Oracle Data Modeling and Relational Database Design

Volume II • Student Guide

D56497GC10
Edition 1.0
September 2010
D67008

Author

Marcie Young

Technical Contributors and Reviewers

Sue Harper
Philip Stoyanov
Nancy Greenberg
Rick Green
Brian Pottle
Anjula Subbiahpillai
Gerry Jurrens
Nick Donatone
David Lapoint
Tom Provenzano
Mike Ritz
Tim Trauernicht
Zhicheng Xu
Ron Berry
David Lyons
Kim Bell
Maria Billings
Steve Friedberg
Bryan Roberts
Priyanka Sharma
Matthew Gregory
Angelika Krupp

Editors

Daniel Milne
Vijayalakshmi Narasimhan

Graphic Designer

Rajiv Chandrabhanu

Publishers

Shaik Basha
Jayanthy Keshavamurthy

Contents

Transforming Your Logical Model to a Relational Design

15

Mapping Your Entity Relationship Diagram to a Relational Database Design

Objectives

After completing this lesson, you should be able to:

- Describe why a database design is needed
- Decide on naming conventions and rules
- Perform a mapping between a logical and a relational model
- Utilize the Oracle SQL Developer Data Modeler facility

Objectives

In this lesson, you learn how to use the Oracle SQL Developer Data Modeler engineering tool to engineer your logical model to a relational model, and how logical objects map to relational objects.

Why Create a Relational Model?

- A relational model:
  - Is closer to the implementation solution
  - Facilitates discussion
  - Forms the basis for the physical database design
- The ideal model can be adapted to a relational database management system (RDBMS) model.

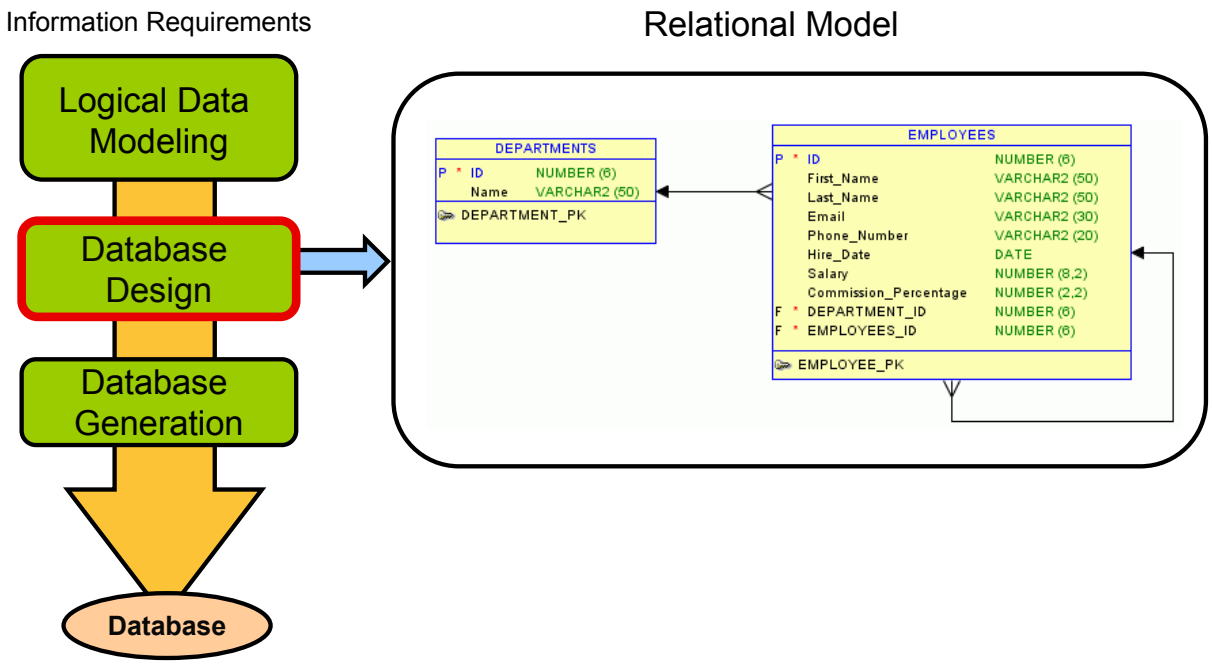
Why Create a Relational Model?

The logical data model describes the data required for the business. This model should be completely independent from any implementation considerations. This same logical data model could also be used as a basis for implementation of any type of database management system (DBMS) or even a file system.

A logical data model is a high-level representation that cannot be implemented as is. People creating these models may not be aware of physical and database constraints, but they still must provide a conceptually “workable” solution. This is why it is important to have a validated and agreed upon logical data model before going into the physical database design.

Engineering the logical data model creates a “first-cut” relational model or database design. This first-cut model is intended to serve as a new basis for defining the physical implementation of the database.

REVIEW: Database Design



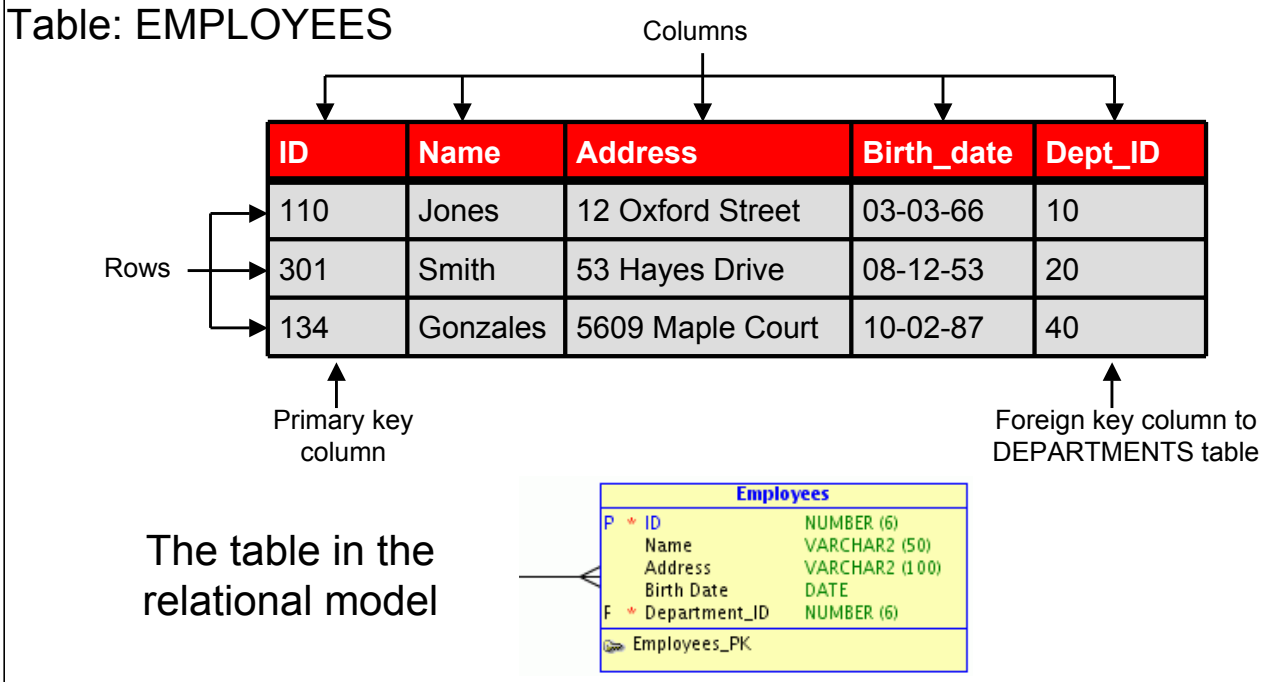
ORACLE

REVIEW: Database Design

The database design, often called the “relational model,” is the model that is built during the Design phase of the database development life cycle. The purpose of this model is to describe the database objects that must be created when the database is generated. The basic components of a database design include objects such as relational tables, columns, primary and foreign keys. The database design maps to the objects in the logical data model.

The relational model shown in the slide has two tables: DEPARTMENTS and EMPLOYEES. Each table has a primary key. The EMPLOYEES table has two foreign keys, one for MANAGER_ID and one for DEPARTMENT_ID.

Relational Database Overview



ORACLE

Relational Database Overview

Tables are supported by integrity rules that protect the data and the structures of the database. Integrity rules require each table to have a primary key and each foreign key to be consistent with its corresponding primary key.

Tables: A table is a very simple structure in which data is organized and stored. Tables have columns and rows. Each column is used to store a specific type of value. In the above example, the EMPLOYEES table is the structure used to store employees' information.

Rows: Each row describes an occurrence of an employee. In the example, each row describes in full all properties required by the system.

Columns: Each column holds information of a specific type: ID, name, address, date of birth, and the ID of the department to which the employee is assigned.

Primary Keys: A column or set of columns that uniquely identifies each row in a table. Each table must have a primary key, and a primary key must be unique. In the example in the slide, the ID column is a primary key (that is, every employee has a unique identification number) in this table, that distinguishes each individual row.

Foreign Keys: A column or combination of columns in one table that refers to a primary key in the same or another table. In the example in the slide, the Dept_ID foreign key identifies the department that an employee works in.

Terminology Mapping

ANALYSIS	DESIGN
Logical model	Relational model
Entity	Table
Attribute	Column
Primary UID	Primary key
Secondary UID	Unique constraint
Relationship	Foreign key
Business constraints	Check constraints

ORACLE

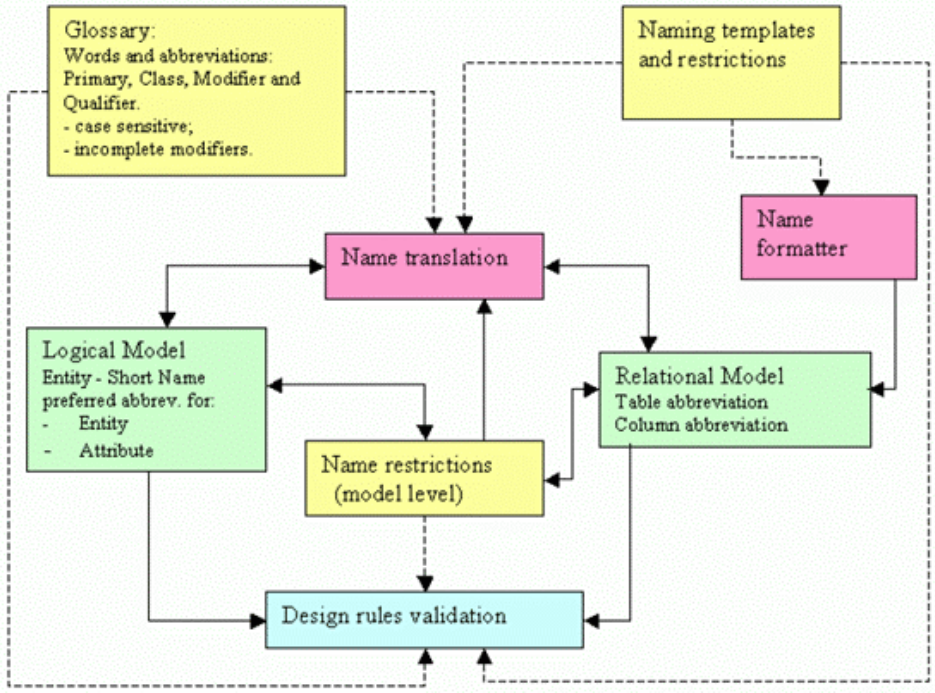
Terminology Mapping

Changing from one world to another also means changing terminology. The mappings are as follows:

- An entity creates a table.
- An attribute becomes a column in a table
- A primary unique identifier becomes a primary key for the table.
- A secondary unique identifier creates a unique constraint
- A relationship is mapped into a foreign key and foreign key columns.
- Constraints are the rules with which the database must be defined to be consistent. Some of the business rules are translated into check constraints, other complex rules require additional programming.

This initial mapping is limited to the design of tables, columns, and constraints that can be declared. A declarative constraint is a business constraint that can be ensured at the server level by using only database language statements; a declarative constraint requires no coding.

Naming Conventions



ORACLE

Naming Conventions

Before transforming the logical data model, you must define a naming convention so that project members use the same standards and produce the same model from the same source. This lesson explains naming conventions used within Oracle, however, they are not the only ones that you can use.

There are many ways to define naming standards in Oracle SQL Developer Data Modeler. In the diagram in the slide, you see there are two models: Logical and Relational. You can enforce naming standards in the following ways:

Using a Glossary: Allows you to define words and abbreviations and identify what type of word they represent (Primary, Class, Modified, and Qualifier)

Using Naming Templates: Allows you to define the way in which the name will be generated. For example, the primary key name will always have a suffix of _PK, and therefore the template would be set to `{table}_PK`.

Using Name Restrictions: Allows you to restrict naming during engineering. For example, if you want entity and attribute names to all be lowercase, you can set this restriction in the Model Properties.

Using Name Translation: Allows you to utilize the naming rules that you already set up, such as the glossary or preferred abbreviations for an object

Naming Conventions (continued)

Using Name Formatter: Allows you to invoke naming rules that utilize name abbreviations or templates

Using Design Rule Validations: Allows you to check your model to make sure objects are complete from a modeling perspective

Naming Conventions

Decide on a convention for:

- Table names
- Special characters (%, *, #, -, space, …)
- Table short names
- Column names
- Primary and unique key constraint names
- Foreign key constraint names
- Foreign key column names

Naming Conventions (continued)

Table Names: The plural of the entity name is used as the corresponding table name. The idea is that the entity is the concept of an abstract thing. For example, you can talk about EMPLOYEE, CUSTOMER, and so on, and therefore singular is a good naming rule; however, a table is made up of rows (the EMPLOYEES table, or CUSTOMERS table) where the plural is more appropriate.

Column Names: Column names are identical to the attribute names, with a few exceptions. Replace special characters with an underscore character. In particular, remove the spaces from attribute names, because SQL does not allow spaces in the names of relational elements. The Start Date attribute converts to the Start_Date column; the Delivered Y/N attribute transforms to Delivered_y_n (or preferably Delivered_indicator). Often column names use more abbreviations than attribute names.

Abbreviations: A unique abbreviation for every table is a very useful element for the naming of foreign key columns or foreign key constraints. A suggested way to make these abbreviations is based on the following rules:

For entity names, use one of the following methods:

- Identify an abbreviation that has meaning. For example, the EMPLOYEE entity might have a short name of EMP.
- Remove the vowels from the name. For example, the FLIGHT entity might have a short name of FLT.

Naming Conventions (continued)

These abbreviations construction rules do not guarantee uniqueness; however, it may minimize duplicate naming.

In Oracle SQL Developer Data Modeler, if two names happen to be the same, the tool will automatically add a number to the second object that is created.

For columns that are not null, you might consider appending NN to the name.

Foreign Key Constraints: The recommended rule for naming foreign key constraints is

`<short name of the from table>_<short name of the to table>_<fk>`

For example, a foreign key between tables EMPLOYEES and DEPARTMENTS results in the constraint name `emp_dept_fk`.

Foreign Key Columns: Foreign key columns are prefixed with the short name of the table they refer to. This leads to foreign key column names like `dept_no`. Limiting the attribute name to 22 characters enables you to add two prefixes plus two underscore to the column name.

If there are two (or more) foreign keys between two tables, the foreign keys and foreign key columns would be entitled to the same name. In this situation, you can add the name of the relationship to both foreign key names or Oracle SQL Developer Data Modeler will add a number to the end of the second foreign key. Do the same with the foreign key columns. This way you will never mistake one foreign key for the other.

Check Constraints: Check constraints are named `<table short name>_ck<sequence number>` (for example, as `emp_ck1` for the EMPLOYEES table).

Naming Restrictions with Oracle

- Table and column names:
  - Must start with a letter
  - May contain up to 30 alphanumeric characters
  - Should not contain space or some special characters
  - Should avoid reserve words
- Table names must be unique within a schema.
- Column names must be unique within a table.

ORACLE

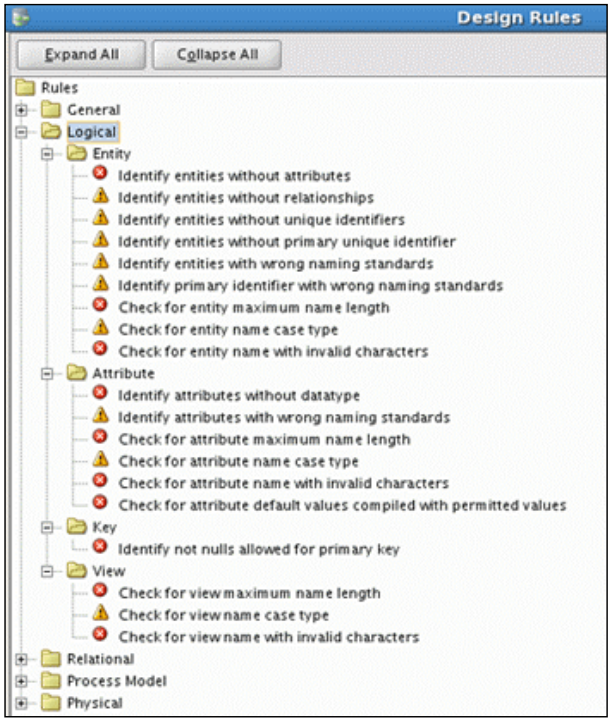
Naming Restrictions with Oracle

Each RDBMS can have its own naming restrictions. You must know whether the convention you decide to use is compatible with the RDBMS that you choose. If your RDBMS is Oracle, it has the following naming restrictions:

- You can use any alphanumeric character for naming as long as the name
  - Starts with a letter
  - Is up to 30 characters long
  - Does not include special characters such as “!”, but “$”, “#” and “_” are permitted
- Table names must be unique within the schema that is shared with views and synonyms.
- Within the same table, two columns cannot have the same name. If so, a “v#” will be appended to the end of the name.
- Make sure that you are not using reserve programming language words such as Number, Sequence, Values, Level, or Type.

Note that if a space is contained in the name, double quotes will be placed around the name when DDL is generated (discussed in a later lesson).

Ensuring That Your Logical Data Model Is Complete



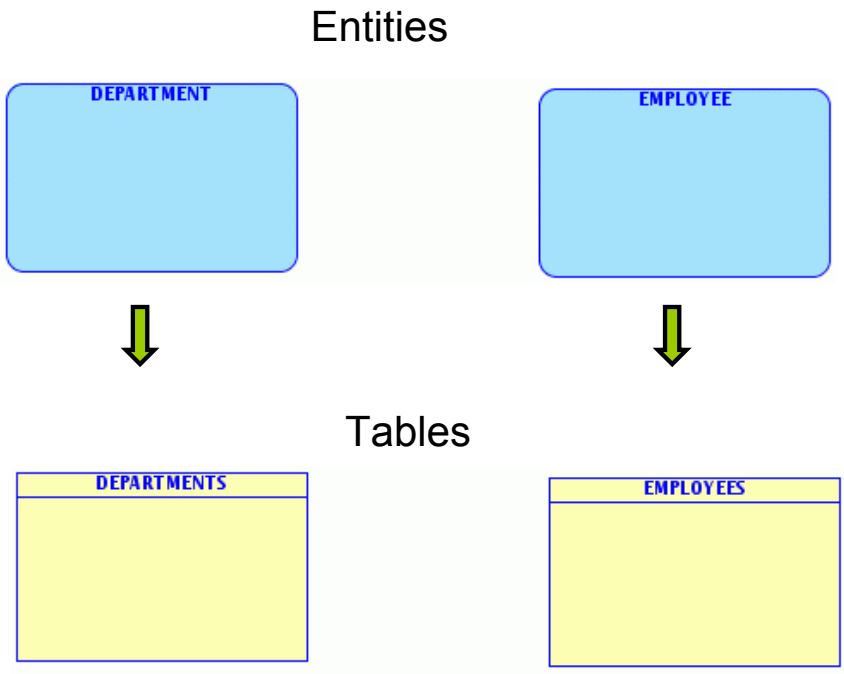
ORACLE

Ensuring That Your Logical Data Model Is Complete

Before you engineer your logical data model to a relational model, you must make sure that it is complete. You can run the design rules to verify that all the objects in your model are defined completely. The Design Rules facility will not tell you whether your model is correct from a business standpoint; however, it may identify some issues such as entity naming discrepancies.

