	de	Brot
103	fr	Pain
104	de	Getreide
104	fr	Céréale
105	de	Käse
105	fr	Fromage

ORACLE

1. Create a Physical Lookup Table

Create a lookup table with translated fields in your physical data source. You can use either of the two techniques described earlier in this lesson:

- Separate lookup table for each base table
- Separate lookup table for each translated field

The example in the slide shows a separate lookup table for the Product Type base table. The lookup table contains a foreign key reference (`TYPECODE`) to records in the Product Type base table, and it contains the values for each translated field in a separate column (`ITEMTYPE`).

`LANG_ID` identifies the two-character language code for each row. Assuming a completely dense lookup table, the number of rows in the lookup table for a particular language is equal to the number of rows in the base table. In this example, there are rows for German (`de`) and French (`fr`) for each product type.

2. Create an Available Language Table

In your physical data source, create an available language table that defines the language for users when they log in to Oracle BI Server.

LAN_INT	LANG_ID	USER_NAME
0	en	Administrator
1	fr	Administrator_f
1	de	Administrator_d

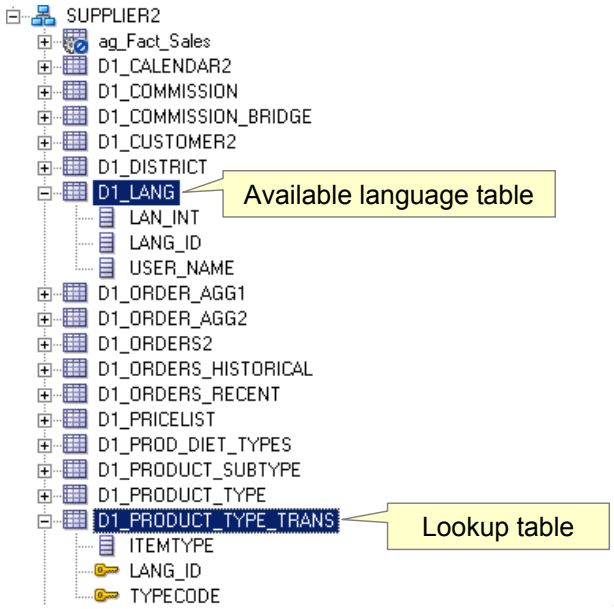
2. Create an Available language Table

In the example in the slide, `LAN_INT` is set to `0` for English and to `1` for all other languages. A separate user is defined for each language.

After you import this table into the Physical layer of the repository, you create an initialization block with a session variable that is populated by values in the table.

3. Import Tables into the Physical Layer

Import the lookup table and available language table into the Physical layer of the repository.

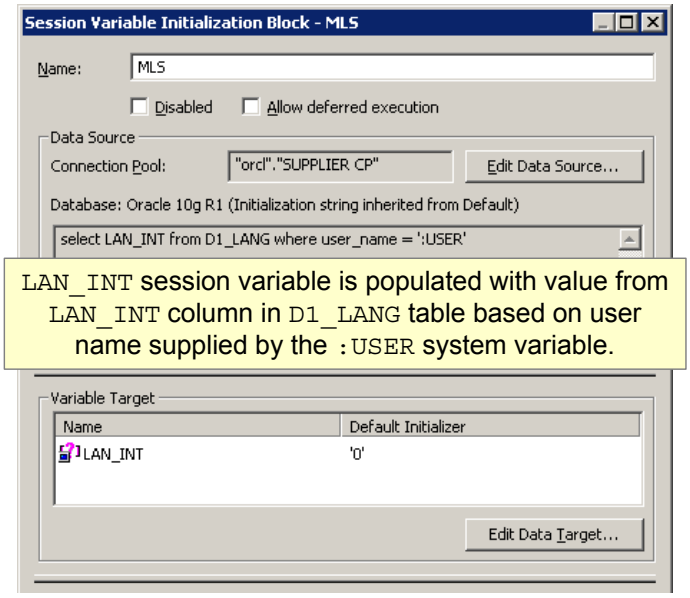


3. Import Tables into the Physical Layer

Use known techniques to import the lookup table and available language tables into the Physical layer of the repository.

4. Create a Session Variable Initialization Block

Create an initialization block that populates a variable with a language code based on user login.



4. Create a Session Variable Initialization Block

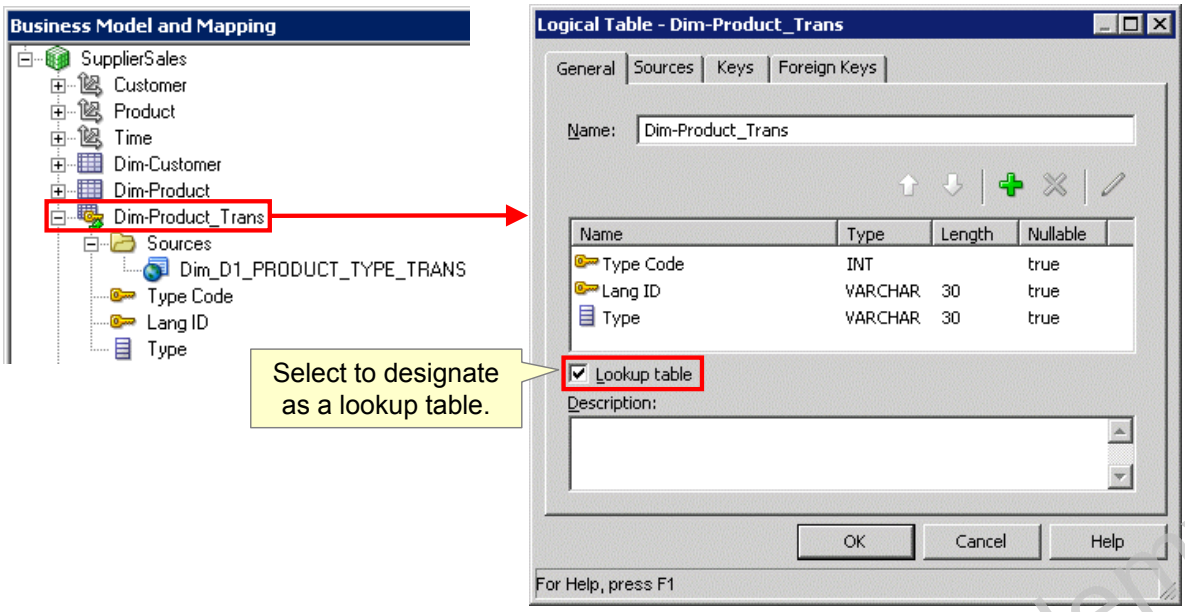
Use known techniques to create an initialization block that populates a session variable with a language code base on the user who logs in.