To execute the design rules, from the Tools menu, select Design Rules. To run a particular rule, select it from the list and click Apply Selected. If you want to run all the rules, click Apply All.

Mapping Simple Entities

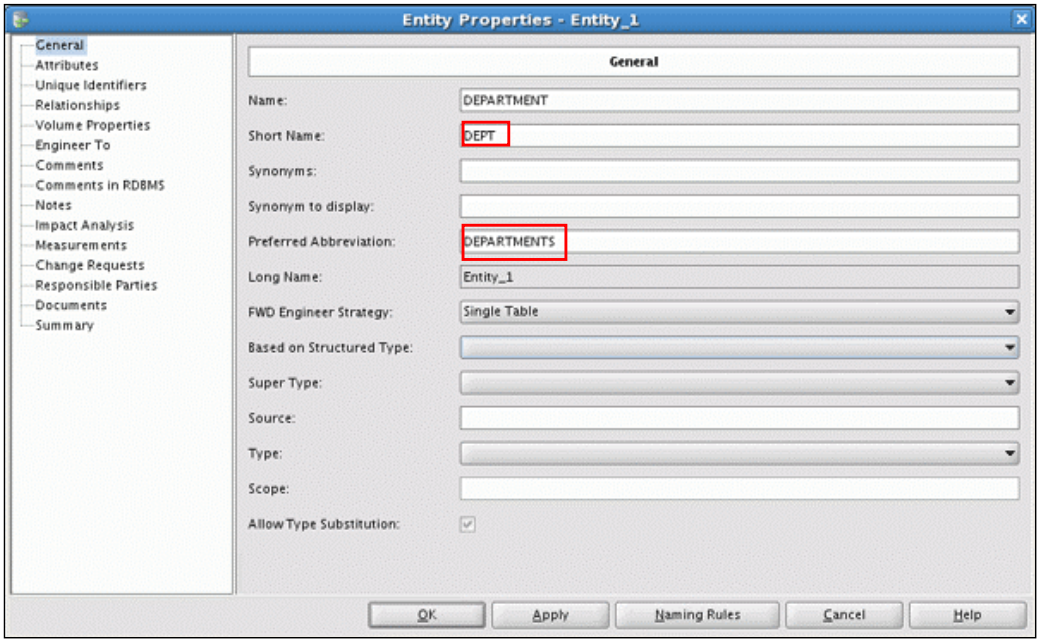


Mapping Simple Entities

Map each simple entity to a table. The table name should be easy to trace back to the entity name. The plural of the entity name is sometimes used because the table will contain a set of rows.

A simple entity is not a subtype or super type (discussed later in this lesson).

Naming Entities

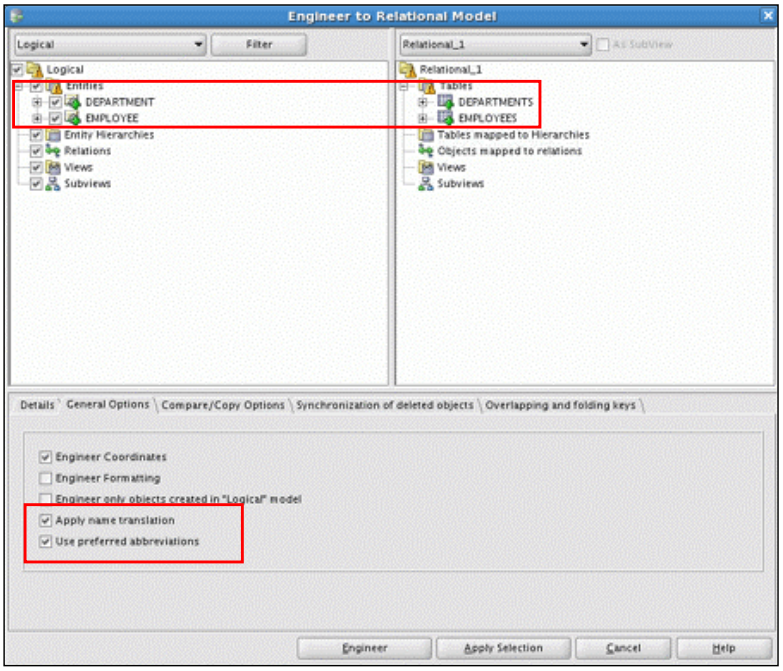


ORACLE

Naming Entities

In the Sort Name field, define a short name for an entity. The short name will be engineer as a table abbreviation. Use the Preferred Abbreviation field to define the preferred abbreviation, which can be engineered as the table name.

Engineering Entities



ORACLE

Engineering Entities

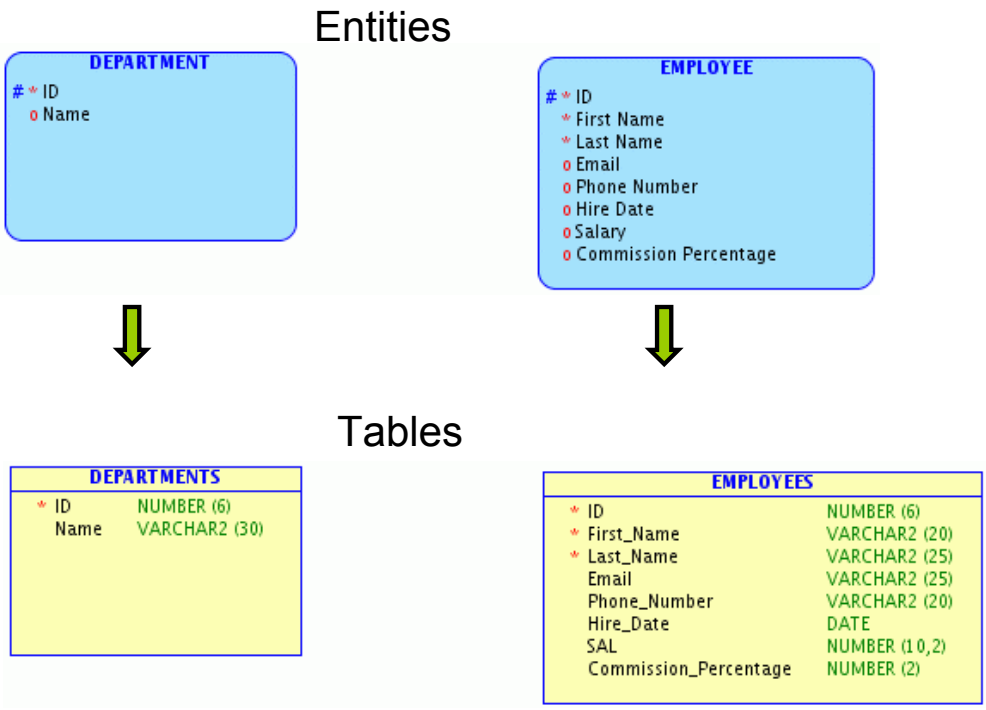
After your design rules have been verified and your naming parameters specified, you can engineer your logical model to a relational model. Select the Engineer icon in your toolbar. The Engineering window appears. To view the objects that will be engineered, expand the object type in either the Logical or Relational side of the window. Notice in this case that two tables will be created in the Relational model.

To apply the naming standards that you have set, select the General Options tab and select “Apply name translation.” Notice that the “Use preferred abbreviations” check box is then highlighted and checked by default.

Click Engineer to engineer your model.

Note: If you do not like the results that you see, click Cancel, make changes, and re-run the engineering step.

Mapping Attributes to Columns

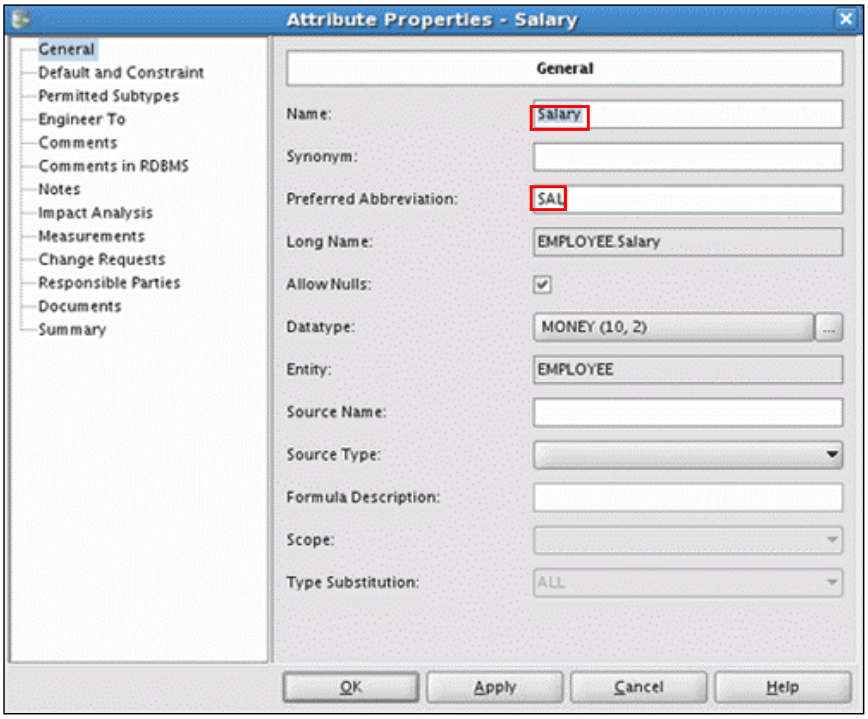


ORACLE

Mapping Attributes to Columns

Map each attribute to a column in its entity's table. Map mandatory attributes to NOT NULL (NN) columns. In the example in the slide, the attributes of the DEPARTMENT entity are mapped to columns in the DEPARTMENTS table, and the attributes in the EMPLOYEE entity are mapped to the columns in the EMPLOYEES table.

Mapping Attributes to Columns: Column Names



ORACLE

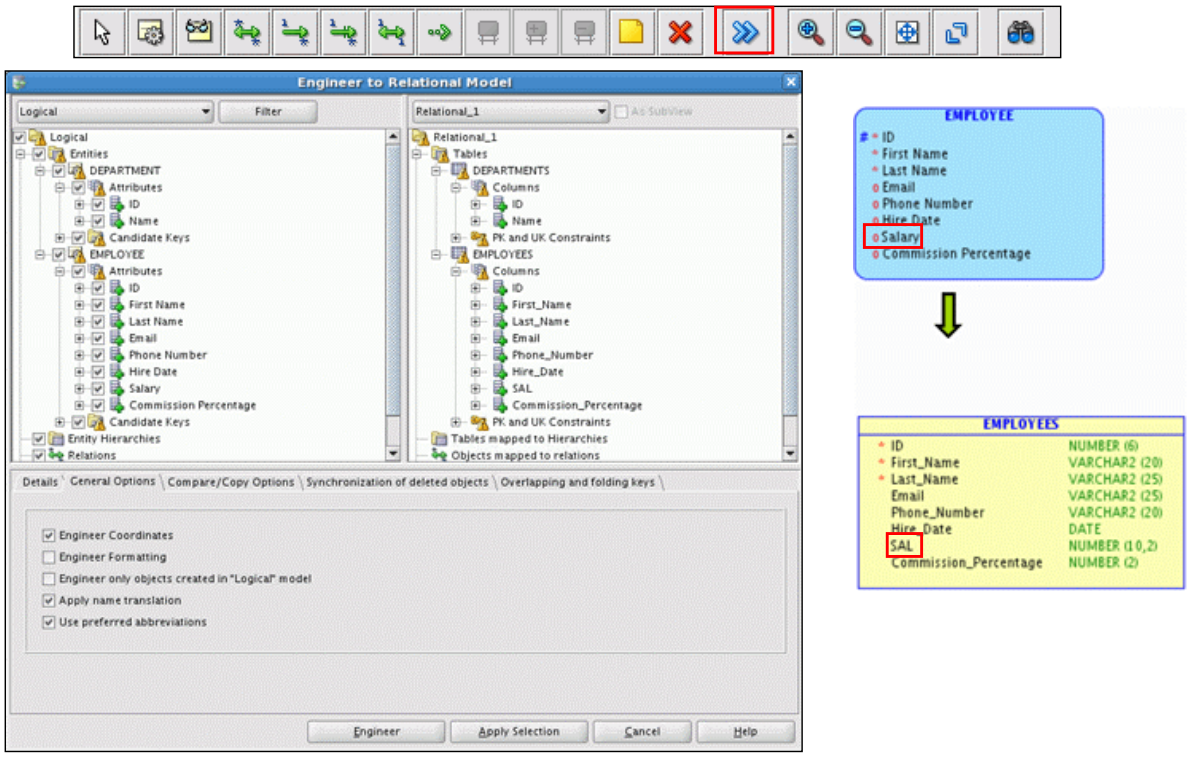
Mapping Attributes to Columns: Column Names

For each attribute, guidelines for specifying names are as follows:

- Avoid the use of SQL reserve words, such as NUMBER, for column names.
- Use consistent abbreviations to avoid programmer and user confusion. For example, will Number be abbreviated as NO or NUM. Is it DEPTNO or DEPTNUM?
- Short column names will reduce the time required for SQL command parsing.

You can also specify a preferred abbreviation that can be used when engineering attributes to columns in your relational model.

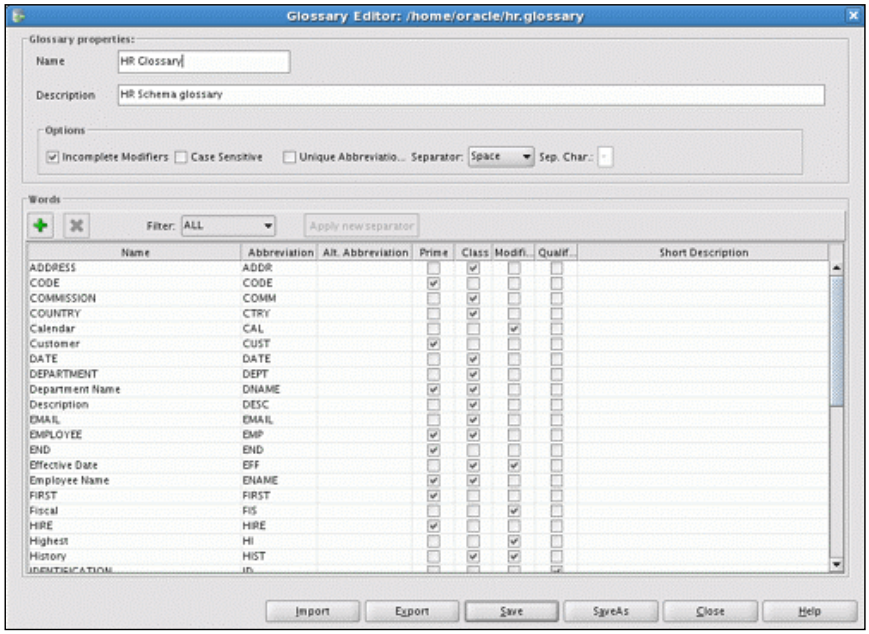
Engineering Attributes



Engineering Attributes

When engineering attributes to columns, you can view the list of attributes that will be engineered to columns from the engineering page. Notice the Add icon for each column that will be created.

Reviewing the Glossary



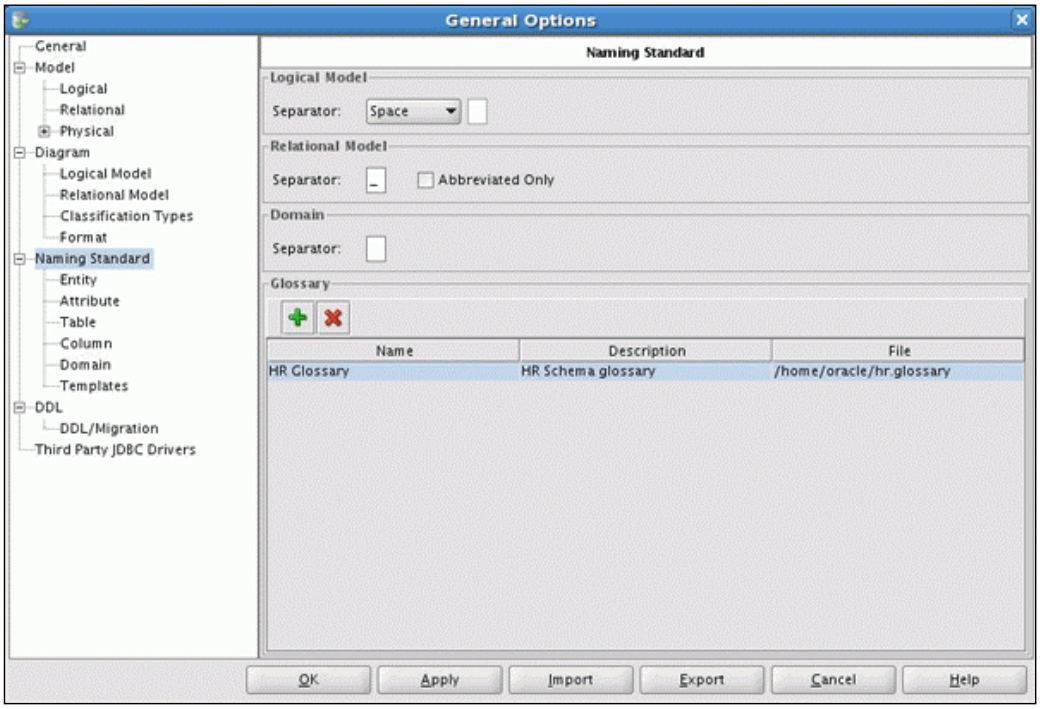
Reviewing the Glossary

The glossary is used during the engineering process to translate names between the logical and relational models.

You can define one or more glossaries as validation glossaries. If there is more than one glossary, then a name is considered to be valid if it can be validated using any one of the defined glossaries. You can use different glossaries to represent separate domains (areas) of interest. However, using many glossaries together can lead to unpredictable results, especially when abbreviations are used in the validation process. For example, AP could be “Accounts-Payable”, but also can match to “Actual Placement” if defined in another glossary.

Creating and using a glossary is a two-step process: Create and review the glossary. Define a new glossary or modify definitions from another glossary file. When defining a glossary, you specify all the terms in your model and specify an abbreviation that will be used when engineering to a relational model. In addition, you can define various options such as the separator character that will be used during engineering. To access the glossary, select Tools > Glossary Editor.

Adding the Glossary as the Naming Standard

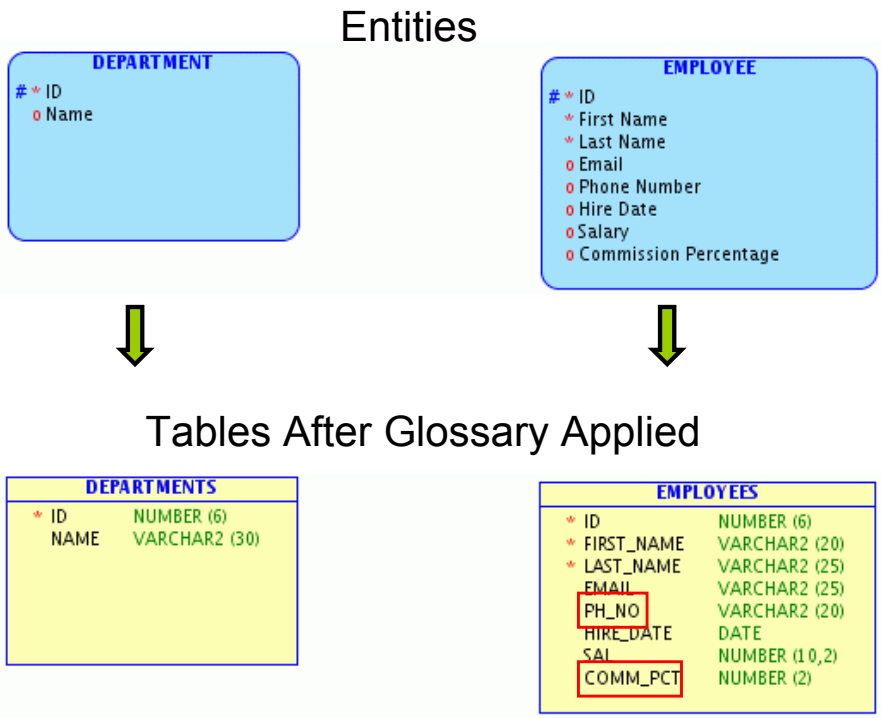


ORACLE

Adding the Glossary as the Naming Standard

In order for the glossary to be applied during engineering, you must add it on the Naming Standard page under General options.