In this example, the MLS initialization block populates the `LAN_INT` session variable with the value from the `LAN_INT` field in the `D1_LANG` table based on the user name supplied by the `:USER` system variable.

For example, if `administrator_f` logs in to Oracle BI, the `LAN_INT` session variable is populated with the value `fr`. You use this value later when you build a lookup expression for a logical column.

5. Create a Logical Lookup Table

Create a logical lookup table object in the business model to define the necessary metadata for a translation lookup table.



5. Create a Logical Lookup Table

A lookup table is a logical table with a property that designates it as a lookup table. A logical table must be designated as a lookup table (using the Administration Tool) before you can use it as a lookup table. To designate a logical table as a lookup table, select the “Lookup table” option in the Logical Table dialog box.

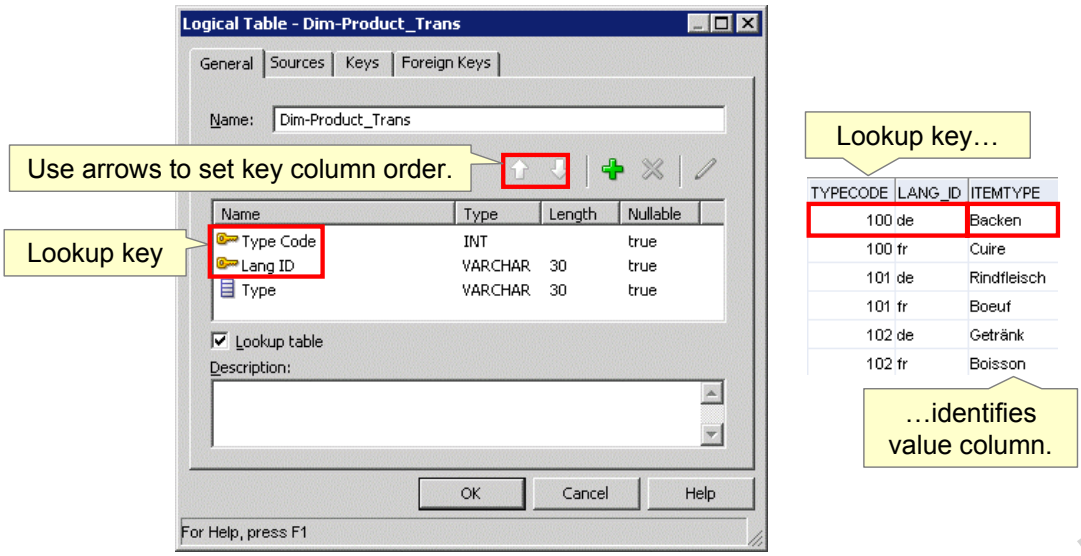
In this example, `Dim-Product_Trans` is the logical lookup table in the `SupplierSales` business model. Its logical table source is an alias of the `D1_PRODUCT_TYPE_TRANS` table.

A lookup table has at least one value column. In this example, the value column is `Type`, and it contains the translated value for the product type.

A lookup table is a stand-alone without joining to any other logical tables. Consistency checking rules are relaxed for lookup tables. As a result, if a table is designated as a lookup table, it need not be joined with any other table in the subject area (logical tables would normally be joined to at least one table in the subject area).

6. Set Keys for the Logical Lookup Table

Specify the key column order to match the order of corresponding arguments in the `LOOKUP` function.



6. Set Keys for the Logical Lookup Table

The order in which the columns are specified in the lookup table primary key determines the order of the corresponding arguments in the `LOOKUP` function. After you set the keys on the Keys tab, you can use the up and down arrows on the General tab to set the order. The `LOOKUP` function is discussed in more detail on the next slide.

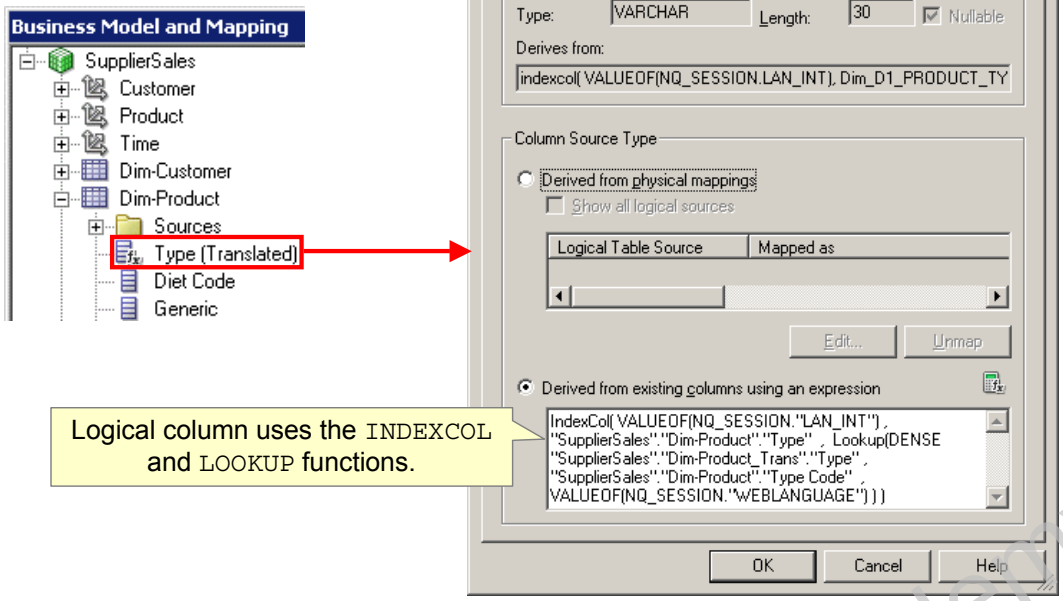
Each of the lookup table's primary keys are considered together as a lookup key and perform the lookup. The lookup can be performed only when the values for all lookup key columns are provided. In this example, the combined `Type Code` and `Lang ID` together form the primary key of this lookup table.

A lookup table has only one lookup key. A lookup key is different from a logical table key because lookup key columns are order sensitive. For example, `Type Code` and `Lang ID` are considered a different lookup key to `Lang ID` and `Type Code`.

There must be a functional dependency from a lookup key to each value column. In other words, the lookup key can be used to identify the value column. The lookup key and value column should both belong to the same physical table. In this example, the lookup key `Type Code` and `Lang ID` identify the `Type` value column.

7. Create a Logical Lookup Column

Create a logical column that includes the LOOKUP function.