Mapping Attributes to Columns with the Glossary

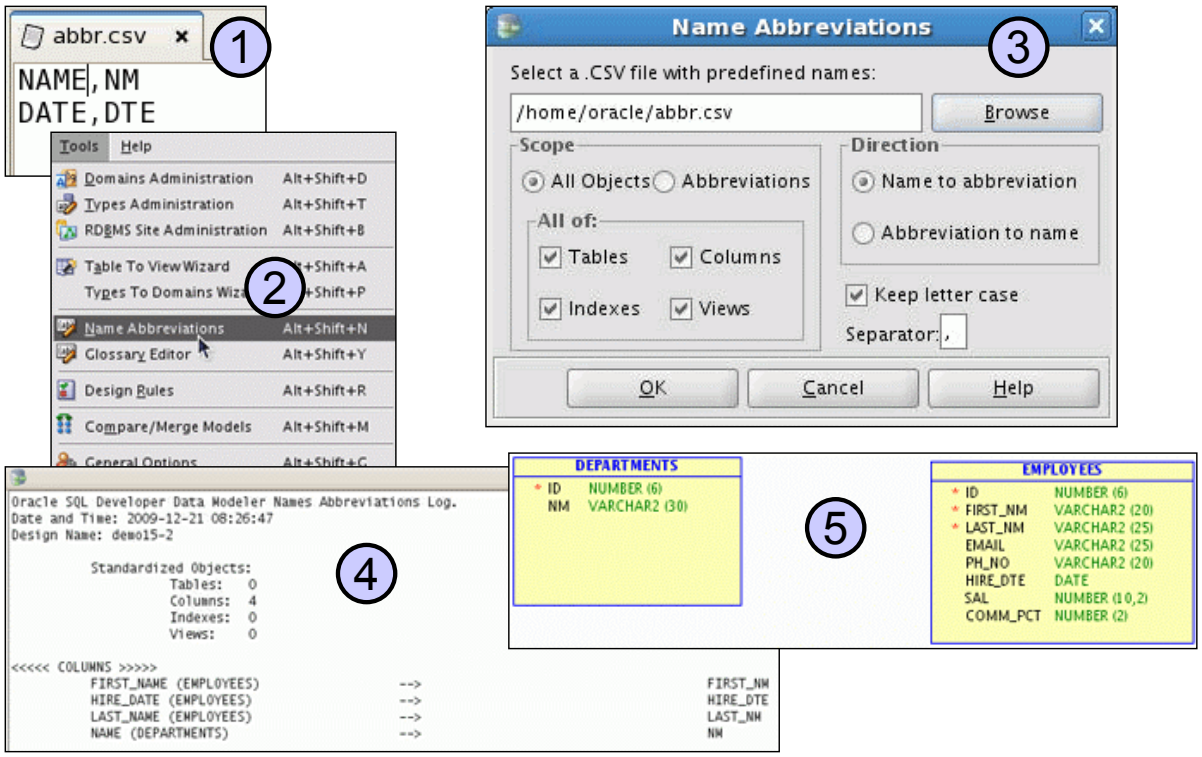


Mapping Attributes to Columns with the Glossary

After the glossary is applied, all the attribute names are separated by an underscore character ("_"), and the abbreviations' additional abbreviations have been applied. SAL remains the same because that abbreviation was defined in the attribute properties window.

Note that if you have an abbreviation in the glossary and a different one in the definition of an object, and you select Use Preferred Abbreviation when you engineer, the preferred abbreviation in the object definition will be used. For example, if SAL is specified for the preferred abbreviation in the Salary definition, and the abbreviation for Salary in the glossary is set to SALARY, SAL will be used if you have Use Preferred Abbreviation in the General Options tab when you engineer the model.

Applying Name Abbreviations

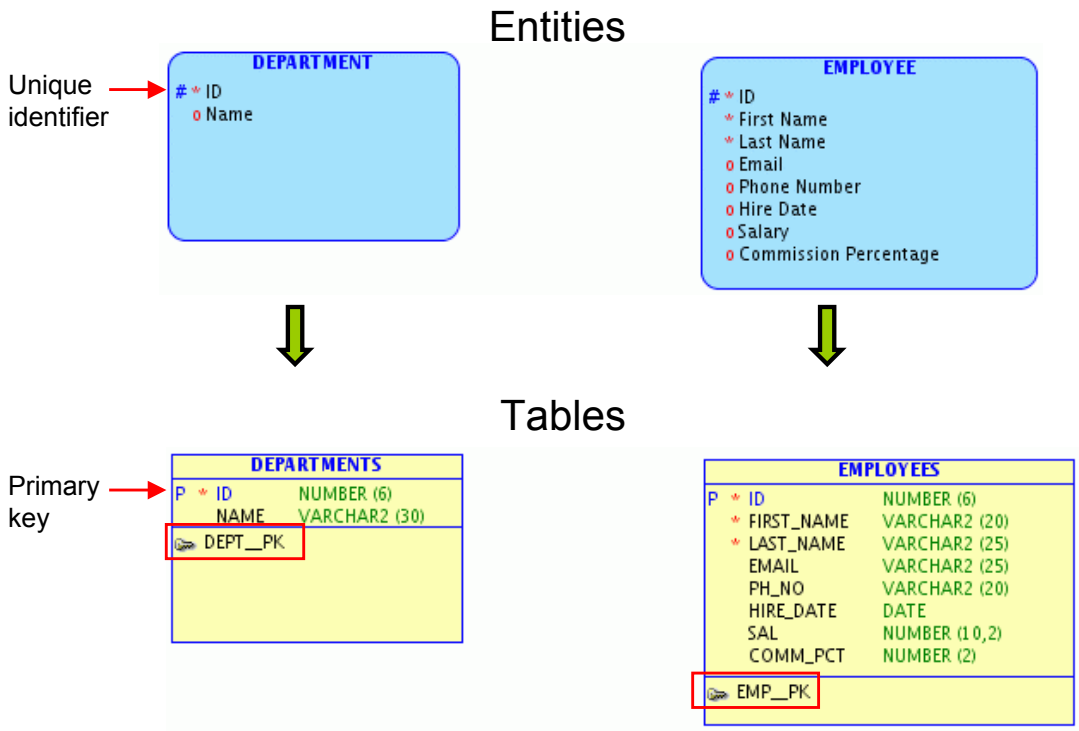


Applying Name Abbreviations

If after you engineer your model, you want to abbreviate additional relational objects, you can apply the abbreviations directly to the relational model by performing the following steps:

1. Create a .csv file with the abbreviations for a set of names that you want to translate. In the example in the slide, you want to change NAME to NM and DATE to DTE. Note that these names are case-sensitive.
2. Select Tools > Name Abbreviations.
3. In the Name Abbreviations dialog box, select the .csv file that you created in step 1, select which objects to apply the abbreviations to or whether you want to apply to change the abbreviation to the name instead, and click OK.
4. A log window appears with the results. In this case, the name was change to the abbreviation. Review, and then click Close.
5. The results are displayed. Notice that in the DEPARTMENTS table, the NAME column was changed to NM.

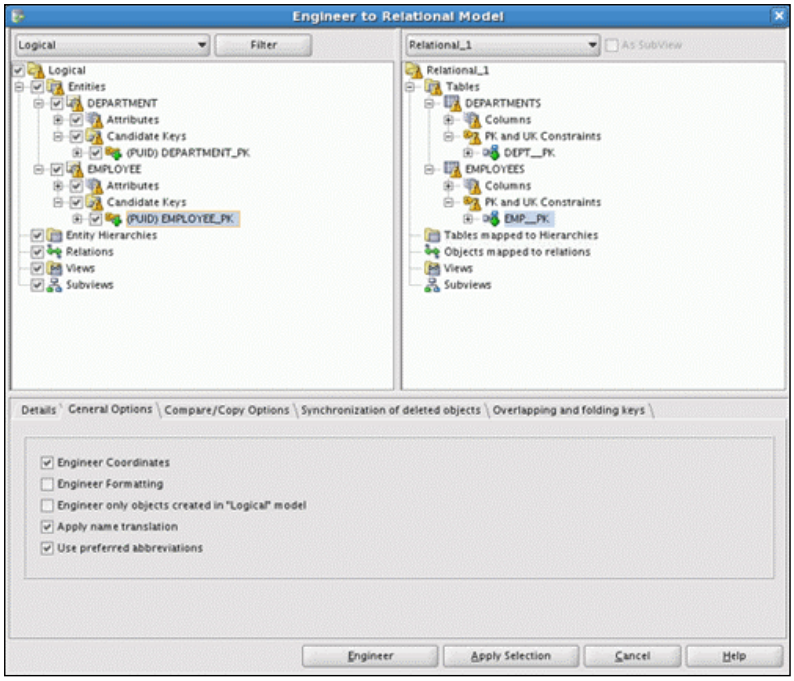
Mapping Unique Identifiers to Primary Keys



Mapping Unique Identifiers to Primary Keys

Attributes that are part of the entity's unique identifier are mapped to primary key columns in the relational model. All columns labeled as the primary key must also be mandatory and must not allow nulls. In the example in the slide, the unique identifier for the DEPARTMENT entity was ID. After engineering, the ID column in the DEPARTMENTS table is the primary key column contained in the DEPT_PK primary key.

Engineering Unique Identifiers

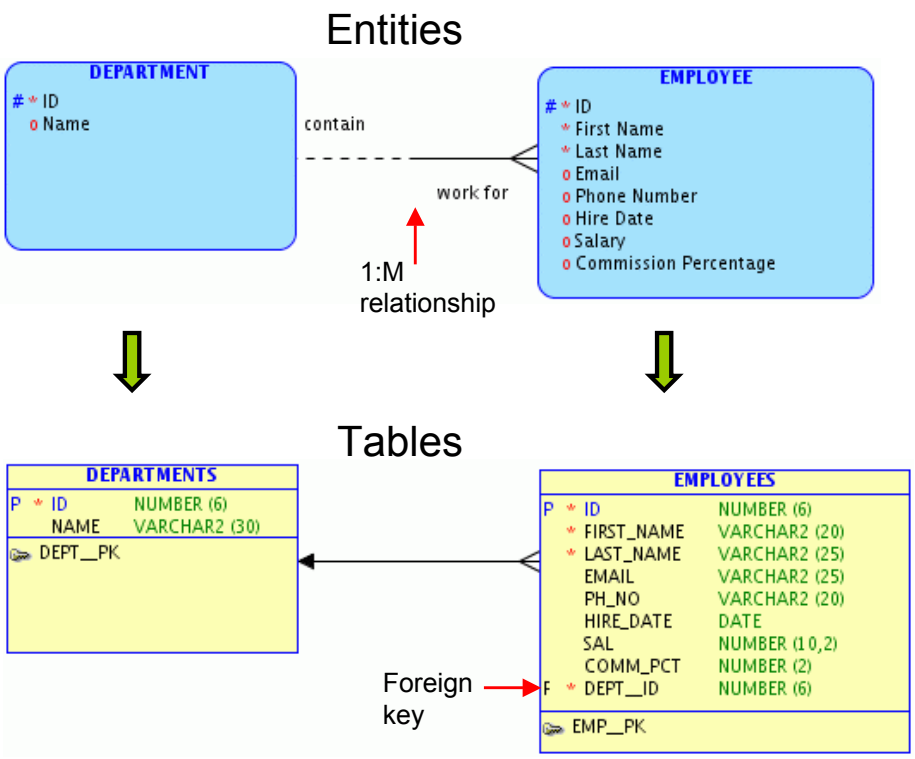


ORACLE

Engineering Unique Identifiers

When engineering unique identifiers, expand Entities > <*the entity*> > Candidate Keys to see the list of keys that will be engineered.

Mapping Relationships to Foreign Keys



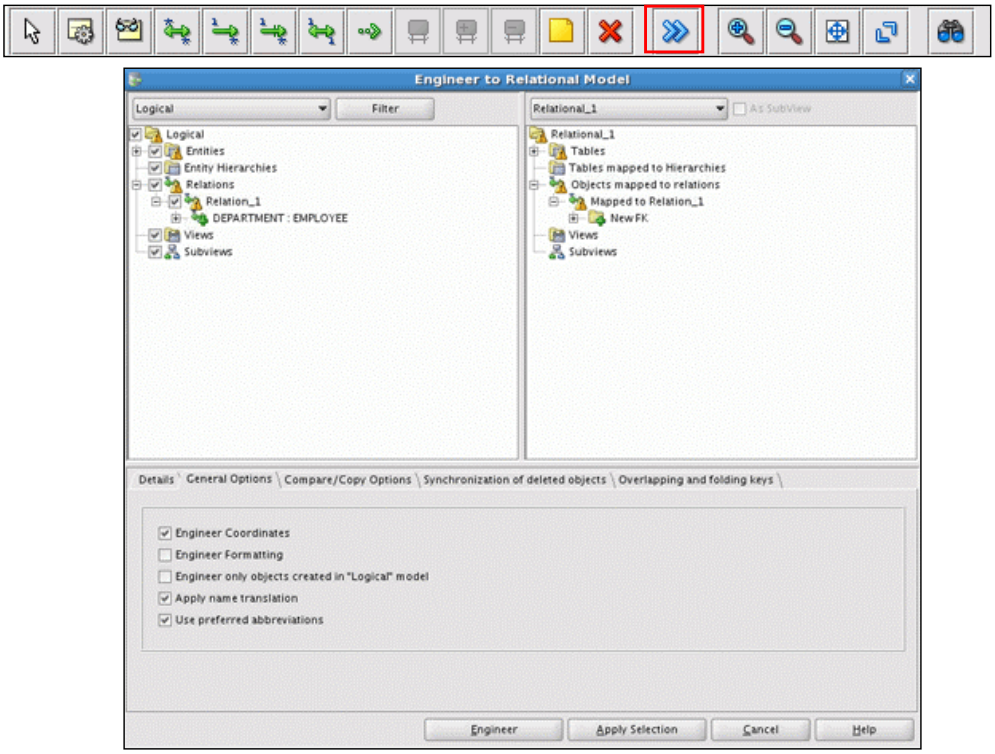
Mapping Relationships to Foreign Keys

For 1:M relationships, after the model is engineered, the primary key column of the originating table becomes a foreign key column in the many side of the table. In the example in the slide, the primary key for DEPARTMENTS (ID) becomes the foreign key column in the EMPLOYEES table. Notice that the short name of the table is concatenated with the name of the column. This is done through templates (templates are the next topic in this lesson).

Note the following rules of mapping relationships to foreign keys:

- If the table's primary key includes a foreign key, the foreign key column may be a result of an identifying relationship and may have already been added.
- For a mandatory 1:1 relationship, the foreign key will be placed on the mandatory side of the table and use the NOT NULL constraint to enforce the mandatory condition.
- If a 1:1 relationship is optional in both directions, the foreign key can be in either table.
- For a 1:M recursive relationship, the foreign key is added to the same table. The foreign key column refers to values from the primary key column.
- For a 1:1 recursive relationship, a unique foreign key is added to the table. The foreign key column refers to values of the primary key column.

Mapping Relationships to Foreign Keys



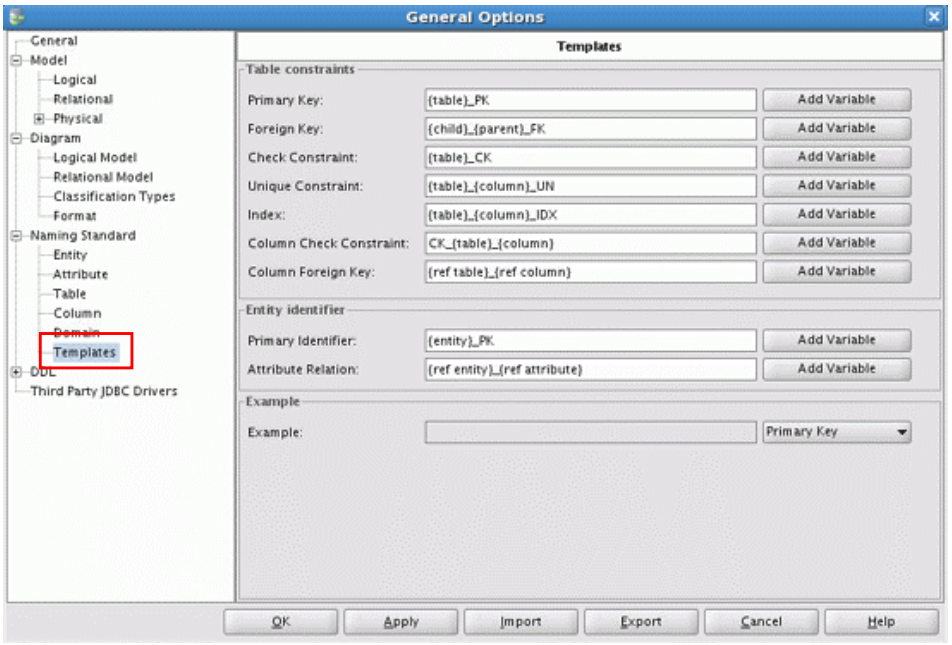
ORACLE

Mapping Relationships to Foreign Keys (continued)

Expand Relations to view the relations that will engineer to foreign keys, and click Engineer.

Defining Naming Templates

Tools > General Options > Naming Standard > Templates



ORACLE

Defining Naming Templates

You can define templates (name patterns) for keys, indexes, and constraints, by using combinations of predefined variables. To access the Naming Templates page, select Generation Options from the Tools menu, and then expand Naming Standard and select Templates.

For each element, you can use a set or predefined variable, to set the pattern. The predefined variables include the following:

- {table}
- {table abbr}
- {child}
- {child abbr}
- {parent}
- {parent abbr}
- {column}
- {column abbr}
- {ref column}
- {ref column abbr}
- {seq nr}
- {model}
- Alphanumeric constants

Defining Naming Templates

Examples:

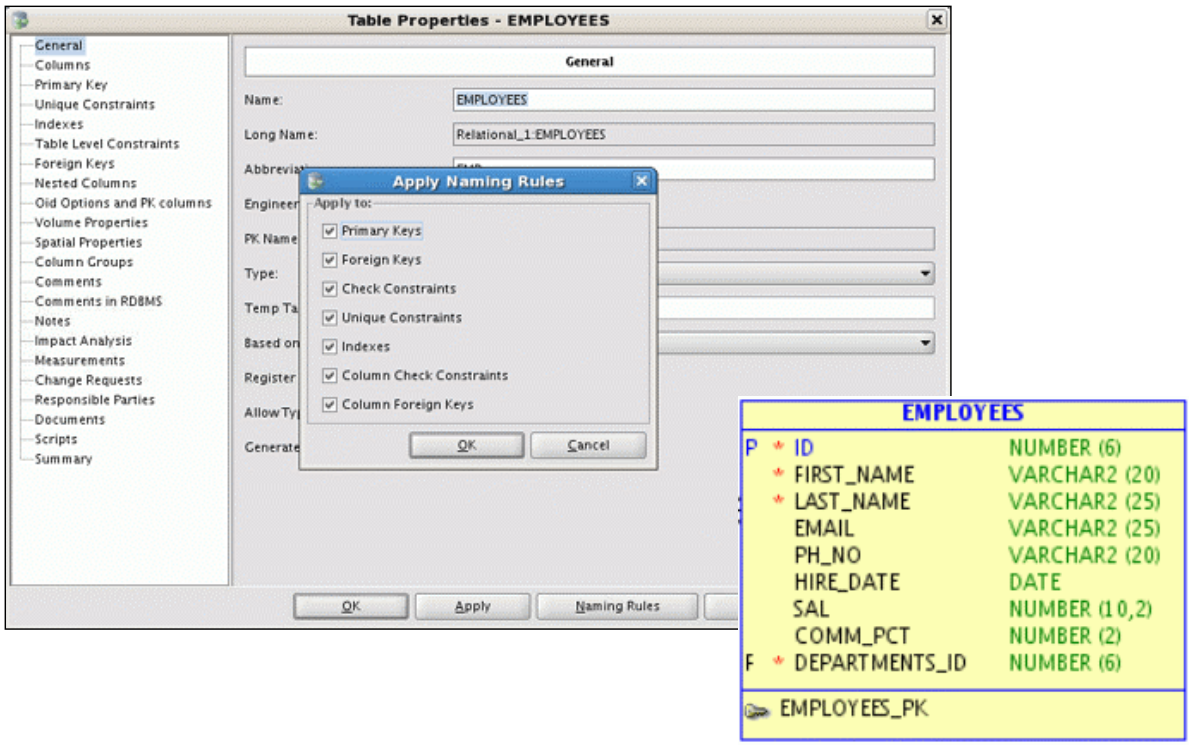
- Table name: ADMIN
- Model name: ORACLEDEMO

Template	Result
{table}_PK	ADMIN_PK
SUBSTR(7,4,FRONT,{model})	DEMO
SUBSTR(1,3,FRONT,{table})	ADM
SUBSTR(1,3,FRONT,TABLE)	TAB (where “TABLE” is a constant not a variable)
IX_SUBSTR(7,4,FRONT,{model})_SUBSTR (1,3,FRONT,{table})_{seq nr}	IX_DEMO_ADM_1

Defining Naming Templates (continued)

Using a combination of the predefined variables, and optionally the SUBSTR function with the syntax SUBSTR (index, length, direction, expression), you can build a naming pattern for each element. In the example in the slide, with a table name of ADMIN and model name of ORACLEDEMO, you can specify a template of SUBSTR_7,4,FRONT,{model}), which would result in the name DEMO.

Applying Templates to One Table

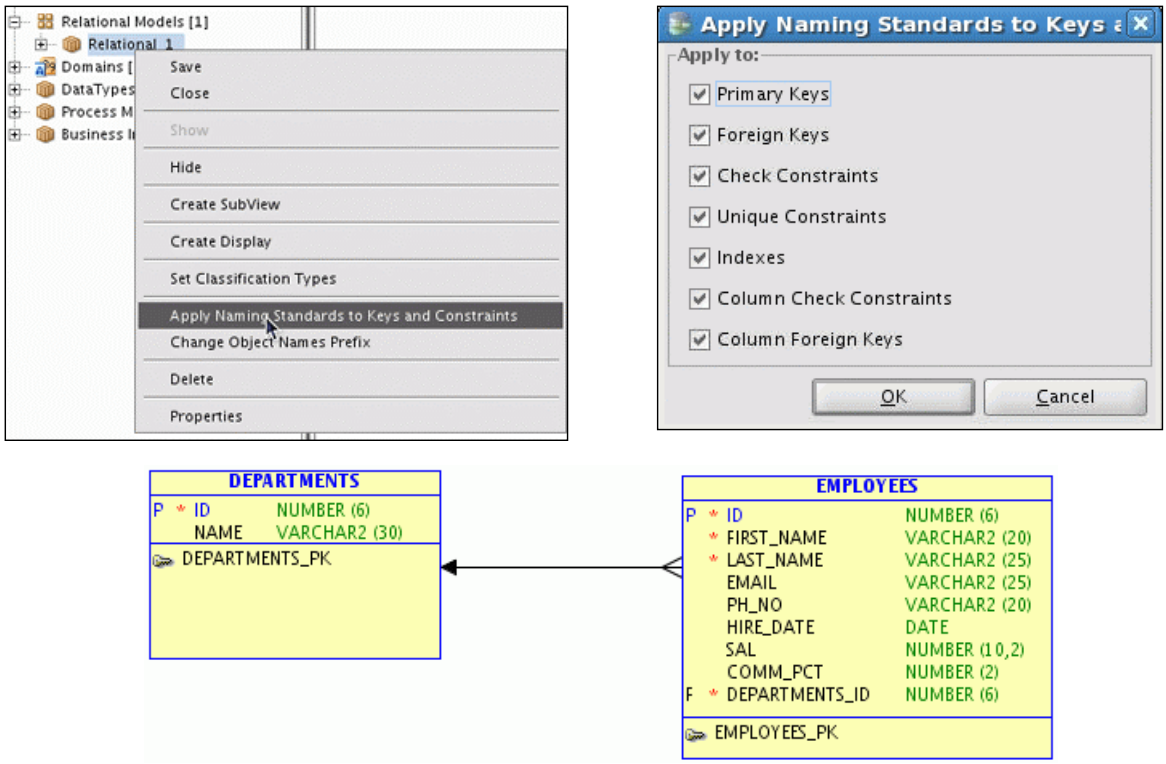


Applying Templates to One Table

After you make a change to the templates, you can apply them to one table by performing the following steps:

1. Double-click the table.
2. Click Naming Rules.
3. Select the check box for each object type that you want to apply the template to, and click OK.

Applying Templates to the Relational Model



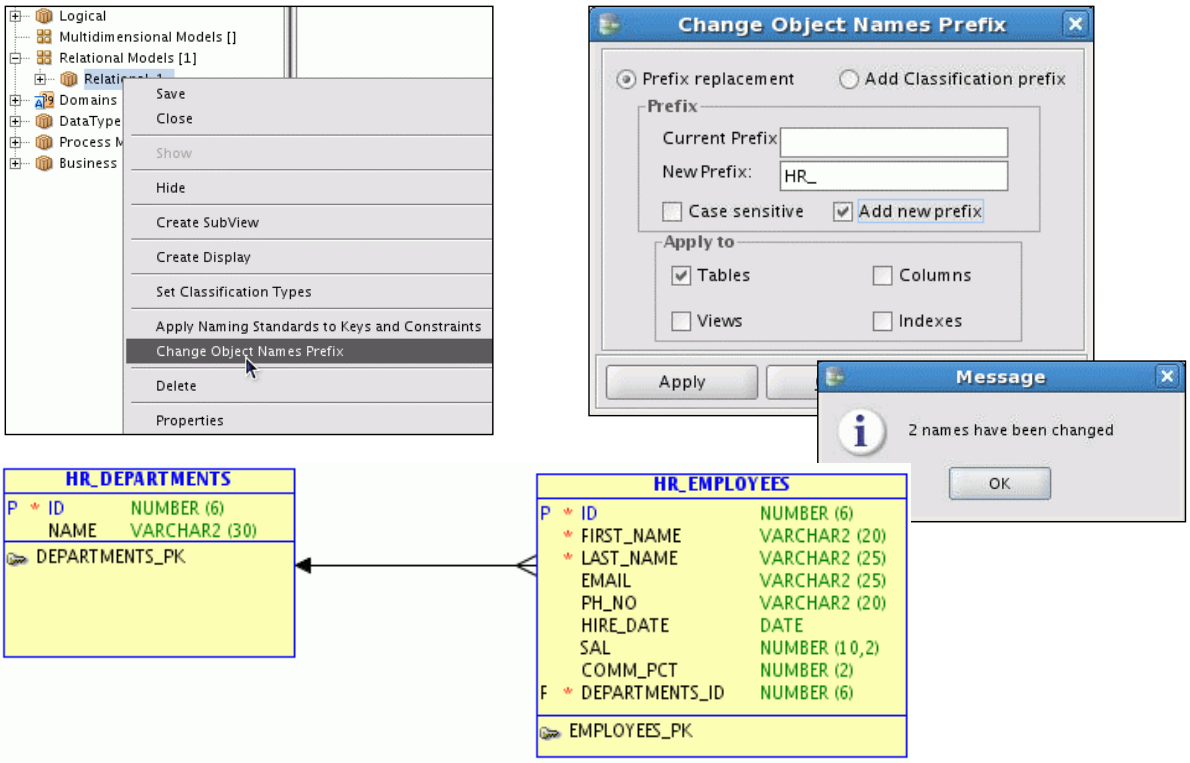
Applying Templates to the Relational Model

You can also apply the template to the entire relational model. Perform the following steps:

1. Right-click the Relational model in the navigator, and select “Apply Naming Standards to Keys and Constraints.”
2. Select the object types that you want to apply the templates to, and click OK.
3. Notice that the names of the primary keys changed from using the abbreviation to using the template, which was {table}_PK.

Note that in Oracle SQL Developer Data Modeler release 2.0 patch 1, this method only works for column foreign keys.

Managing Prefixes



Managing Prefixes

Often prefixes are added to the names of objects in order to represent different aspects of their life cycle, ownership, or usage. Two alternatives are available: to permanently change or to temporarily change the object names.

Permanently changing the object name: In some cases, you may want to replace or add a prefix to objects in the design. You can apply these changes to tables or views, (columns and indexes also are supported) represented in a specific subview, or to objects in an entire relational model in which you either add a new prefix or replace an existing prefix.

Temporarily change the name of the object when you generate the DDL script (discussed in a later lesson). In this case, define the old_prefix and then apply the name substitution during the DDL generation. This approach has no impact on the names of the objects in the models.

To add a prefix to your model, perform the following:

1. Right-click the Relational model and select Change Object Names Prefix.
2. Specify a new prefix, click “Add new prefix,” select the objects that you want to apply it to, and click Apply.
3. A message dialog box appears, indicating how many names have changed. Click OK.
4. Note that the names of the tables now have the prefix.

Quiz

Different naming standards can be applied for primary keys and foreign keys through the use of naming templates.

a. True
b. False

Answer: a

Quiz

A glossary is used to: (Select all that apply.)

a. Define a name template
b. Define abbreviations
c. Apply a prefix
d. Define separators

Answer: b, d

Practice 15-1 Overview: Create an Initial Relational Model

This practice covers the following topics:

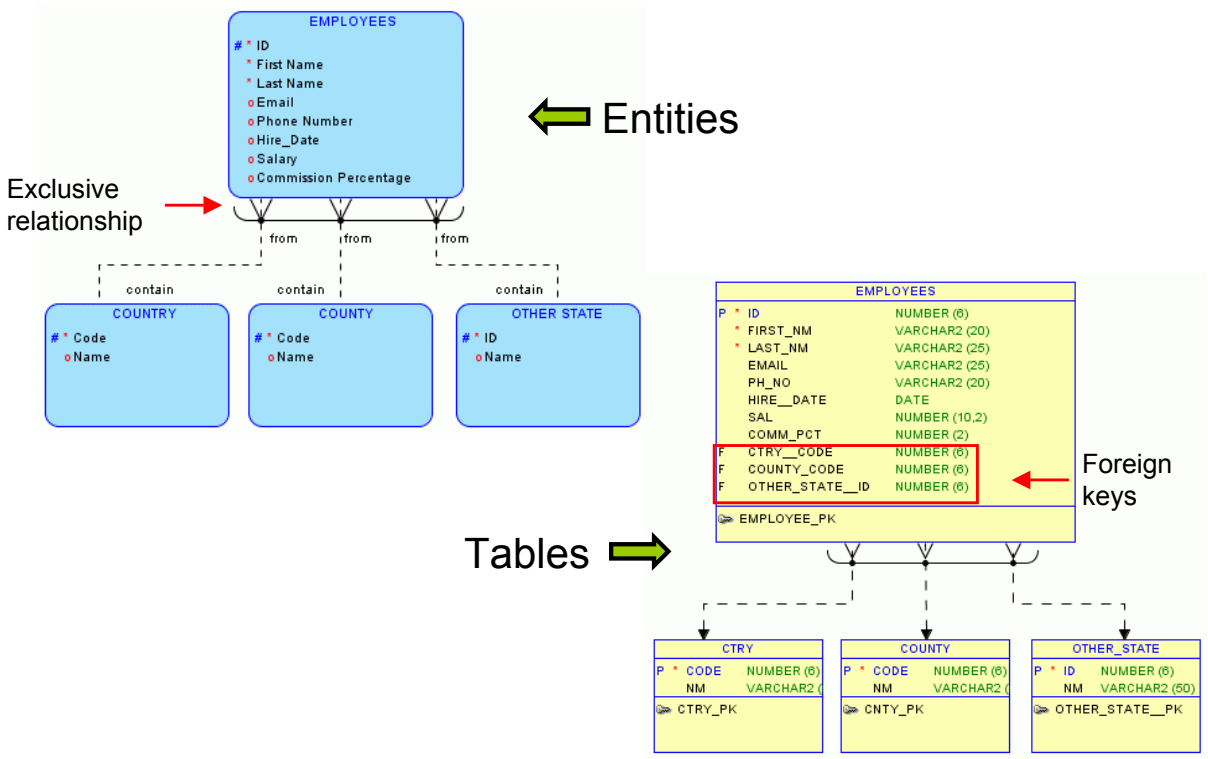
1. Creating a glossary
2. Engineering the model to a relational model.
3. Creating a Name template
4. Adding a prefix to each table

ORACLE

Practice 15-1 Overview: Creating an Initial Relational Model

In this practice, you map a logical model to a relational model.

Mapping Exclusive Relationships to Foreign Keys

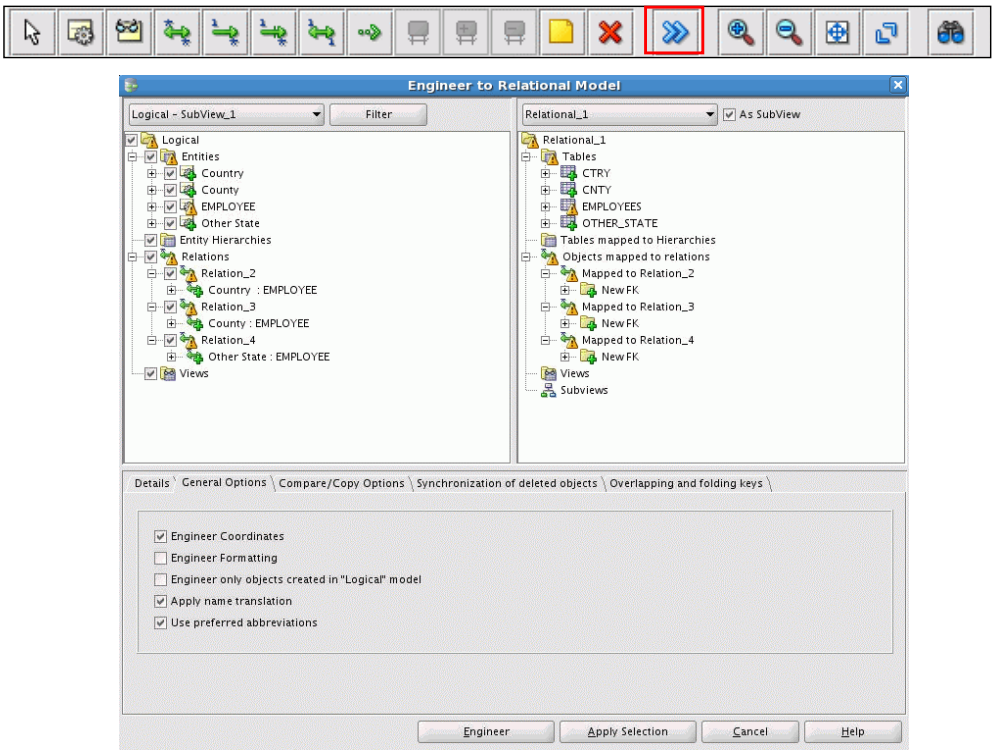


Mapping Exclusive Relationships to Foreign Keys

As you learned in a previous lesson, an exclusive relationship is when two or more mutually exclusive relationships from the same entity use an arc. In the example in the slide, an EMPLOYEE may be from a COUNTRY, COUNTY, or OTHER_STATE.

When an exclusive relationship is engineered to a relational model, a foreign key column is created for each relationship included in the arc.

Engineering Exclusive Relationships

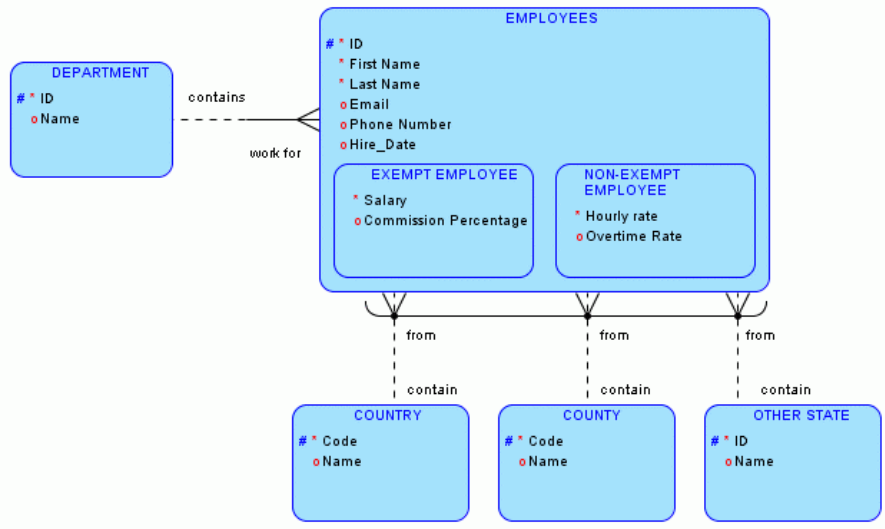


ORACLE

Engineering Exclusive Relationships

When engineering exclusive relationships, a new foreign key is created for each relation.

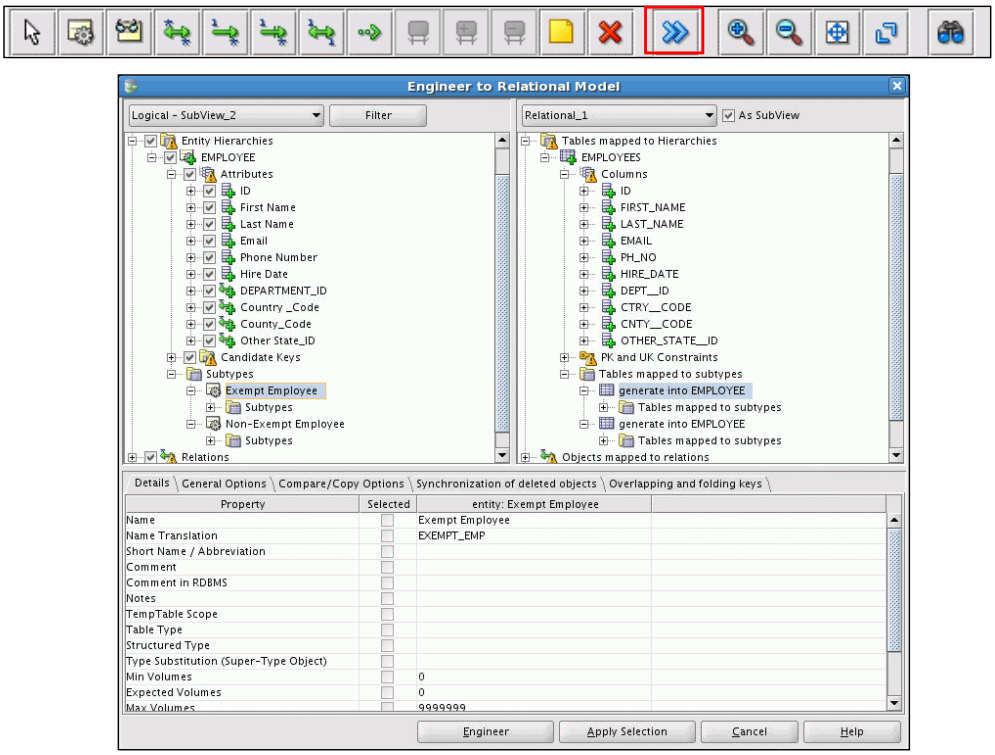
Mapping Subtypes to Tables



Mapping Subtypes to Tables

As you learned in a previous lesson, an entity type hierarchy is used when you must model exclusive entity types that have common attributes and common relationships. In the example in the slide, an EMPLOYEE can be either an EXEMPT EMPLOYEE or a NON_EXEMPT EMPLOYEE. Specific attributes must be stored for each EMPLOYEE type.