Logical column uses the INDEXCOL and LOOKUP functions.

7. Create a Logical Lookup Column

You use the Expression Builder in the Administration Tool to create a new logical column that includes the lookup function. The value of the logical column depends on the language associated with the current user. You create a new logical column by using a derived column expression in the Column Source tab. In this example, the expression retrieves the translated product type.

The meaning of the expression is as follows:

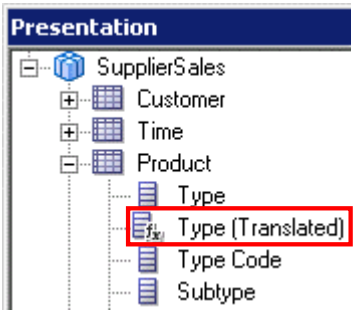
The `INDEXCOL` function helps to select the proper column. In this example, the expression returns the value of the base column (`Dim-Product.Type`) only if the user language is the base language , that is, when the value of the `LAN_INT` session variable is `0`. If the user language is not the base language, that is, when the value of the session variable `LAN_INT` is `1`, the expression returns the value from the lookup table of the language that is passed in the `WEBLANGUAGE` session variable. In our example, the possible values are `de` and `fr`.

The lookup returns the translated value of `Type` from the logical translation table, `Dim-Product_Trans`, with the following conditions for the `Dim-Product_Trans` key columns:

```
Type_Code = SupplierSales.Dim-Product.Type Code and Lang ID =
VALUEOF(NQ_SESSION."WEBLANGUAGE")
```

Note: The expressions must reference the lookup key columns in the same order in which they are defined in the lookup table. In this example, the order is `Type Code`, `Lang_ID`.

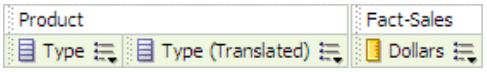
8. Add the Logical Lookup Column to the Presentation Layer



9. Test Your Work

Log in as local user.

Create an analysis using the logical lookup column.



Verify that results are returned with the expected translation.

Type	Type (Translated)	Dollars
Baking	Cuire	$4 925 521
Beef	Boeuf	$4 916 016
Beverage	Boisson	$4 398 107
Bread	Pain	$1 578 743
Cereal	Céréale	$1 309 071
Cheese	Fromage	$7 140 616
Condiments	Condiments	$9 105 121
Dessert	Dessert	$2 208 427
Entre	Entre	$1 807 794
Frozen	Gelé	$521

Summary

In this lesson, you should have learned how to localize Oracle BI data to support multilingual environments.

Practice 16-1 Overview: Localizing Oracle BI Data

In this practice, you localize product type data from English to French.

Practice 16-1 Overview: Localizing Oracle BI Data

To localize product type data from English to French, you import the necessary localization tables, create an initialization block and session variable to establish a user-preferred language, create a lookup table in the business model, build an expression for the column being translated, and test results by using an Oracle BI analysis.

17

Setting an Implicit Fact Column

Objectives

After completing this lesson, you should be able to:

- Describe the purpose and process of setting an implicit fact column
- Set an implicit fact column for a subject area

Business Challenge: Dimension-Only Queries

Dimension-only queries with columns from more than one dimension may not return the desired results.

- In a business model with conforming dimensions, many fact tables may join to the same dimensions.
- For dimension-only queries across multiple dimensions, Oracle BI Server picks the most economical fact table source based on the number of joined dimensions.

Business Challenge: Dimension-Only Queries

In this context, dimension-only queries refer to queries that contain columns from more than one dimension with no fact columns included. A dimension-only query with columns from the same dimension do not cause a problem.

There may be occasions when users want to build queries with only dimension data. For example, a user might want to see all products purchased by a customer. However, dimension-only queries may not return the desired results. This is because in a business model with conforming dimensions, many fact tables may join to the same dimensions. For example, a sales fact and a service fact both join to the product dimension.

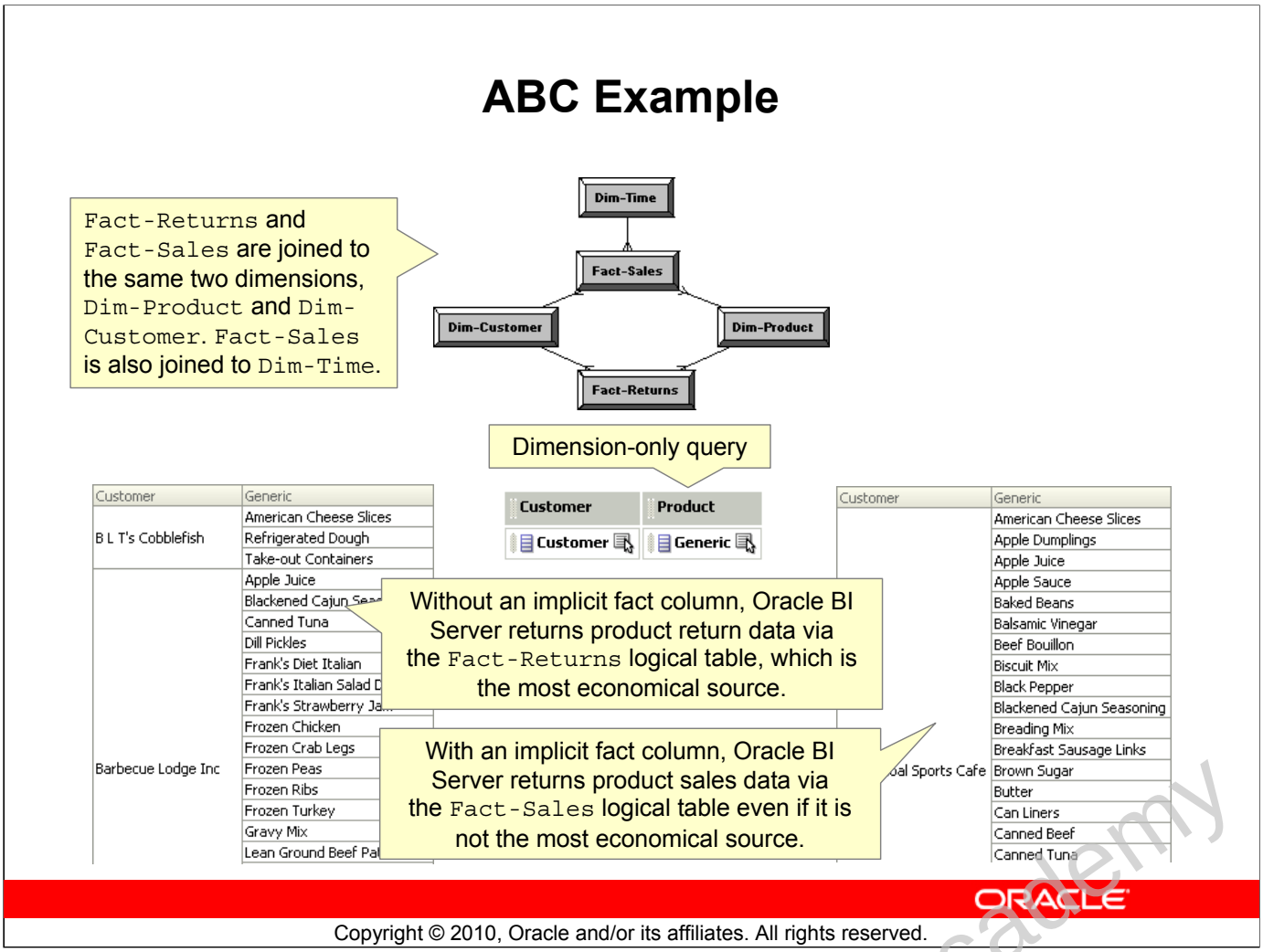
When a user runs a dimension-only query, Oracle BI Server picks the most economical fact source based on the number of joined dimensions. This may not return the desired results.