Engineering Subtypes



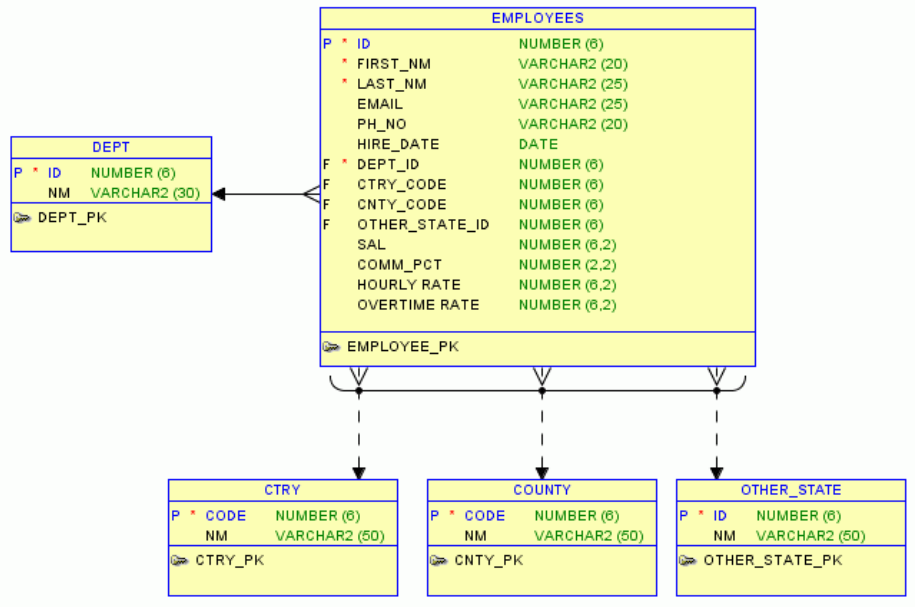
ORACLE

Engineering Subtypes

The default engineering strategy for an entity type hierarchy is to create a single table. When you engineer with the default strategy the subtypes attributes will be added to the super type table. In the example in the slide, the EXEMPT EMPLOYEE and NON-EXEMPT EMPLOYEE subtype entities and all their attributes will be engineered into the EMPLOYEES table.

To determine which forward engineering strategy to use, the designer should look at the DFD or business needs/ application requirements and also at the data access requirements. The DFD, crosschecked with the ERD, will show whether, for example, the super type is generally queried and updated (a one-table design is the better choice), or whether there is an emphasis on using subtype info by itself (separate table design).

Mapping Subtypes to a Single Table



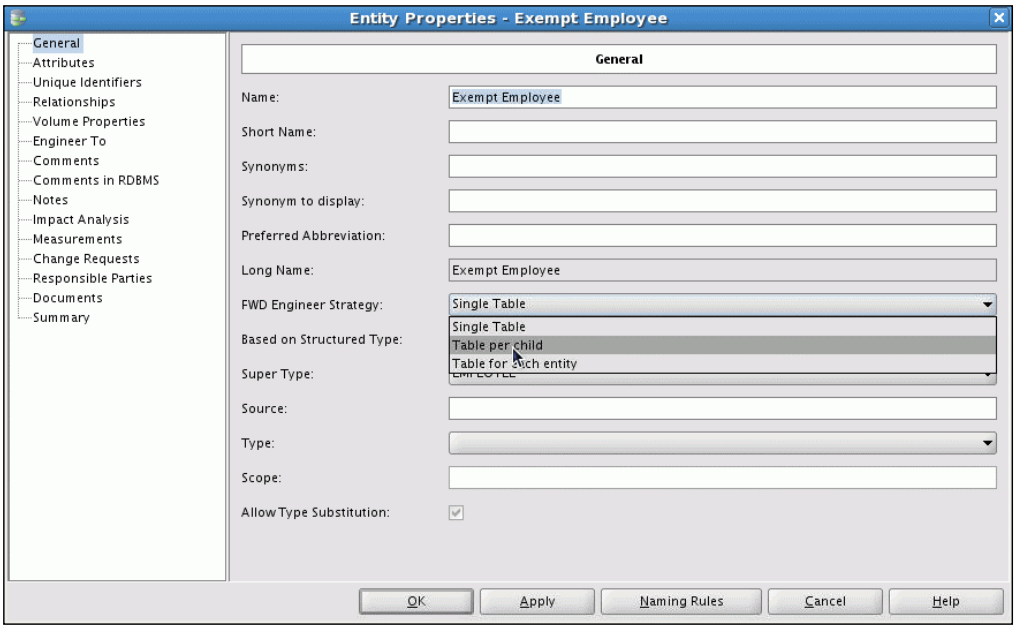
ORACLE

Mapping Subtypes to a Single Table

The result of engineering the subtypes to a single table is that all the attributes from both subtypes are now contained in the EMPLOYEES super type table.

Changing the FWD Engineering Strategy

"Table per child" produces two tables for each of the subtypes.

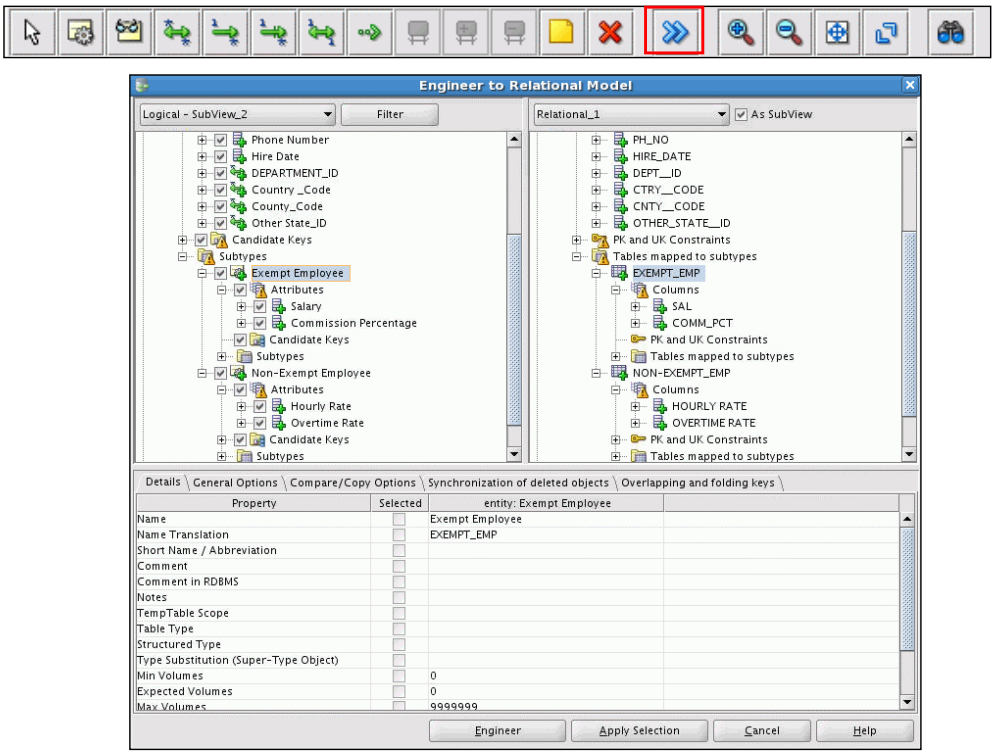


ORACLE

Changing the FWD Engineering Strategy

You can change the way in which the engineering task works from the subtype entity definition. Double-click one of the subtypes in the entity type hierarchy. Change the FWD Engineering Strategy setting to "Table per child," and click OK. All the subtype definitions will have the same change.

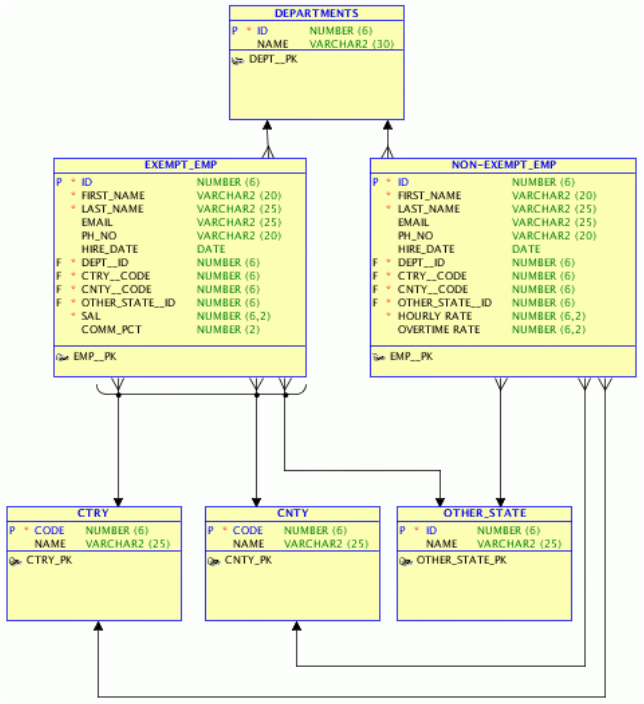
Engineering Subtypes to Table per Child



Engineering Subtypes to Table per Child

When you engineer this time, as you see in the Engineering window, two tables will be created, one for each subtype. All the attributes from the super type will appear in both subtype tables.

Mapping Subtypes for a Table per Child



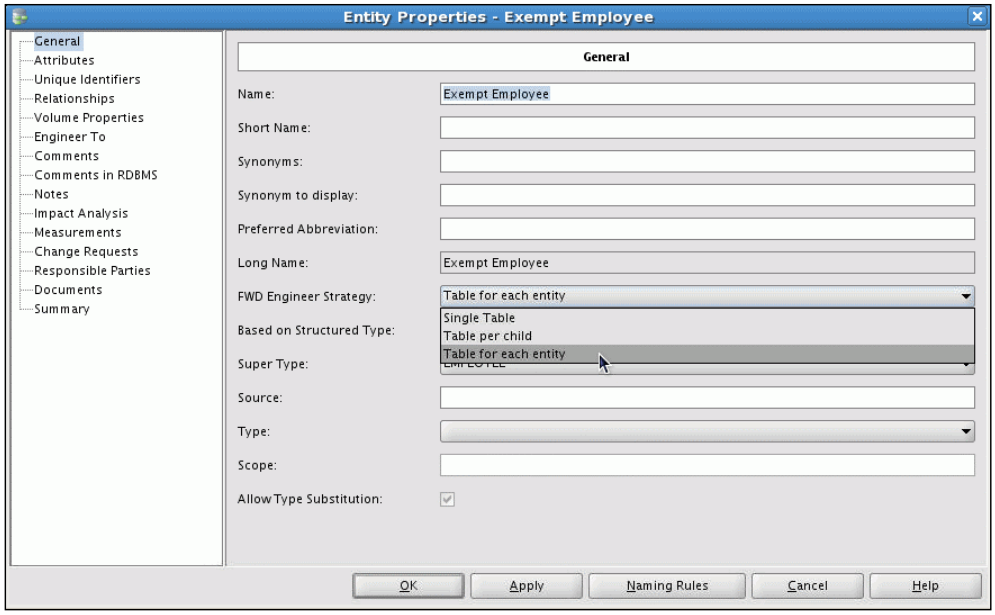
ORACLE

Mapping Subtypes for a Table per Child

In the example in the slide, two tables were created, one for each subtype in the logical model. All the attributes from the EMPLOYEE super type were added to each of the subtype tables.

Changing the FWD Engineering Strategy

"Table for each entity" produces three tables: one for the super type, and one for each subtype.

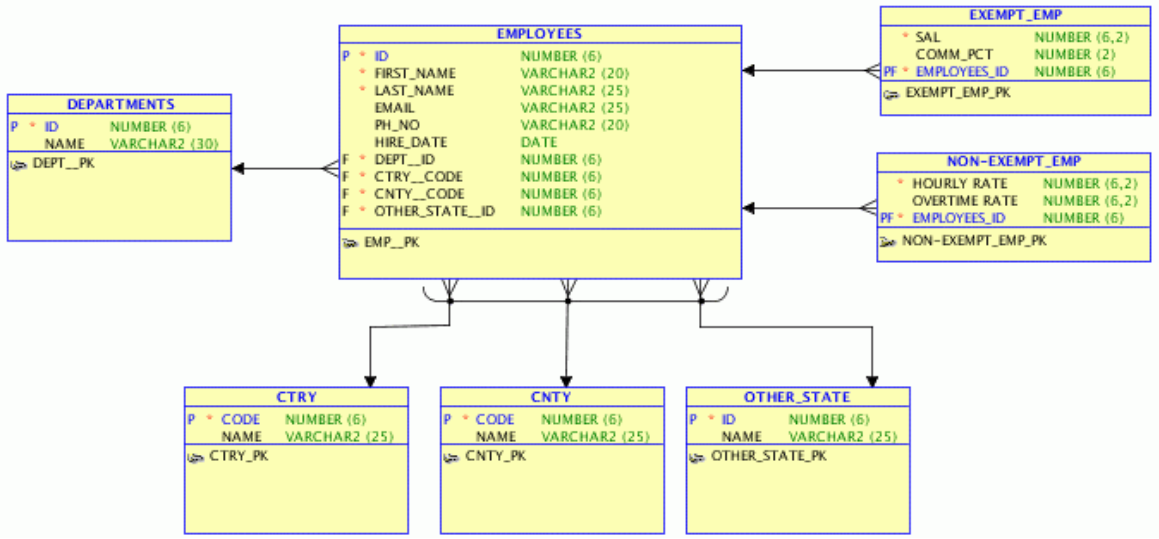


ORACLE

Changing the FWD Engineering Strategy

When the third option is selected, "Table for each entity," you will get one table for the super type entity and one for each subtype entity.

Mapping Subtypes for a Table for Each Entity



ORACLE

Mapping Subtypes for a Table for Each Entity

For the “Table for each entity” option, you see that the EMPLOYEES super type table is created, as well as a table for each subtype, EXEMPT_EMP and NON-EXEMPT_EMP. A primary foreign key was created in each of the subtype tables that refers to the EMPLOYEE table.

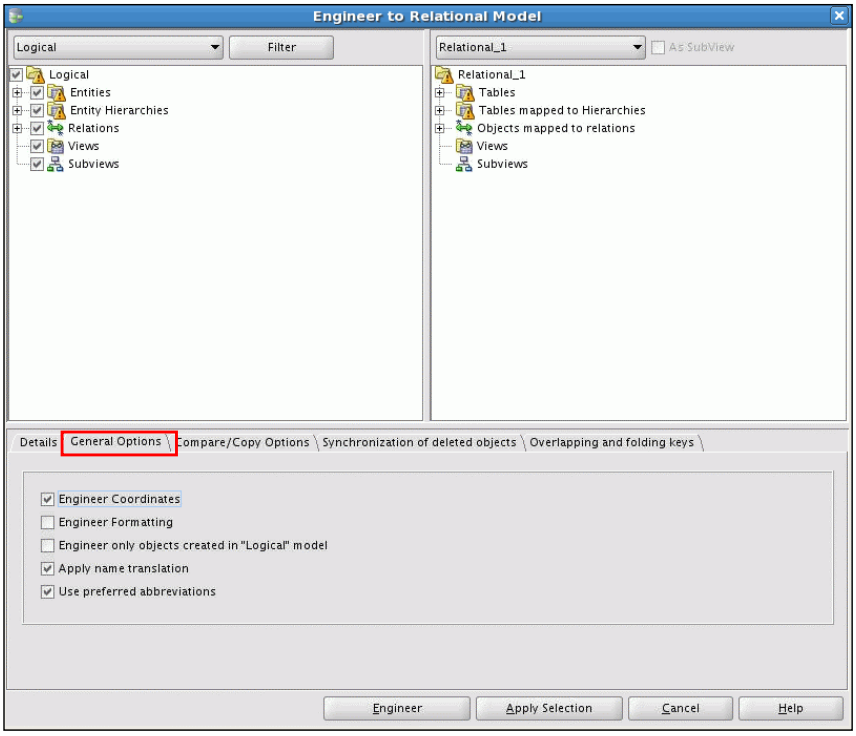
Quiz

If you only want to create a table for each subtype in an entity type hierarchy in the relational model, which option would you select for FWD Engineering Strategy?

a. Table per subtype
b. Single Table
c. Table for each entity
d. Table per child

Answer: d

Applying General Options



ORACLE

Applying General Options

Under the General Options tab in the Engineering window, you can specify a number of options:

Engineer Coordinates: Engineers the coordinates so that the objects appear in the same sequence that they were in the logical model in the relational model.

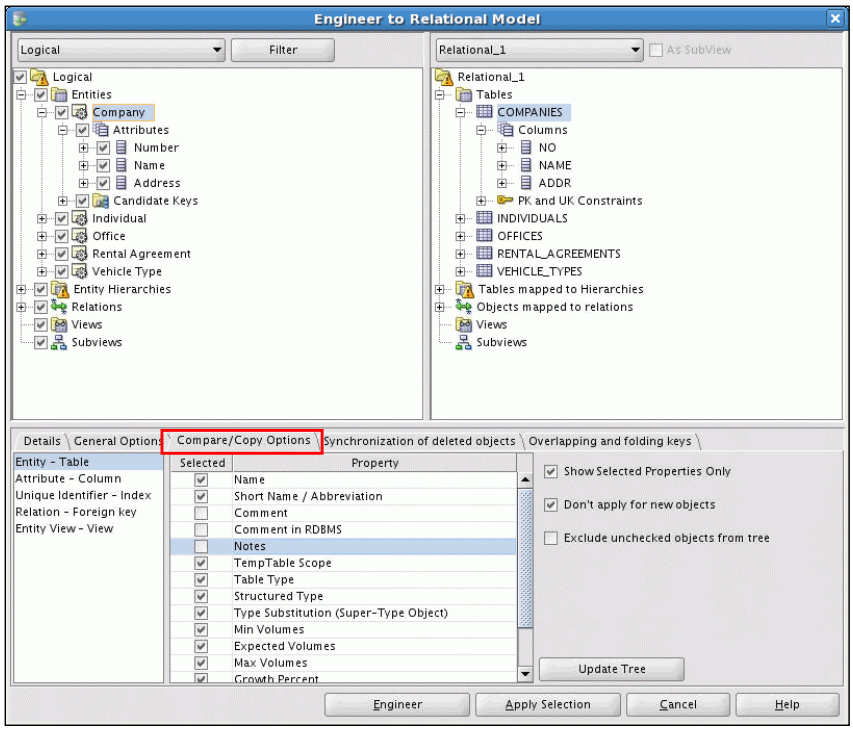
Engineer Formatting: Engineers the formatting specified on the General Options tab. You can indicate what color the table should be in the relational model.

Engineer only objects created in "Logical" model: You may have to run the engineering task many times after changes occur. You can specify that only objects created in the logical model are created in the relational model using this option.

Apply name translation: Engineers the abbreviations specified in the glossary and entity and attribute properties windows.

Use preferred abbreviations: Applies the abbreviation from the glossary and entity and attribute properties windows.

Setting Compare/Copy Options



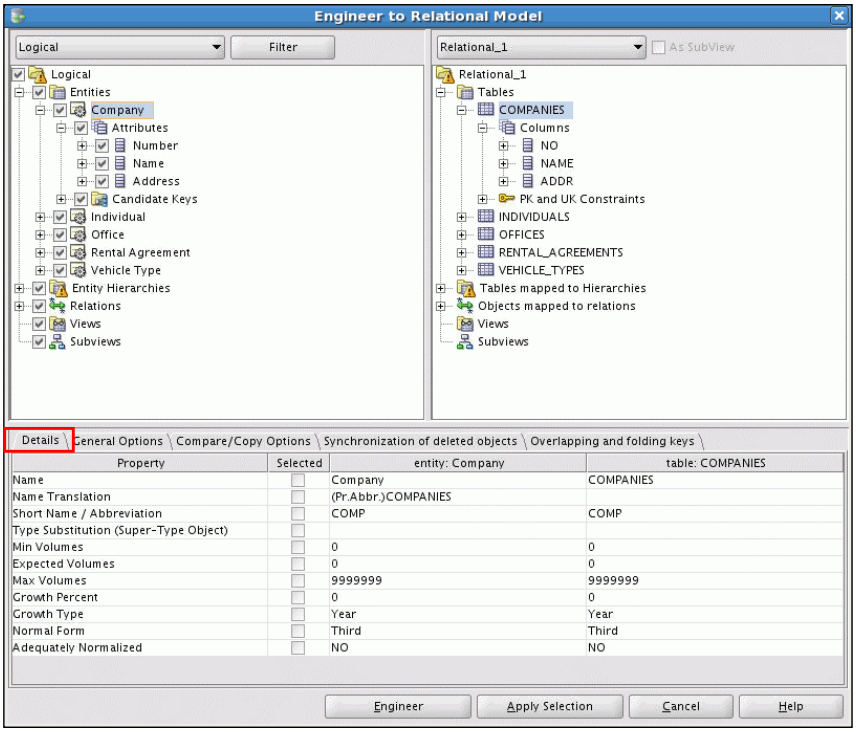
ORACLE

Setting Compare/Copy Option

The Compare/Copy Options tab in the Engineering window allows you to compare the properties of logical and relational objects during the engineering process. Select the mapping pair in the left pane to see a list of the properties available for that pair. In the example in the slide, you select the mapping pair Entity – Table which shows you a list of properties in the middle pane. If you do not need to compare a property, you can deselect it from the list.