Business Solution: Implicit Fact Column

- Is a column that is added automatically to dimension-only queries
  - The column is included in the query but not shown in the results.
- Provides the ability to set a fact table source for a subject area to ensure the expected results for dimension-only queries
- Forces Oracle BI Server to select a predetermined fact table source even if it is not the most economical source
- Specifies a default join path between dimension tables when there are several possible alternatives

Business Solution: Implicit Fact Column

The solution to getting the expected results with dimension-only queries is setting an implicit fact column for subject areas. If you 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 measure column is included in the query and is visible in the query log, but it is not visible in the results returned to the user. It is used to specify a default join path between dimension tables when there are several possible alternatives.



ABC Example

In this example, there are two fact tables, `Fact-Sales` and `Fact-Returns`, which are joined to the same two dimensions, `Dim-Product` and `Dim-Customer`. `Fact-Sales` is also joined to `Dim-Time`. Both fact tables and all three dimensions are in the same subject area.

If a user runs a dimension-only query against Customers and Products without setting an implicit fact column, Oracle BI Server returns product return data via the `Fact-Returns` logical table, which has the least amount of joins and is, therefore, the most economical source. Therefore, the query results show only those customers and products for which products have been returned. This is "incorrect" data if the user expected to see all products for all customers.

With an implicit fact column set to a measure in the `Fact-Sales` table, Oracle BI Server returns product sales data via the `Fact-Sales` logical table even if it is not the most economical source. Therefore, the query results show all products purchased by all customers.

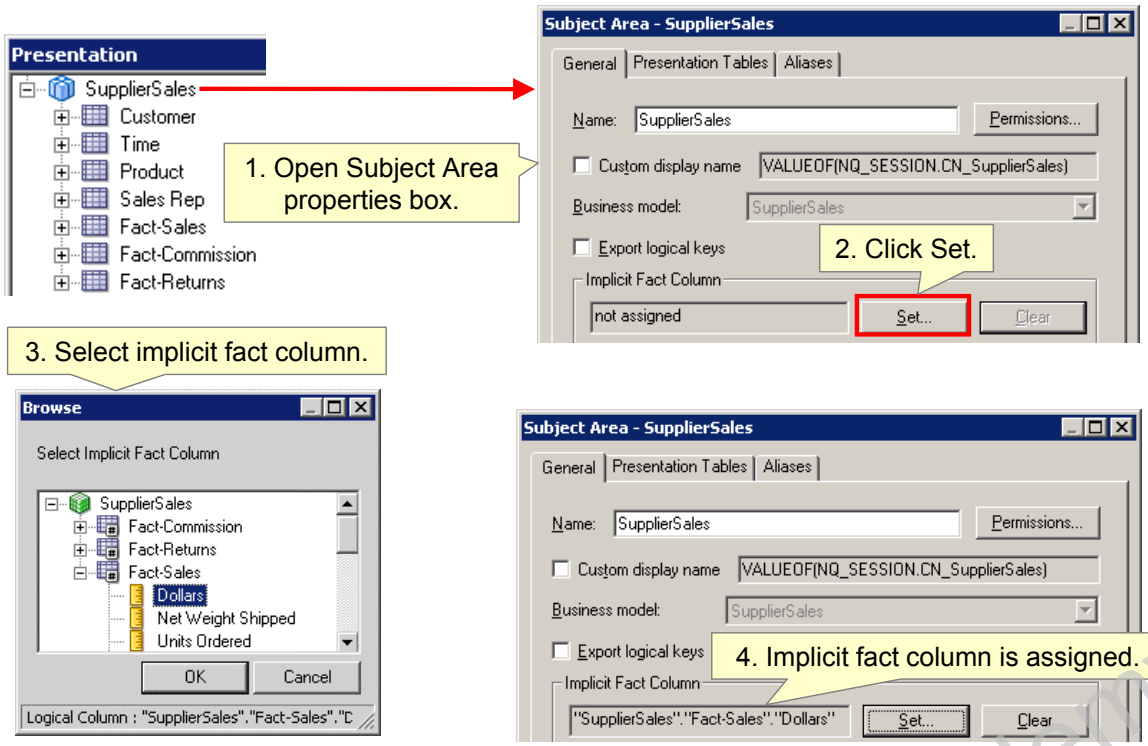
Steps to Configure an Implicit Fact Column

1. Set an implicit fact column.
2. Verify the results.
3. Clear the implicit fact column.

Steps to Configure an Implicit Fact Column

This slide lists the steps to configure an implicit fact column. Each step is presented in detail in subsequent slides.

1. Set an Implicit Fact Column



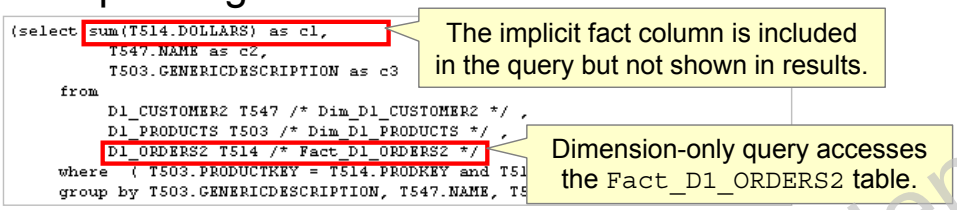
1. Set an Implicit Fact Column

To set an implicit fact column, open the Subject Area properties box. In the Implicit Fact Column section, click the Set button to open the Browse dialog box. Expand the desired fact table and select the implicit fact column. In this example, the implicit fact column is the `Dollars` column in the `Fact-Sales` logical table. After it is selected, the implicit fact column is visible in the Implicit Fact Column section.