The options in the right pane allow you to show or exclude various levels of detail for tables, entities, views, and relationships. This does not apply to attributes or columns.

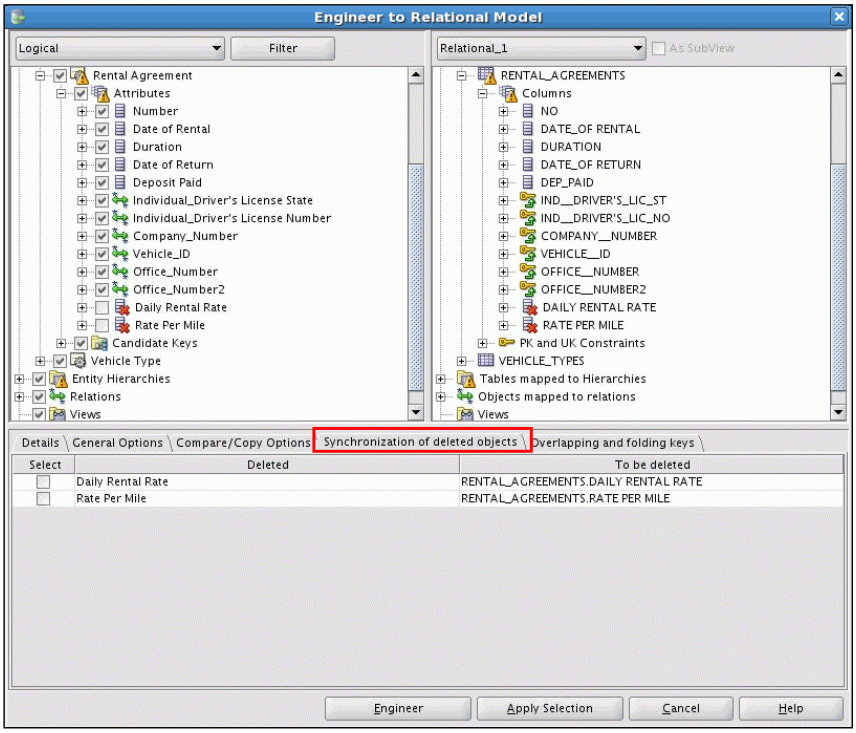
Viewing the Mapping Comparison



Viewing the Mapping Comparison

To see the comparison based on the list of properties from the Compare/Copy Option tab, you expand the tree on either side and select an object. In the bottom pane, select the Details tab to see the comparison. In the example in the slide, the Company entity was selected, and the list of properties for that mapping pair is shown. Notice that the properties deselected in the Compare/Copy Options tab do not appear in the Details tab.

Synchronizing Deleted Objects



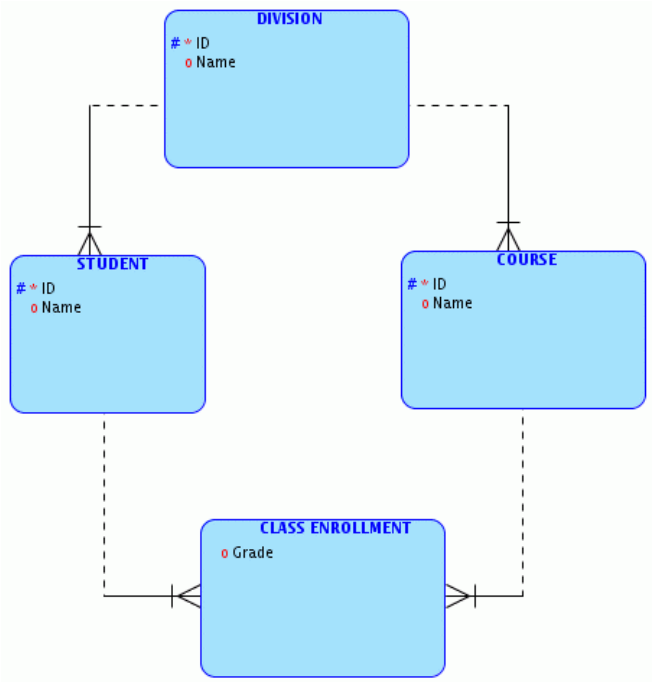
Synchronizing Deleted Objects

By default, deleted objects in the logical model are not selected for engineering. You can expand each object in the tree to view the list of deleted objects, or you can select the “Synchronization of deleted objects” tab to see the list of deleted objects for the entire logical model. From the list of deleted objects, you can select whether you want to engineer the change to the relational model or not.

Identifying Overlapping and Folding Keys

Two attributes in the same entity relate to the same unique identifier attribute.

You can fold the keys into one column in the relational model during engineering.

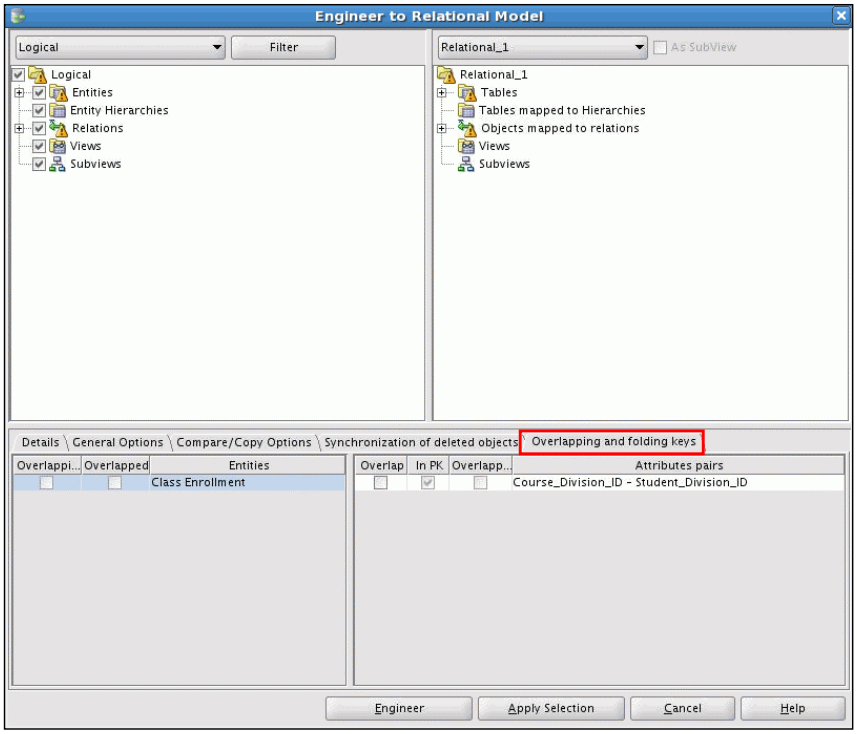


ORACLE

Identifying Overlapping and Folding Keys

In the example in the slide, the ID attribute in the DIVISION entity is repeated in the CLASS ENROLLMENT entity because of the identifying relationships with the STUDENT and COURSE entities. You can overlap or fold the keys in the relational model during engineering.

Identifying Overlapping and Folding Keys



ORACLE

Identifying Overlapping and Folding Keys (continued)

The “Overlapping and Folding keys” tab displays the overlapping keys. You can click the Overlap check box for the listed attributes pairs so that only one column in the relational model will be created.

Summary

In this lesson, you should have learned how to:

- Describe why a database design is needed
- Decide on naming conventions and rules
- Perform a mapping between a logical and a relational model
- Utilize the Oracle SQL Developer Data Modeler facility

ORACLE

Summary

In this lesson, you should have learned how to use the Oracle SQL Developer Data Modeler engineering tool to engineer your logical model to a relational model, and how logical objects map to relational objects.

Practice 15-2 Overview: Forward Engineer a Model

This practice covers the following topics:

1. Defining the FWD engineering strategy for an entity type hierarchy
2. Forward engineering the logical model and evaluating the results

ORACLE

Practice 15-2 Overview: Forward Engineer a Model

In this practice, you forward engineer a model with an entity type hierarchy.

Evaluating Your Design for Database Creation

VI

Overview

In this unit, you examine the following areas:

- Lesson 16: Analyzing Your Relational Model
- Lesson 17: Denormalizing Your Design to Increase Performance
- Lesson 18: Defining Your Physical Model
- Lesson 19: Generating Your Database

ORACLE

16

Analyzing Your Relational Model

Objectives

After completing this lesson, you should be able to:

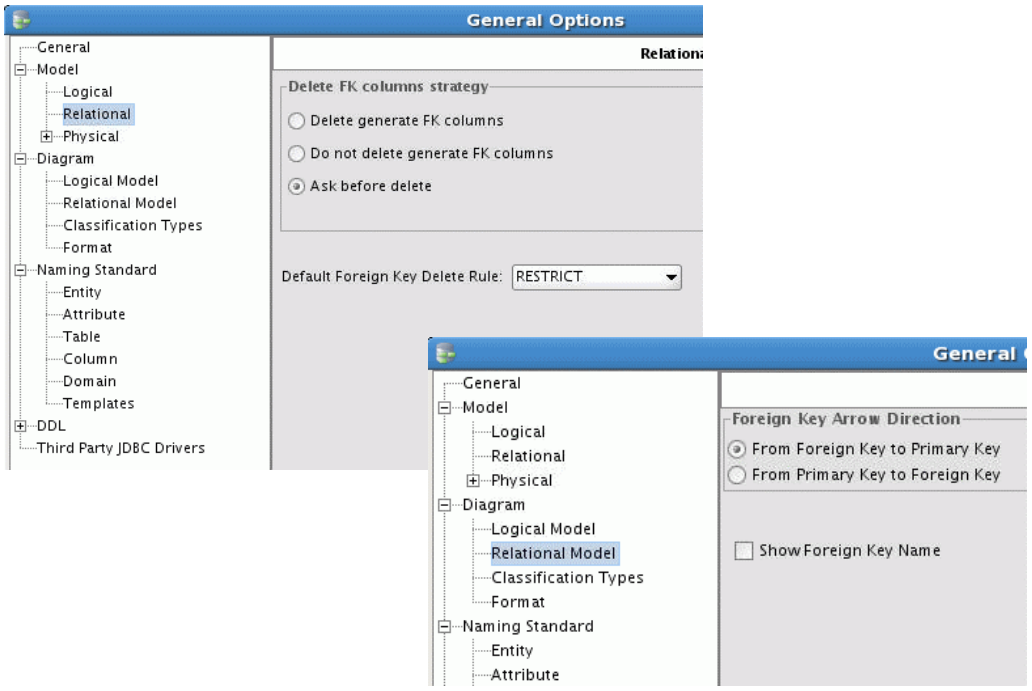
- Modify table properties according to requirements
- Determine when to create an index
- Determine when to create a view

ORACLE

Objectives

In this lesson, you learn how to modify table properties and when to create an index or view.

General Options: Relational Diagram



ORACLE

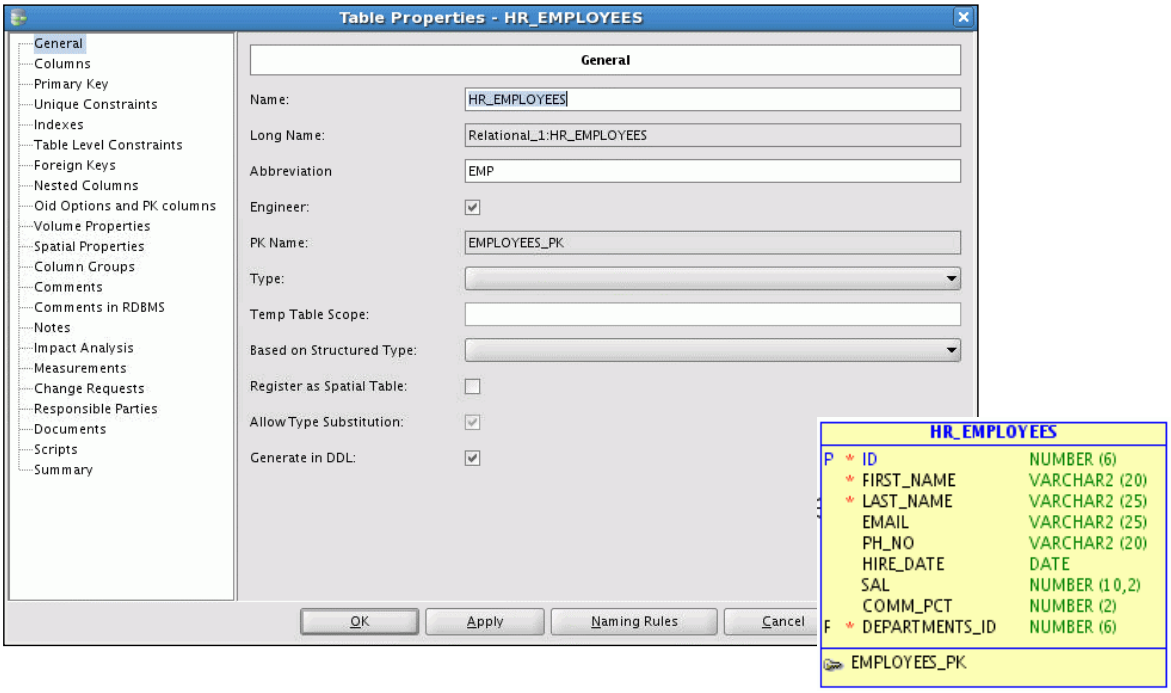
General Options: Relational Diagram

You can specify a number of general options for the relational model. Select Tools > General Options to access the General Options window.

Expand Model and select Relational to find the delete FK column strategy and delete rule for foreign keys.

Expand Diagram and select Relational Model to see the Foreign Key Arrow Direction default and whether foreign key names are displayed by default.

Reviewing Table Properties



ORACLE

Reviewing Table Properties

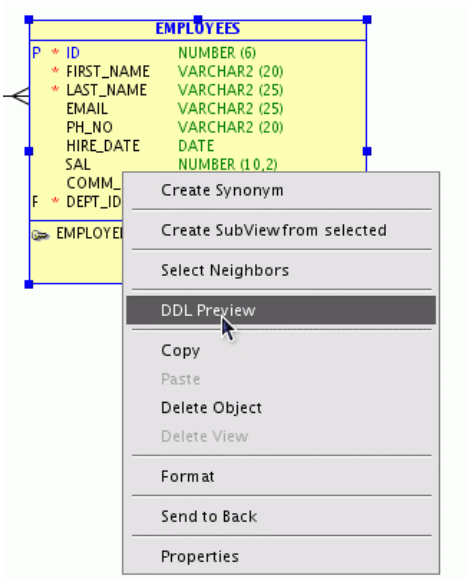
Double-click the table in the relational model to review its table properties. You can specify a number of general properties for each table. Notice that many of the fields are already populated based on your logical model.

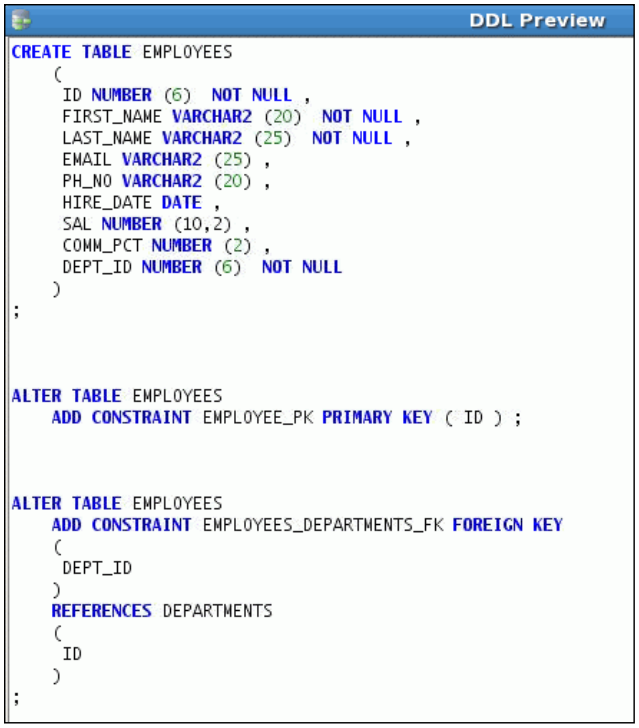
Under the General option, you can specify the following:

- **Type:** Classification type, from the list defined in the General Options. Examples: Fact, Dimension, Logging, Summary, Temporary.
- **Temp Table Scope:** For a table classified as Temporary, you can specify a scope, such as Session or Dimension.
- **Register as Spatial Table:** For a table with a column of type SDO_GEOMETRY, this creates the spatial index and inserts the appropriate entry in the USER_SDO_GEOM_METADATA view.
- **Allow Type Substitution:** For a structured type with Reference disabled, or for a structured type applied to a table, this controls whether a substitutional structured type generation in the DDL is allowed.

You will examine some of the other table properties in the next few slides.

Previewing the DDL for a Table





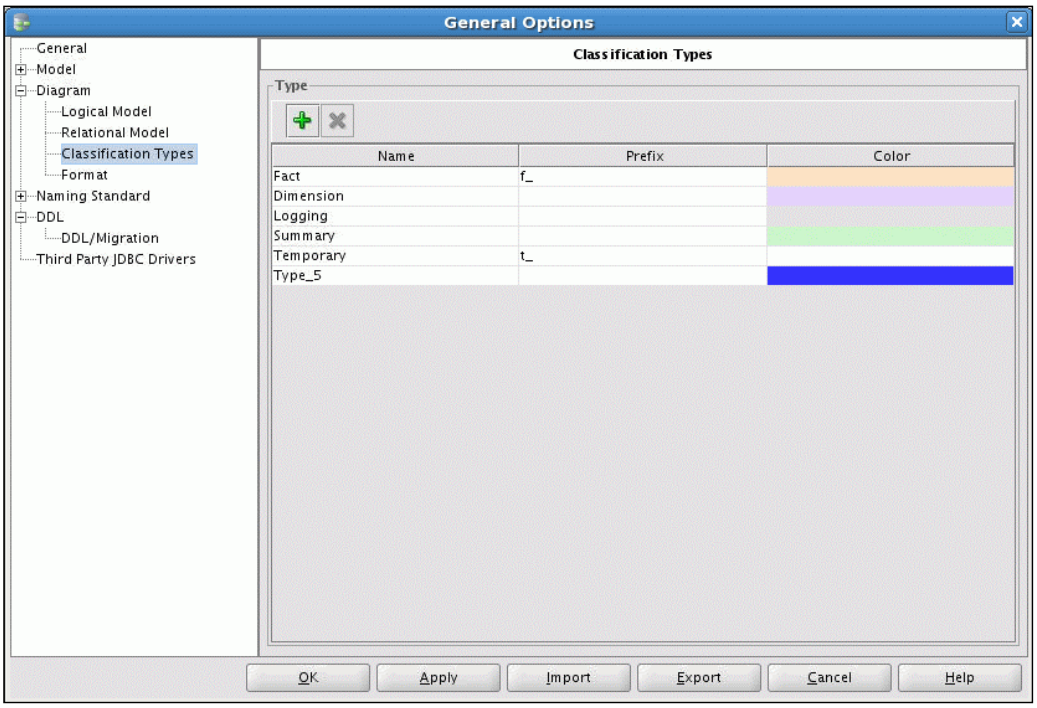
ORACLE

Previewing the DDL for a Table

You can preview what the DDL will look like for a table. Right-click the table and select DDL Preview. The DDL Preview window appears. When done reviewing, click Close.

More about DDL generation will be discussed in a later lesson.