2. Verify the Results

- Run a dimension-only analysis and verify that the correct results are returned.

- Check the log file and verify that the implicit fact column and corresponding fact table are accessed.



ORACLE

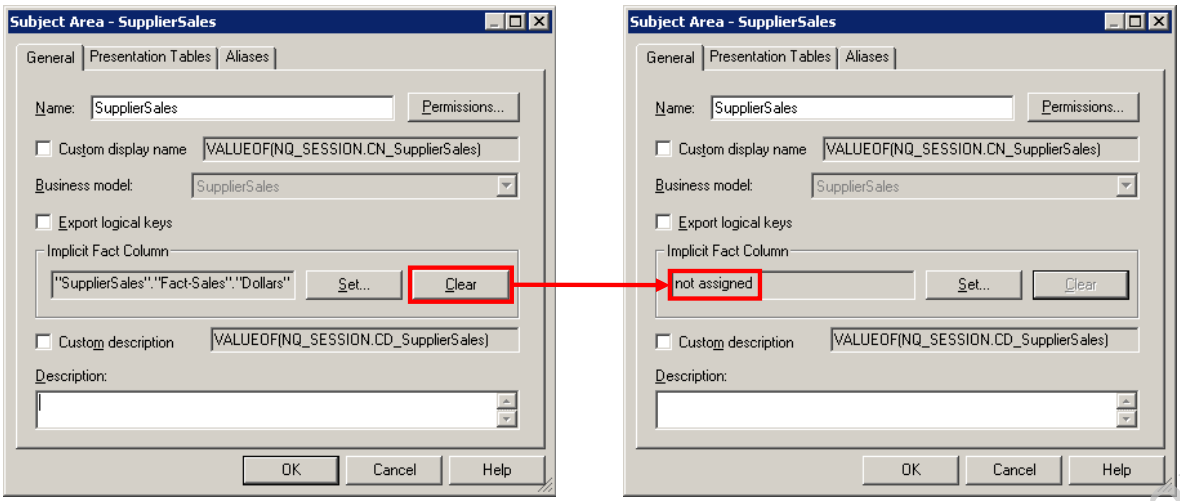
2. Verify the Results

To verify results, run a dimension-only analysis. Check the log file and verify that the implicit fact column is included in the SQL even though it does not appear in the analysis results.

In this example, the dimension-only query uses the `Customer` and `Product` dimensions. The query returns data from the expected dimensions, `Customer` and `Product`, but the `Fact_D1_ORDERS2` table is accessed instead of the `Fact_D1_RETURNS` table. Setting the implicit fact column forced Oracle BI Server to join the dimensions through this fact table. The implicit fact column, `Dollars`, is not visible in the results but is visible in the query log. The implicit fact column is used to specify a default join path between dimension tables when there are several possible alternatives or contexts.

3. Clear the Implicit Fact Column

To remove the implicit fact column, click the Clear button in the Subject Area properties box.



3. Clear the Implicit Fact Column

To clear the implicit fact column, open the Subject Area properties box, click the Clear button in the Implicit Fact Column section, and verify that the implicit fact column is set to “not assigned.”

Summary

In this lesson, you should have learned how to:

- Describe the purpose and process of configuring an implicit fact column
- Set an implicit fact column for a subject area

Practice 17-1 Overview: Setting an Implicit Fact Column

In this practice, you set an implicit fact column for a subject area.

Practice 17-1 Overview: Setting an Implicit Fact Column

ABC tracks both product sales and product return information. ABC wants to ensure that dimension-only queries return the correct results. To ensure the expected results, you test different implicit fact column settings for the `SupplierSales` subject area.

18

Importing Metadata from Multidimensional Data Sources

Objective

After completing this lesson, you should be able to create a repository by using a multidimensional data source.

Overview

- You can use the Administration Tool to add a multidimensional data source to the Physical layer of a repository.
- Multidimensional data sources that Oracle BI Server can connect to include:
  - Essbase
  - Hyperion Financial Management
  - Microsoft Analysis Services
  - SAP/Business Warehouse (SAP/BW)
- Data from multidimensional sources can be used in analyses and Oracle BI interactive dashboards.

Overview

You use the Administration Tool to add a multidimensional data source to the Physical layer of a repository. The ability to use multidimensional data sources enables Oracle BI Server to connect to sources such as Essbase, Hyperion Financial Management, Microsoft Analysis Services, and SAP/Business Warehouse (SAP/BW) to extract data. You can then use data from these sources to create analyses, which can be displayed on an Oracle BI interactive dashboard.

Multidimensional Versus Relational Data Sources

- The primary differences between setting up multidimensional data sources and setting up relational data sources are in the Physical layer.
- The setup processes in the business model and presentation layers for multidimensional data sources and relational data sources are almost identical.

Overview: Importing Multidimensional Data Sources

- Each cube imported from a multidimensional data source is created as a single physical cube table in the Physical layer.
- Oracle BI Server imports cubes with corresponding metrics, hierarchies, and levels.
- After importing cubes, make sure that:
  - Physical cube columns have the correct aggregation rule
  - The default member type `ALL` is correctly imported for a hierarchy
- Remove hierarchies and columns from the Physical layer if they are not used in the business model.

Overview: Importing Multidimensional Data Sources

During the import process, each cube in a multidimensional data source is created as a single physical cube table.

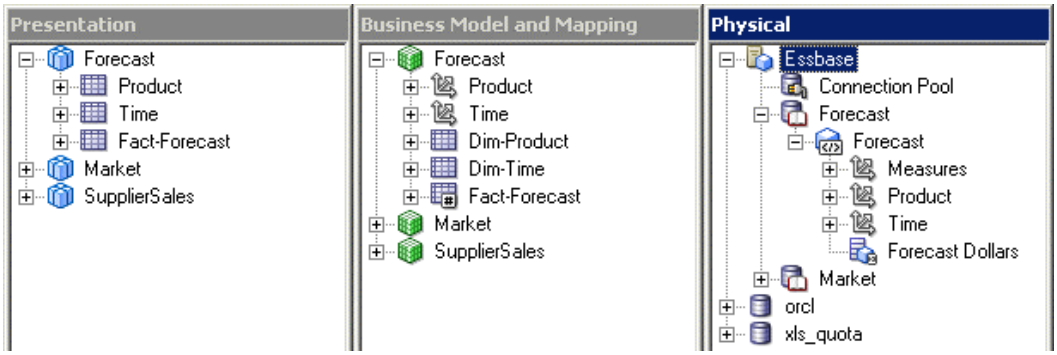
Oracle BI Server imports the cube, including its metrics, dimensions, and hierarchies.

After importing the cubes, you need to make sure that the physical cube columns have the correct aggregation rule and that the default member type `ALL` is correctly imported for a hierarchy.

It is recommended that you remove hierarchies and columns from the Physical layer if they are not used in the business model. This eliminates maintaining unnecessary objects in the Administration Tool and might result in better performance.

ABC Example

Create a business model by using an Essbase multidimensional data source.



ABC Example

In the example, you create a business model in an Oracle BI repository by using an Essbase multidimensional data source.

Creating a Multidimensional Business Model

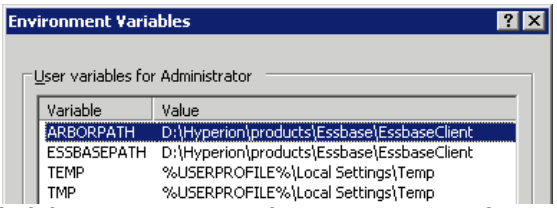
1. Set up an Essbase data source.
2. Import metadata.
3. Verify the import.
4. Choose options to control the model.
5. View members and update member counts.
6. Create the business model.
7. Create the Presentation layer.
8. Verify the results.

Creating a Multidimensional Business Model

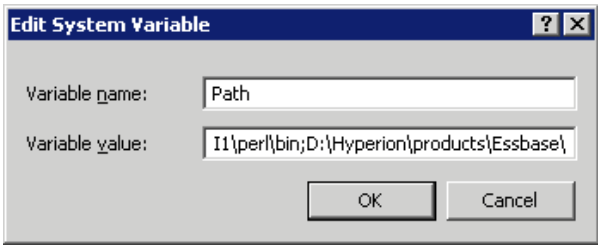
This slide lists the steps for creating a business model by using an Essbase multidimensional data source. Each step is presented in detail in the following slides.

1. Set Up an Essbase Data Source

- Install the Essbase client libraries on the computer that is running Oracle BI Server.
- In a Windows environment:
  a. Set the `ARBORPATH` environment variable.



  b. Add an entry to the `PATH` environment variable.



Set Up an Essbase Data Source

Oracle BI Server uses the Essbase client libraries to connect to Essbase data sources. You must install the Essbase client libraries on the computer that is running Oracle BI Server before you can import metadata from Essbase data sources. You also must ensure that the Essbase client libraries are installed on any computer where you want to run the Administration Tool.

After you verify that the Essbase client libraries are installed on the appropriate computers, you must ensure that the `PATH` environment variable on each computer includes the location of the Essbase client driver.

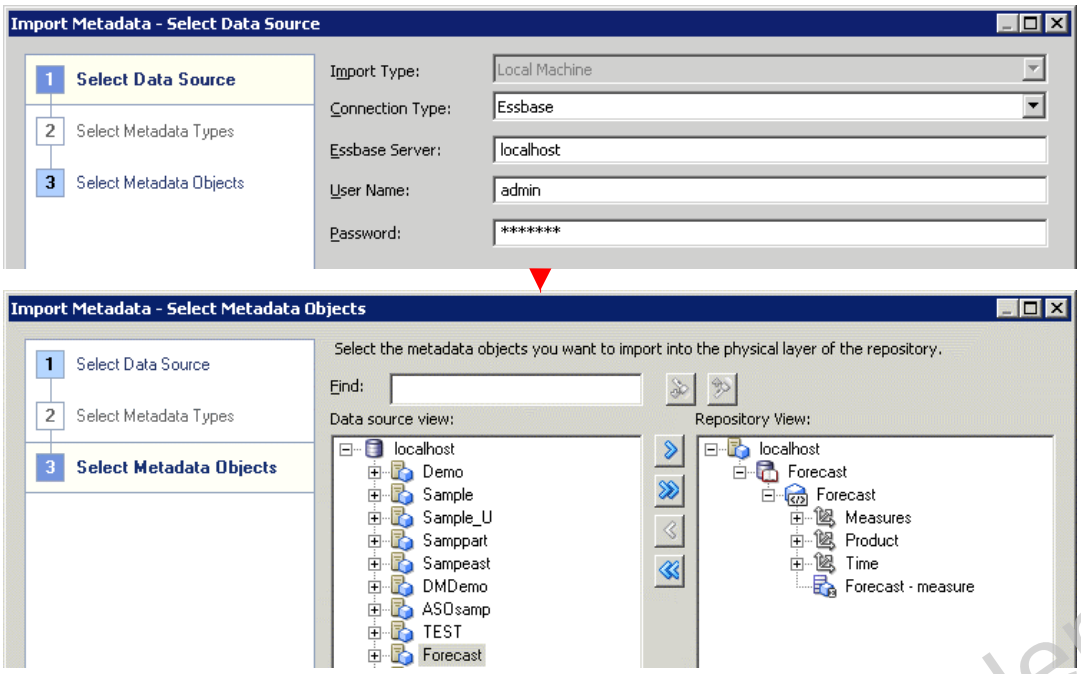
Example: `EPM_ORACLE_HOME/products/Essbase/EssbaseClient`)

In addition, you also need to ensure that an additional environment variable is set appropriately for each computer (either `ESSBASEPATH` or `ARBORPATH`, depending on your client version). For more information, see the *Oracle Hyperion Enterprise Performance Management System Installation and Configuration Guide* (or the equivalent title for your client version).

For information about setting up Essbase client libraries on other platforms, see “Configuring Essbase Data Sources on Linux” in the *Metadata Repository Builder's Guide for Oracle Business Intelligence Enterprise Edition*.

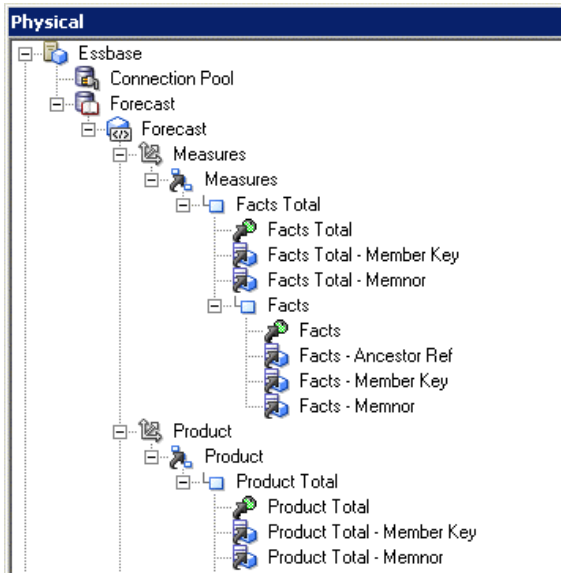
2. Import Metadata

Use known techniques to import from an Essbase source.