General Options: Classification Types



ORACLE

General Options: Classification Types

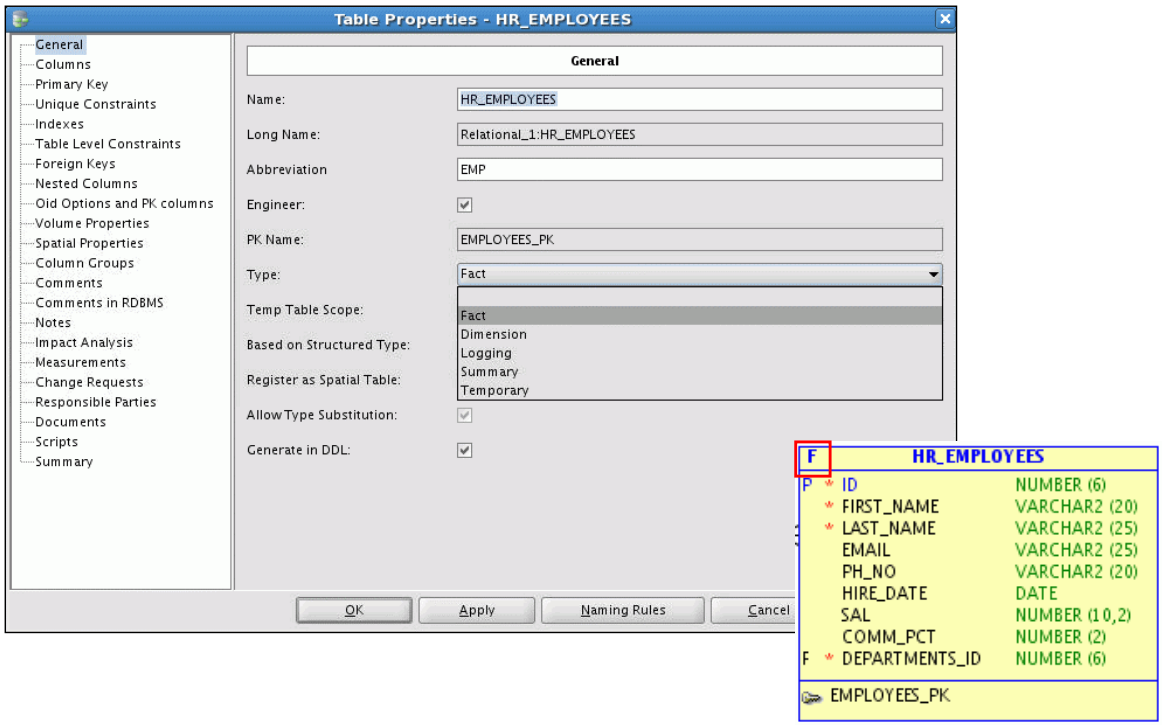
Classification types indicate what type of entity or table it is.

The default types are: Fact, Dimension, Logging, Summary, and Temporary. These default types may have different meanings to different people so it may be useful to define them as part of your standards.

To add additional classification types, click the add icon.

After your classification types are defined, you can reference them in the entity or table properties.

Assigning a Classification Type to One Table

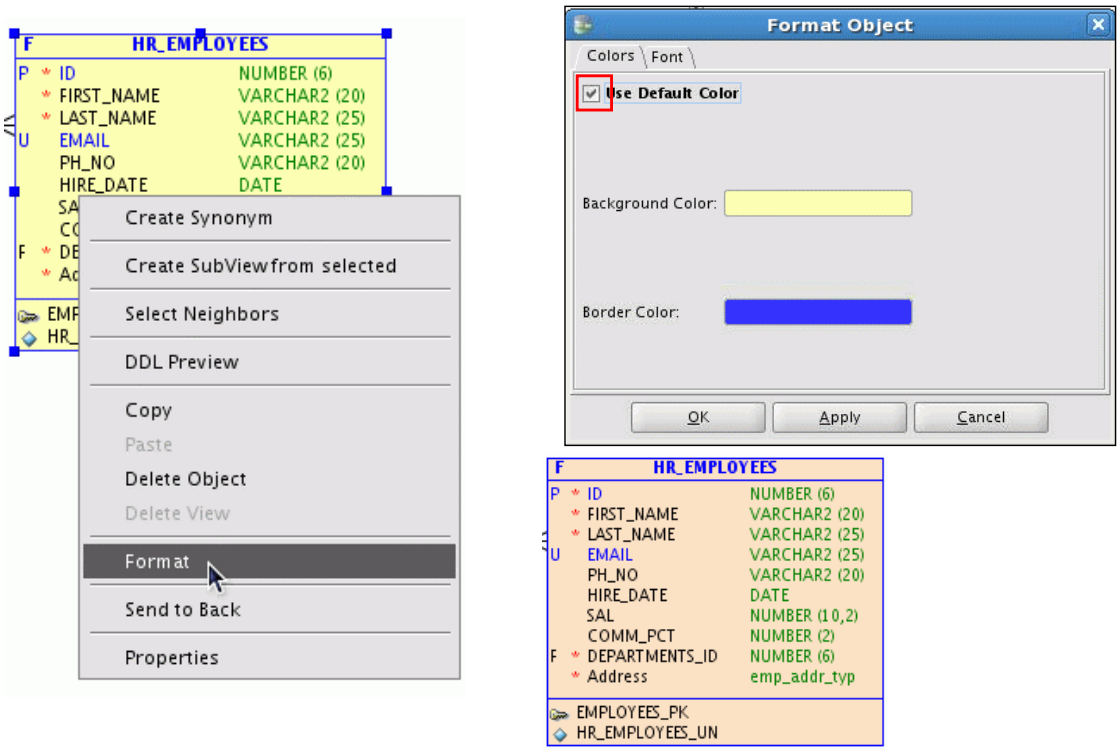


Assigning a Classification Type to One Table

To assign a classification type to one table, select the type from the drop-down list. An indicator appears in the upper-left corner of the table to indicate the table's type.

Additional types added through the General Options > Classification Type area, are all indicated by "T."

Changing the Color for Classified Tables

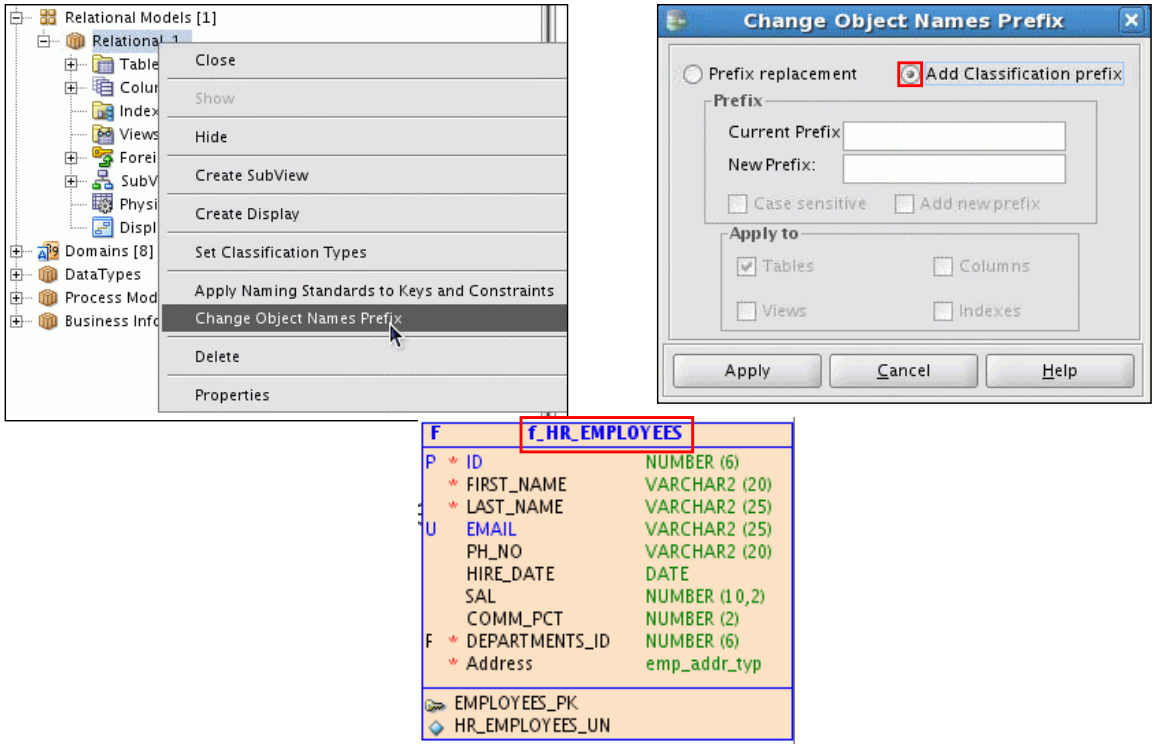


Changing the Color for Classified Tables

After you set a classification type for a table, it will use the color of the classification type if Use Default Color in the object format is set. If the color is not set after assigning a classification type, you must check the format by performing the following steps:

1. Right-click the table that has the classification type set and select Format.
2. Make sure that Use Default Color is checked, and click OK.
   The table color will change to the color set in the Classification Types general options.

Changing the Prefix for Classified Tables



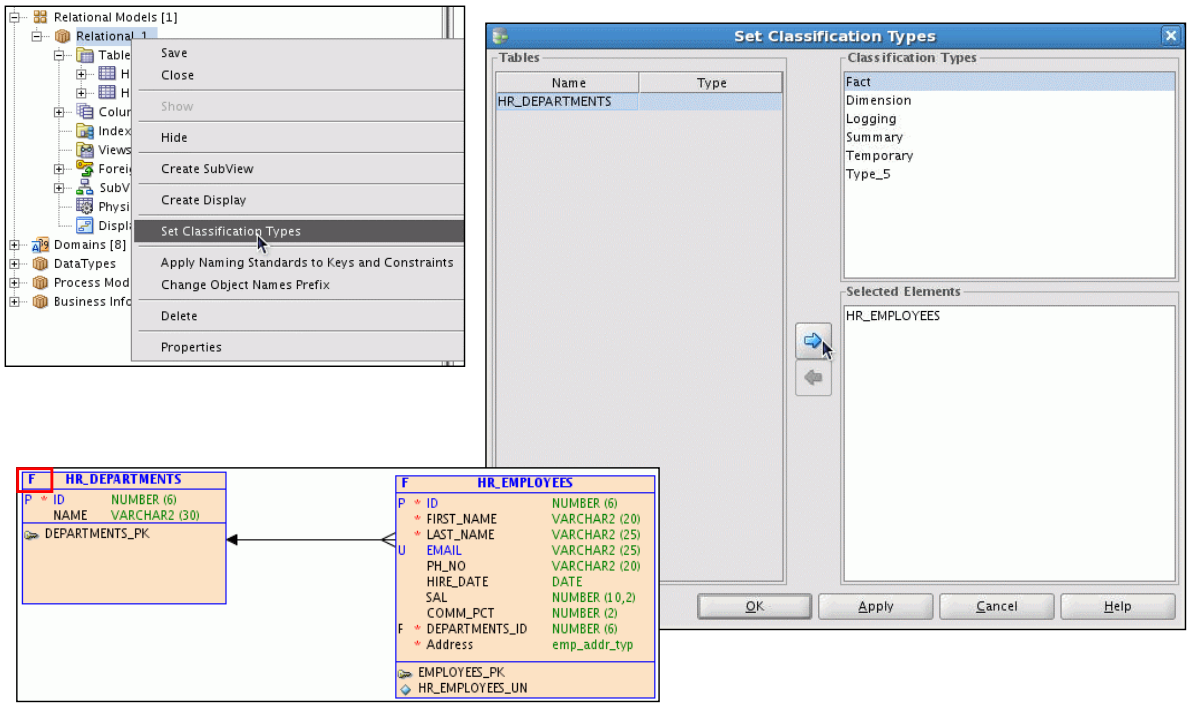
ORACLE

Changing the Prefix for Classified Tables

To change the prefix for tables that you assigned a classification type, perform the following steps:

1. In the Object Browser Hierarchy, right-click the relational model and select Change Object Names Prefix.
2. Select “Add Classification Prefix” and click Apply. A window indicating how many objects were changed appears.
3. Click OK.
4. Notice that the name of the table now has the prefix that you specified in the Classification Types general options.

Assigning Classification Types to Multiple Tables



Assigning Classification Types to Multiple Tables

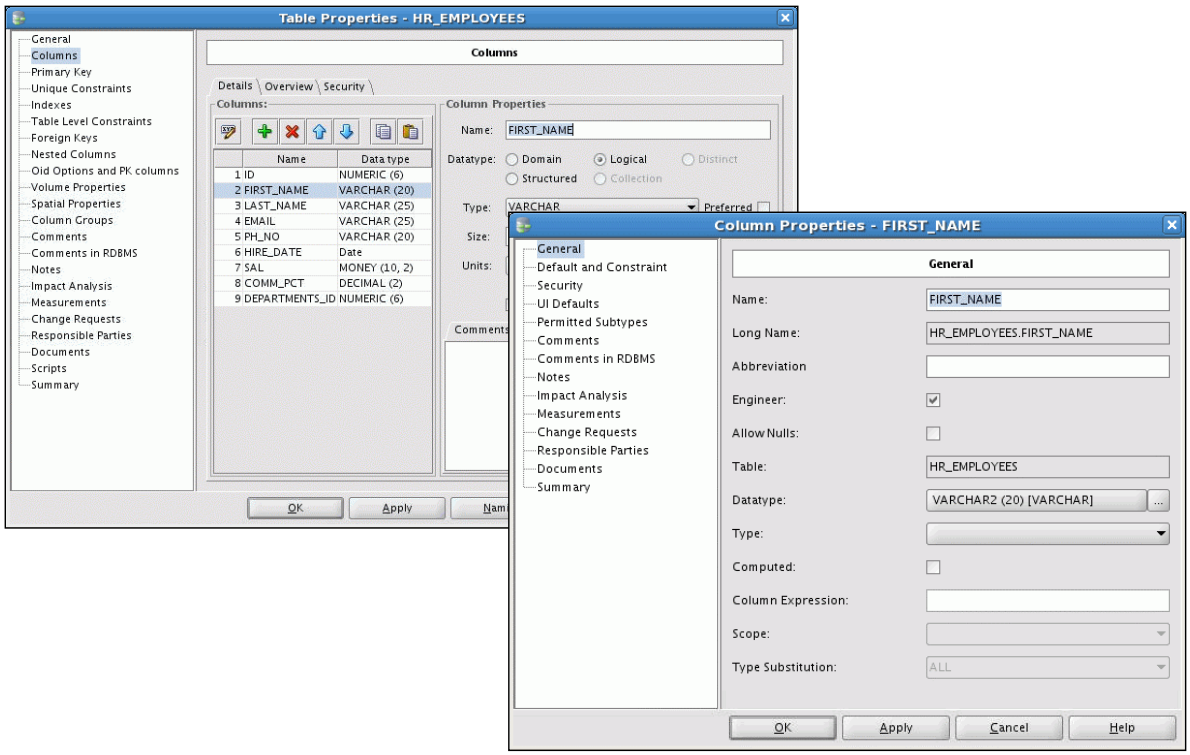
You can also assign multiple tables to one classification type by performing the following steps:

1. In the Object Browser Hierarchy, right-click the relational model and select Set Classification Types. The Set Classification Types window appears.
2. Select the classification type in the Classification Types list, select the table in the Tables list, and click the add arrow to add the table to the Selected Elements list.
3. Click OK.

Notice that both tables now have the same classification type.

Note: To change the table to a different classification type, you must delete it from one classification type, select the other classification type, and then add the table to it.

Reviewing Column Properties



ORACLE

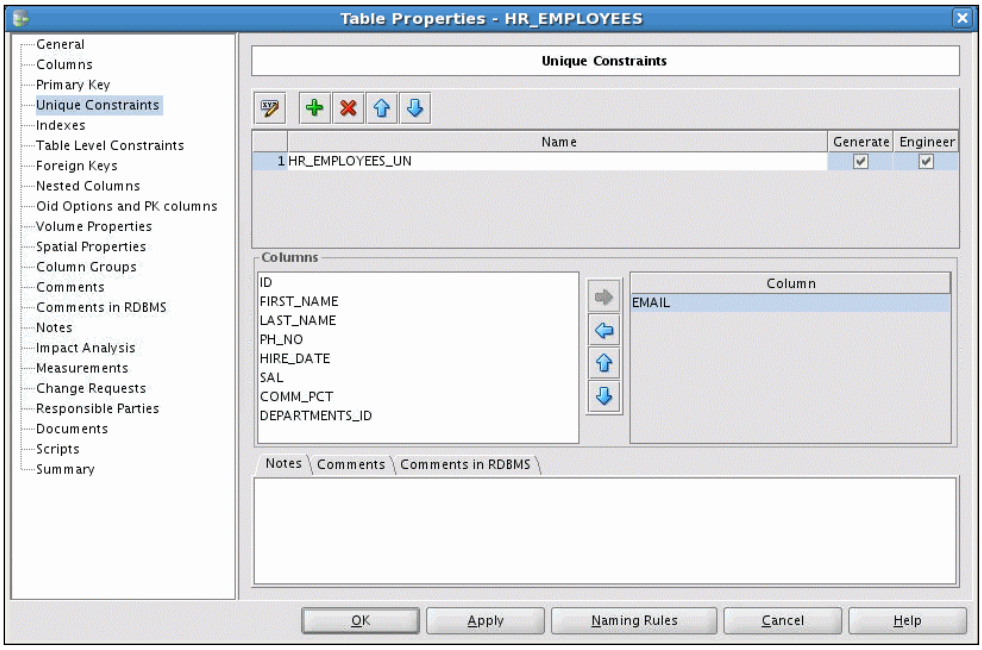
Reviewing Column Properties

To review the properties for a specific column, select Columns in the left navigator in the Table Properties window and double-click the column that you want to review properties for.

Some of the properties that you can define for a specific column are:

- **General:** Defines the naming and engineering characteristics of the column. In addition, you can specify what type of column it is. For example, Manual, System, Derived, or Aggregate. Depending on what type, you can specify additional information in other fields on this page.
- **Default and Constraint:** Specifies a default value and/or constraint for this column where you can define a range or list of values.
- **Security:** Specifies whether you have personally identifiable information (PII), sensitive information, or information that needs to be masked for development databases.
- **UI Defaults:** Specifies the format that this column should have when included within an application.

Defining a Unique Constraint



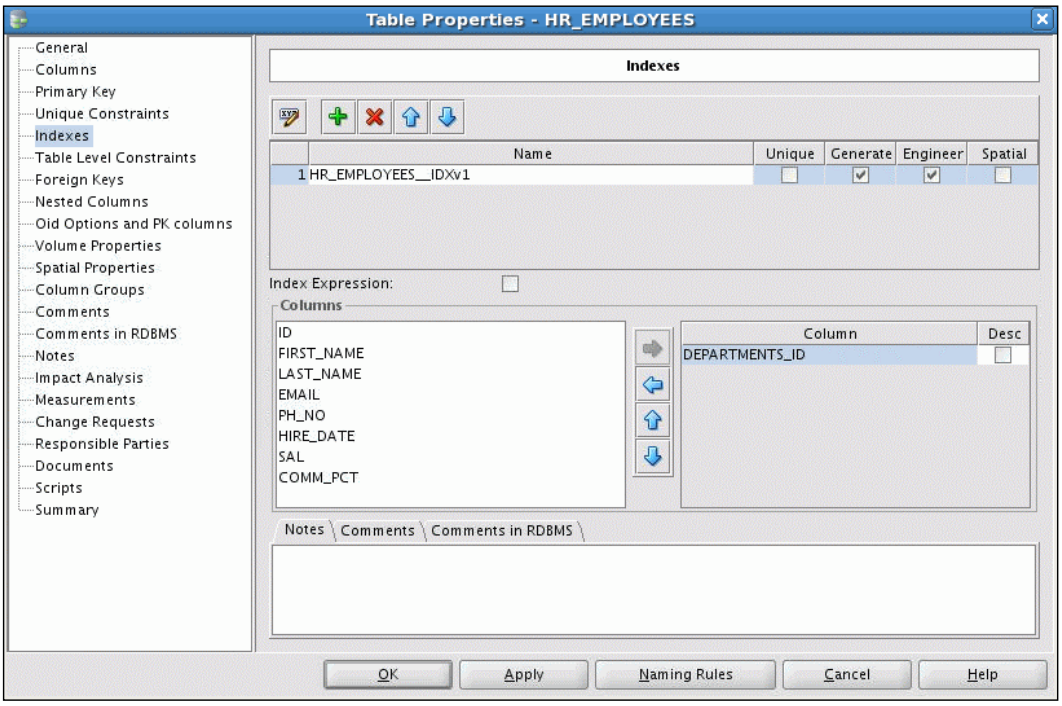
ORACLE

Defining a Unique Constraint

A unique constraint can be defined on a column that is unique but not part of the primary key. To define a unique constraint, perform the following steps:

1. Select Unique Constraint in the left navigator in the Table Properties window.
2. Click the Create icon and specify a name for the constraint. Notice that the default name is based on the Naming Standard > Templates in the General options.
3. Select the columns in the Columns list, and click the right arrow to move them to the Column list. Click OK.