3. Verify the Import

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.



ORACLE

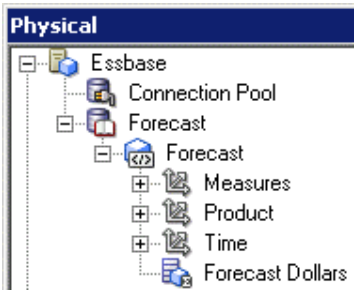
3. Verify the Import

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. Column types such as Outline Sort, Ancestor Reference, Member Key, Leaf, Root, and Parent Reference are used internally by the system and should not be changed.

4. Choose Options to Control the Model

Use different options in the Physical layer to control how you want to model certain types of metadata.



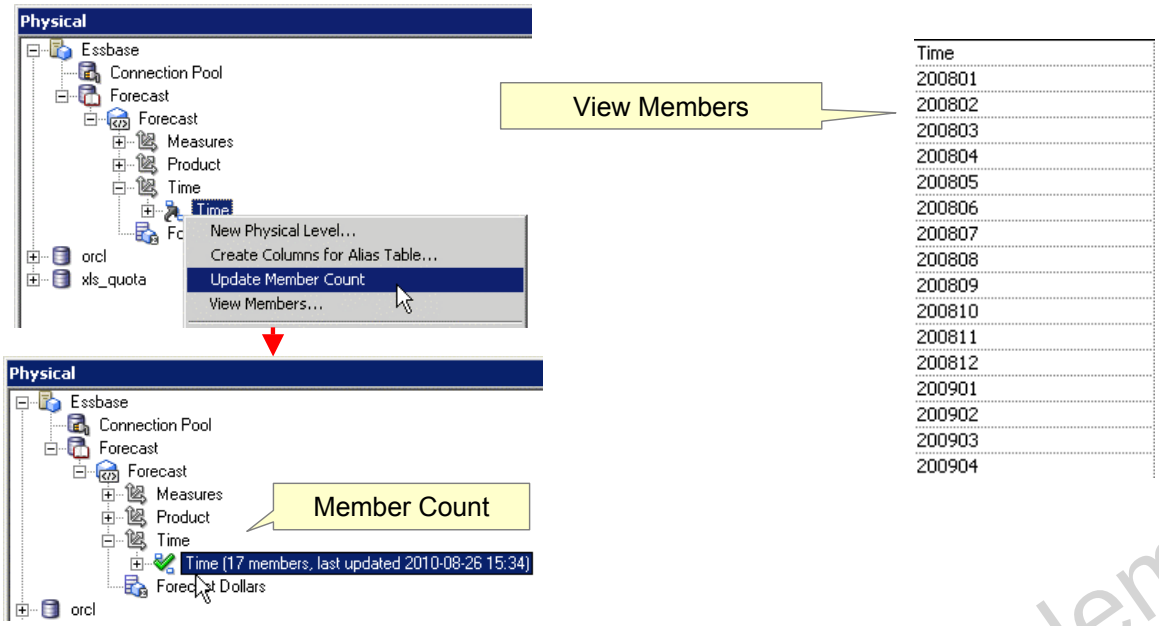
4. Choose Options to Control the Model

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 your user base. For example, many types of Essbase metadata are modeled as dimension properties by default in the Physical layer. This multidimensional structure works best with the new hierarchical reporting style introduced in the current release.

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 screenshot shows the result of flattening the measure hierarchy. 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."

5. View Members and Update Member Count

To update member counts, the repository must be open in online mode.



5. View Members and Update Member Count

You can view members and update member count to verify connectivity. You must open the repository in online mode to update member count. You can view members in offline or online mode.

To determine if counts need to be updated, place your cursor over the hierarchy or level name. A message informs you that the counts need to be updated or shows you when they were last updated.

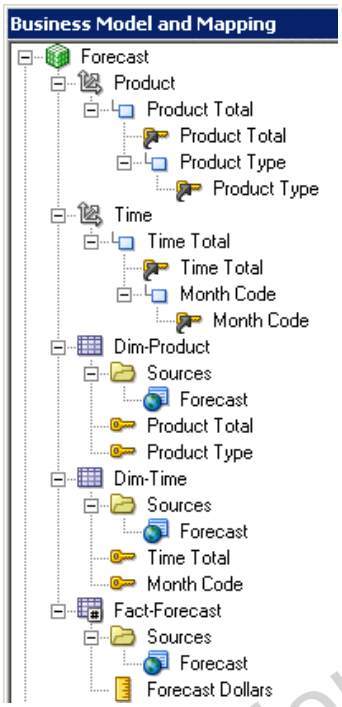
When you update member counts, the current number of members is returned from the selected hierarchy. After the member count is updated successfully, it appears in a message when you place the cursor over the hierarchy or level name. The message appears in the following syntax:

```
<hierarchy name> (<x> members, last updated <time stamp>)
```

6. Create the Business Model

Setting up a business model for multidimensional data sources is similar to setting up a business model for a relational data source.

Drag the cube to the BMM layer.

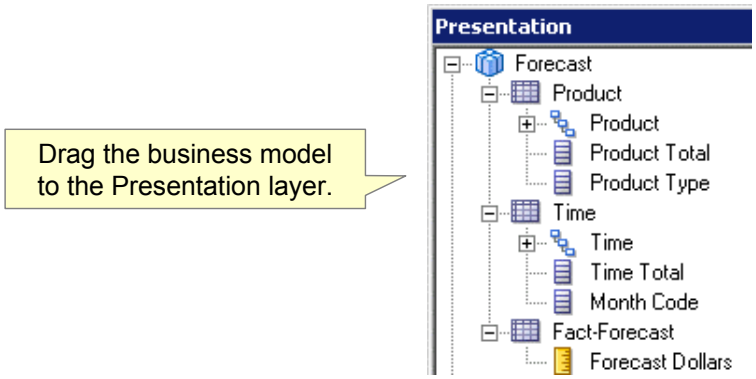


6. Create the Business Model

To create the business model, you drag the physical layer cube to the logical layer. When you drag from the Physical layer, logical tables, dimensions, and relationships are created automatically. You can then modify objects in the BMM layer just as you would with a relational data source.

7. Create the Presentation Layer

Setting up the Presentation layer for multidimensional data sources is similar to setting up the Presentation layer for a relational data source.

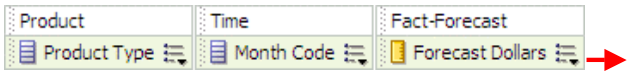


7. Create the Presentation Layer

To create the Presentation layer, you drag the business model to the Presentation layer. You can then modify objects in the Presentation layer just as you would with a relational data source.

8. Verify the Results

- Build and execute a query.



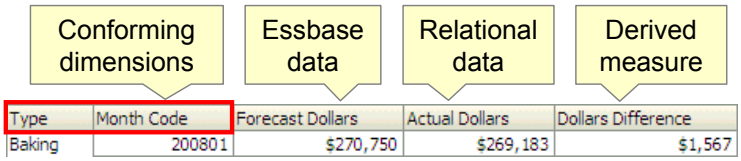
Type	Month Code	Forecast Dollars
Baking	200801	$269,183
	200802	$287,437
	200803	$312,651
	200804	$293,180
	200805	$331,326
	200806	$316,510
	200807	$317,913
	200808	$314,553
	200809	$290,870
	200810	$359,976

- Verify SQL in the query log.

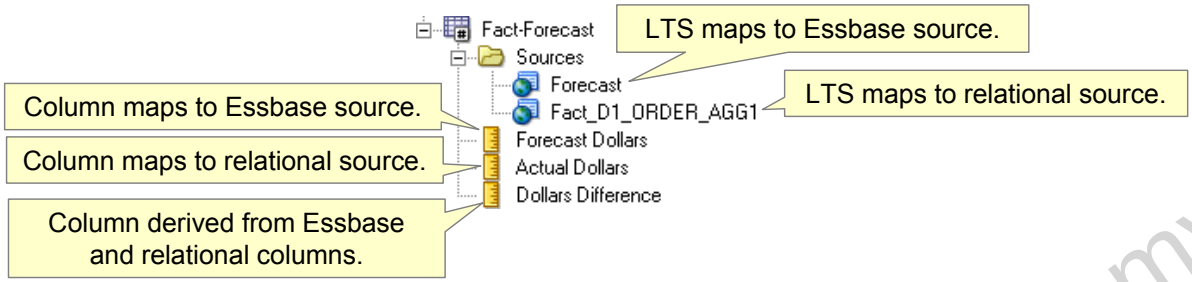
```
-------------------- Sending query to database named Essbase (id: <<4654>>)
With
  set [Product2]  as '[Product].Generations(2).members'
  set [Time2]  as '[Time].Generations(2).members'
select
  { [Measures].[Forecast Dollars]
  } on columns,
  NON EMPTY {crossjoin ({[Product2]},{[Time2]})} properties ANCESTOR_NAMES,
from [Forecast.Forecast]
```

Horizontal Federation

- Provides the ability to create reports that can display data from both multidimensional and relational data sources



- Requires modification in the BMM layer



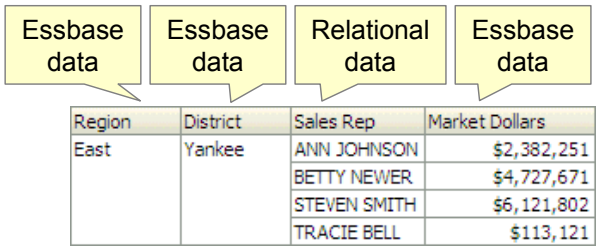
Horizontal Federation

In this example, data for the Forecast Dollars measure is coming from an Essbase multidimensional source, data for the Actual Dollars measure is coming from a relational source, and Dollars Difference is a derived measure calculated across the Essbase and relational measures. Data from the conforming dimensions—`Product` and `Time`—is applied across the measures.

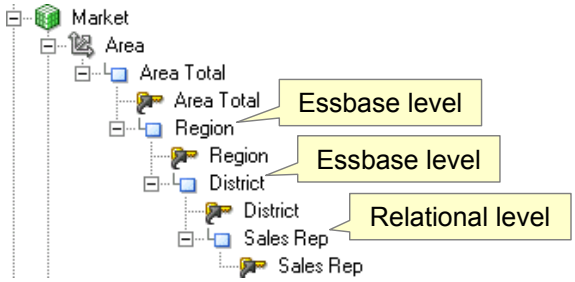
Horizontal federation requires modification of objects in the BMM layer. In this example, you must add new logical table sources and columns to the `Fact-Forecast` logical table, the `Product` logical table, and the `Time` logical table to establish a relationship between the Essbase source and the relational source in the business model. You also create derived measures. The example in the slide shows only the modifications to the `Fact-Forecast` logical table.

Vertical Federation

- Provides reports with the ability to drill through aggregate multidimensional data into detail relational data



- Requires modification of hierarchies in the BMM layer



Vertical Federation

Because you cannot create physical joins between multidimensional and relational sources in the repository, the relational source is unable to inherit the Area hierarchy from the Essbase source. Therefore, you must physically define the hierarchy to allow for drill down.

In this example, you must have `Region`, `District`, and `Sales Rep` columns in your relational source so that you are able to drill all the way through `Region` and `District` in the Essbase cube to `Sales Rep` in the relational source.

Summary

In this lesson, you should have learned how to create a repository by using a multidimensional data source.

Practice 18-1: Importing a Multidimensional Data Source into a Repository

This practice covers the following topics:

- Importing an Essbase cube into the Physical layer
- Building a business model
- Verifying the results

Practice 18-2: Incorporating Horizontal Federation into a Business Model

This practice covers building a business model that incorporates horizontal federation of Essbase and relational data sources.

Practice 18-2: Incorporating Horizontal Federation into a Business Model

Horizontal federation provides the ability to create reports that can display data from both Essbase and relational data sources. In the previous practice, you imported an Essbase cube and viewed query results. In this practice, you add objects from the Supplier Sales relational data source to the Essbase business model so that you can build a report that compares forecast dollars with actual dollars.

Practice 18-3: Incorporating Vertical Federation into a Business Model

This practice covers building a business model that incorporates vertical federation of Essbase and relational data sources.

Practice 18-3: Incorporating Vertical Federation into a Business Model

Vertical federation provides reports with the ability to drill through aggregate Essbase data into detail relational data. In this practice, you expand the business model to include vertical federation of Essbase and relational data sources.

Practice 18-4: Adding Essbase Measures to a Relational Model

This practice covers adding an Essbase measure to an existing fact table with a relational source

Practice 18-4: Adding Essbase Measures to a Relational Model

You add a forecast measure from an Essbase cube to the `Fact-Sales` table in the `SupplierSales` business model.