Defining Indexes



ORACLE

Defining Indexes

Indexes are database structures that are stored separately from the tables that they reference. Indexes provide quick access to rows of data and help to avoid full table scans. Indexes facilitate table joins, and ensure the uniqueness of a value if it is defined as unique. Indexes are used automatically when referenced in the WHERE clause of a query if the column is not modified.

Types of indexes:

- **B-Tree:** An ordered list of values divided into ranges. By associating a key with a row or range of rows, B-trees provide excellent retrieval performance for a wide range of queries, including exact match and range searches. The example in the slide shows an index on the department_id column, which is a foreign key column in the employees table. B-tree indexes have the following subtypes: index-organized tables, reverse key indexes, descending indexes, or B-tree cluster indexes. Use B-Tree indexes for:
  - Columns used in join conditions to improve performance on joins
  - Columns that contain a wide range of values
  - Columns that are often used in the WHERE clause of a query
  - Columns that are often used in an ORDER BY clause of a query

Defining Indexes (continued)

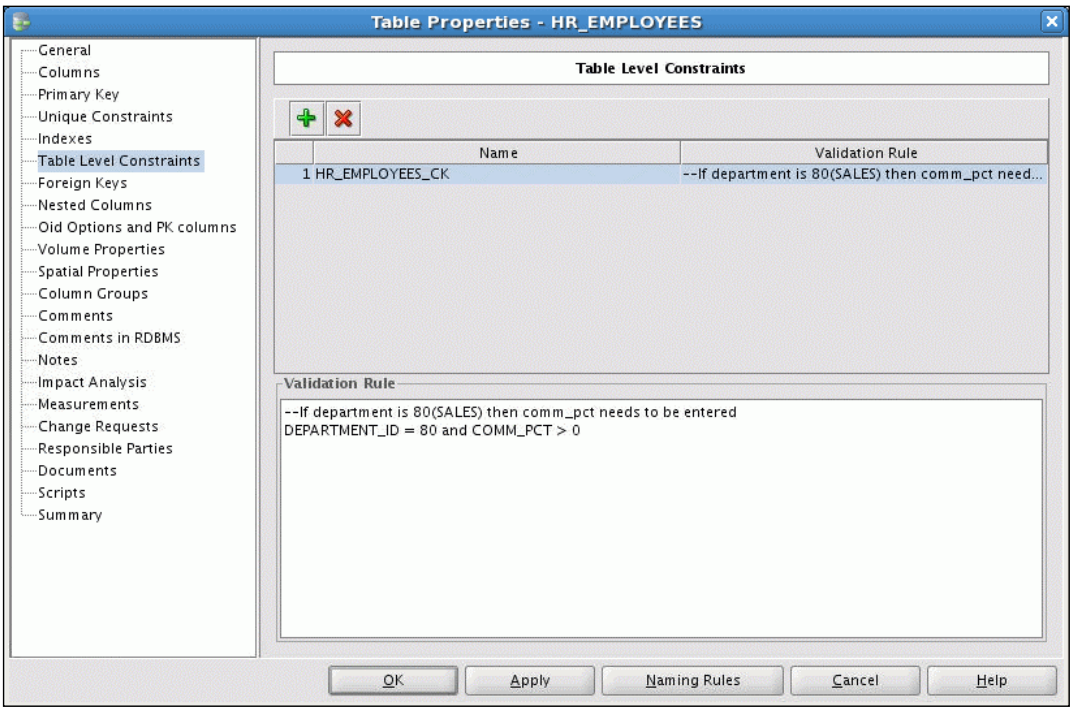
- **Bitmap:** An index entry uses a bitmap to point to multiple rows. In contrast, a B-tree index entry points to a single row. A bitmap join index is a bitmap index for the join of two or more tables. In a bitmap index, the database stores a bitmap for each index key. In a conventional B-tree index, one index entry points to a single row. In a bitmap index, each index key stores pointers to multiple rows. Bitmap indexes are primarily designed for data warehousing or environments in which queries reference many columns in an ad hoc fashion. Use bitmap indexing when columns have few distinct values (for example, when a column containing indicator values (Y/N), or when a column is used for gender).
- **Function Based:** Includes columns that are either transformed by a function, such as the UPPER function, or included in an expression. The function used for building the index can be an arithmetic expression or an expression that contains a SQL function, user-defined PL/SQL function, package function, or C callout. For example, a function could add the values in two columns. B-tree or bitmap indexes can be function-based.

Less suitable for indexing are columns that contain many NULL values where you usually search rows with the NULL values.

Columns that cannot or should not be indexed include:

- LONG and LONG RAW columns
- Columns that are rarely used in WHERE or ORDER BY clauses
- Small tables occupying only a few data blocks

Defining a Table-Level Constraint

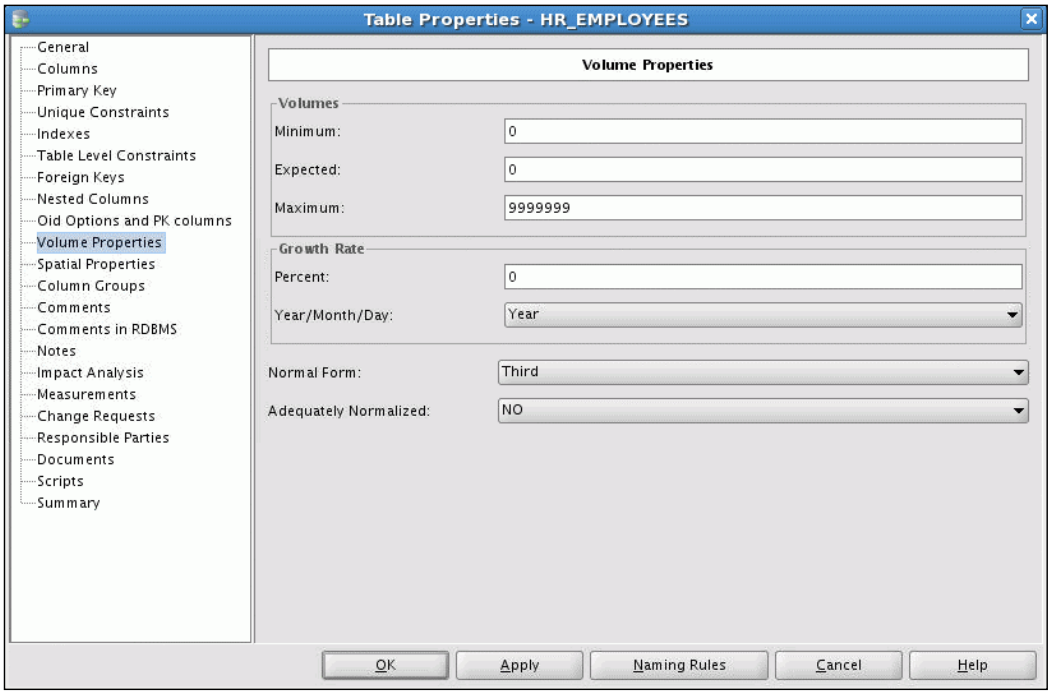


ORACLE

Defining a Table-Level Constraint

A table-level constraint defines various validation rules for a table. Typically a table-level constraint would involve a validation of more than one column in the table. In the example in the slide, a table-level constraint has been defined to validate that COMM_PCT is not null when the DEPARTMENT_ID is equal to 80 (which is for the Sales department). Other examples of table-level constraints: to ensure that the area code of a phone number is valid for a particular state, or to ensure that the country code is valid for a particular country.

Specifying Volume Properties

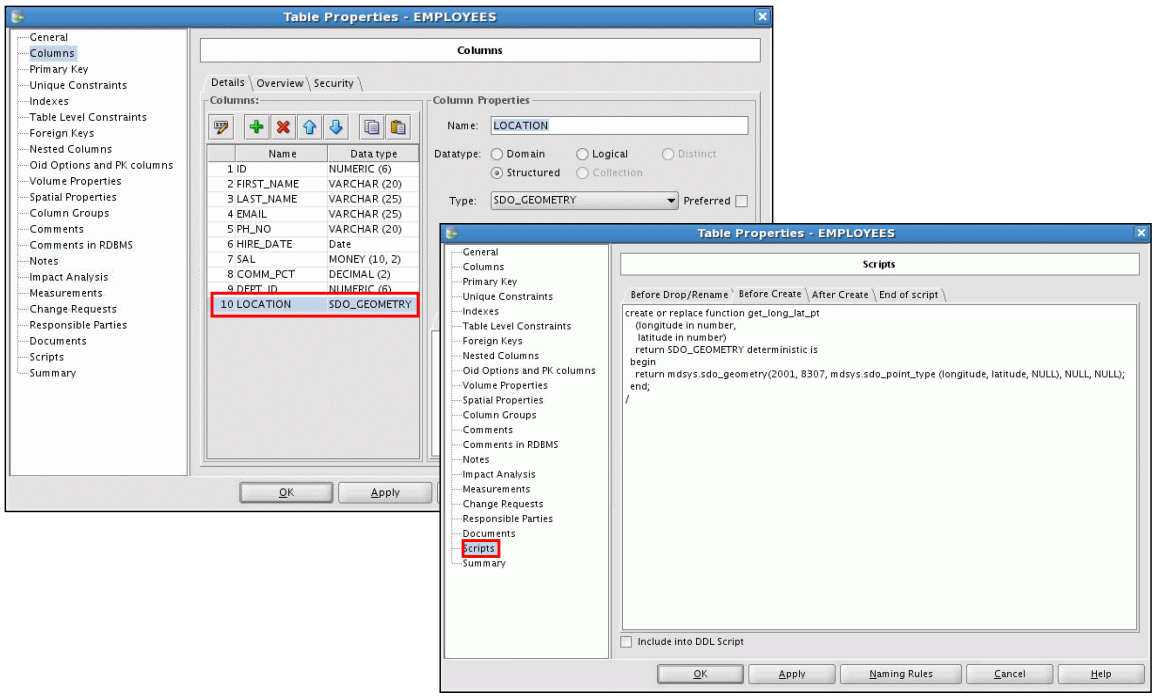


Specifying Volume Properties

Defining the volume information for a table is important later when it comes to determining the size of a database. This information is captured in the relational model. Select Volume Properties in the left navigator for the table.

- **Volume Minimum:** Minimum data volume for the table
- **Volume Expected:** Expected or typical data volume for the table
- **Volume Maximum:** Maximum data volume for the table
- **Growth Rate Percent:** Expected growth rate percentage for the table, for each period as specified in the next field
- **Growth Rate Year/Month/Day:** The period (year, month, or day) to which the expected growth rate applies
- **Normal Form:** The required normal form (database normalization) for the table: None, First, Second, Third, or Fourth
- **Adequately Normalized:** YES indicates that the model is sufficiently normalized. NO indicates that the model is not sufficiently normalized, and that additional normalization may be required on the relational model.

Defining Spatial Properties



ORACLE

Defining Spatial Properties

Oracle Spatial is an integrated set of functions and procedures that enables spatial data to be stored, accessed, and analyzed quickly and efficiently in an Oracle database.

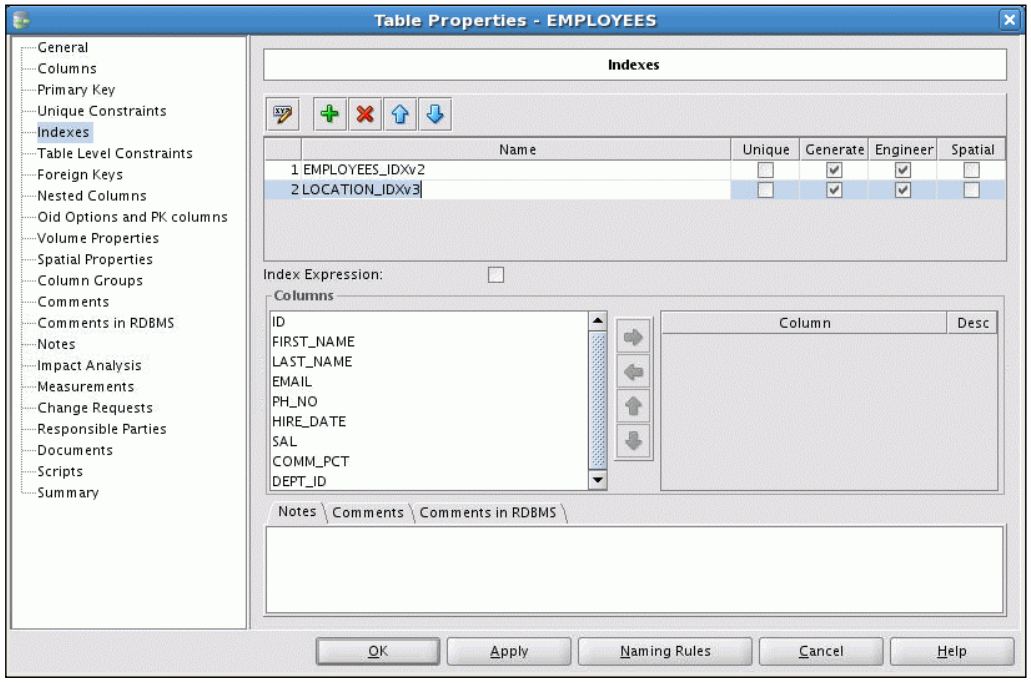
Spatial data represents the essential location characteristics of real or conceptual objects as those objects relate to the real or conceptual space in which they exist.

Spatial supports the object-relational model for representing geometries. This model stores an entire geometry in the Oracle native spatial data type for vector data, SDO_GEOMETRY. An Oracle table can contain one or more SDO_GEOMETRY columns.

Currently defined Oracle Spatial properties are displayed in Spatial properties, each being a data column (type SDO_GEOMETRY or a function that evaluates to an SDO_GEOMETRY object) in the table.

In the example in the slide, a new column called LOCATION is created that is assigned the spatial data type SDO_GEOMETRY. A script is also added for the table that will create a function to get the longitude and latitude points.

Defining Spatial Properties

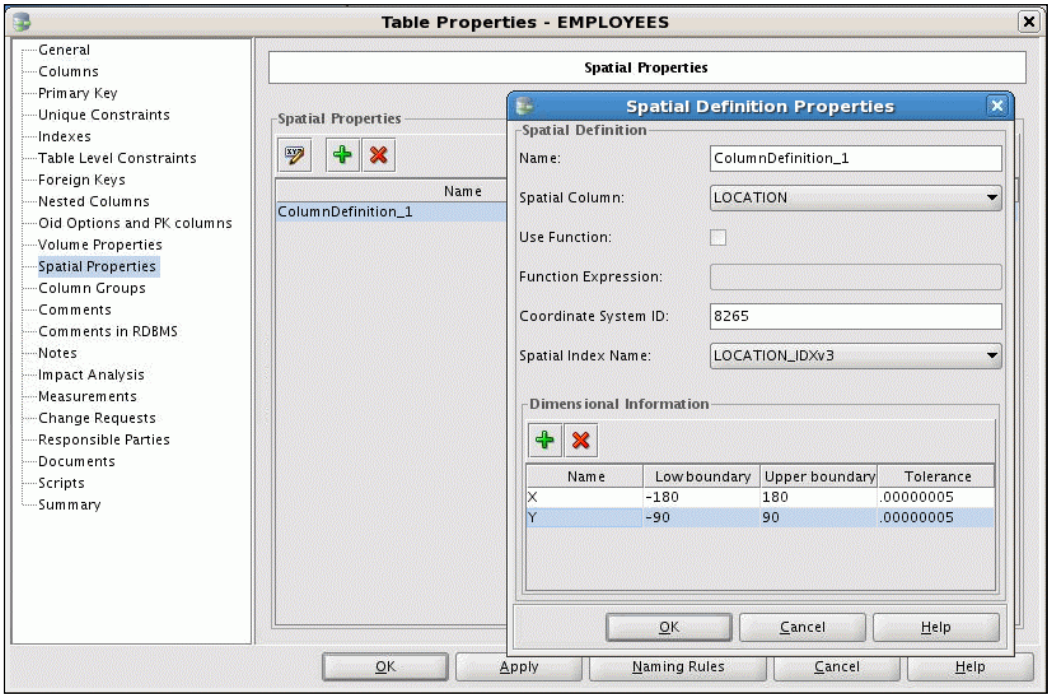


ORACLE

Defining Spatial Properties (continued)

If you want to base your spatial column on an index, create the index as you would any other index. In the example in the slide, you create an index and change the name to LOCATION so that you know which index should be used for spatial purposes.

Defining Spatial Properties



ORACLE

Defining Spatial Properties (continued)

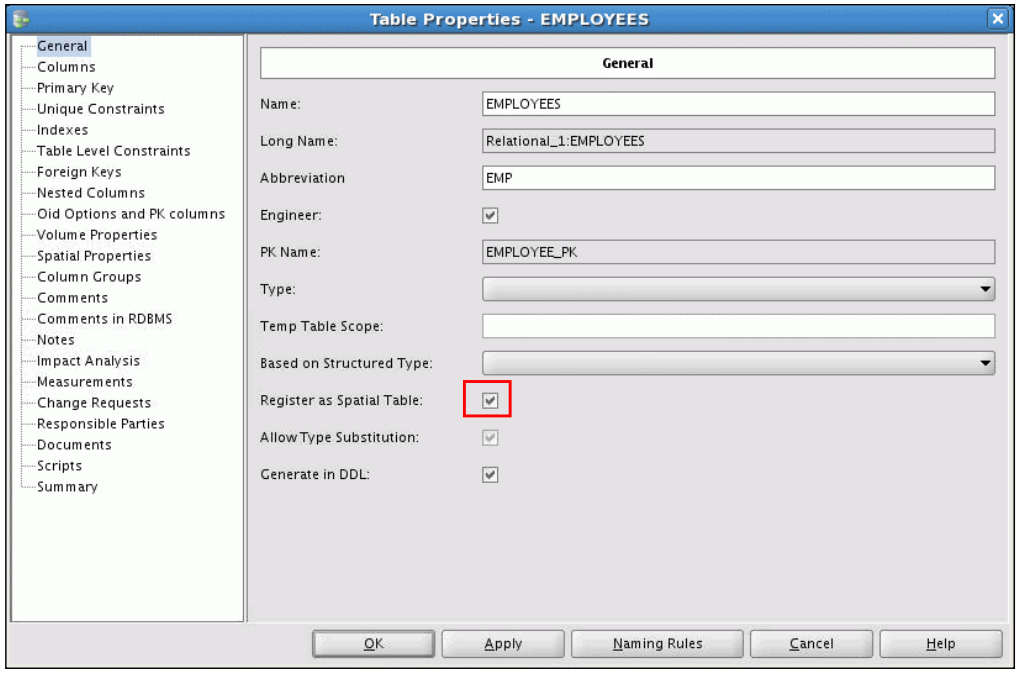
After the column is defined, add a spatial definition by performing the following steps:

1. Select Spatial Properties in the object browser.
2. Click the Add '+' icon.
3. Specify the name of the definition.
4. Base this definition on a column or a function. In the example in the slide, the LOCATION column is selected from the Spatial Column drop-down list.
5. Select the index (created in the previous slide) in the Spatial Index Name drop-down list.
6. Specify the Coordinate System ID and create the dimension information for this spatial column. In the example on this slide, two dimensions were created to add an entry in the metadata view.

Note: The DDL that will be generated is:

```
INSERT INTO USER_SDO_GEOM_METADATA ( TABLE_NAME , COLUMN_NAME , DIMINFO , SRID )
    VALUES ( 'EMPLOYEES', 'LOCATION' ,
        MDSYS.SDO_DIM_ARRAY (
            MDSYS.SDO_DIM_ELEMENT ('X',-180,180,.00000005),
            MDSYS.SDO_DIM_ELEMENT ('Y',-90,90,.00000005)),
        8265
    );
```

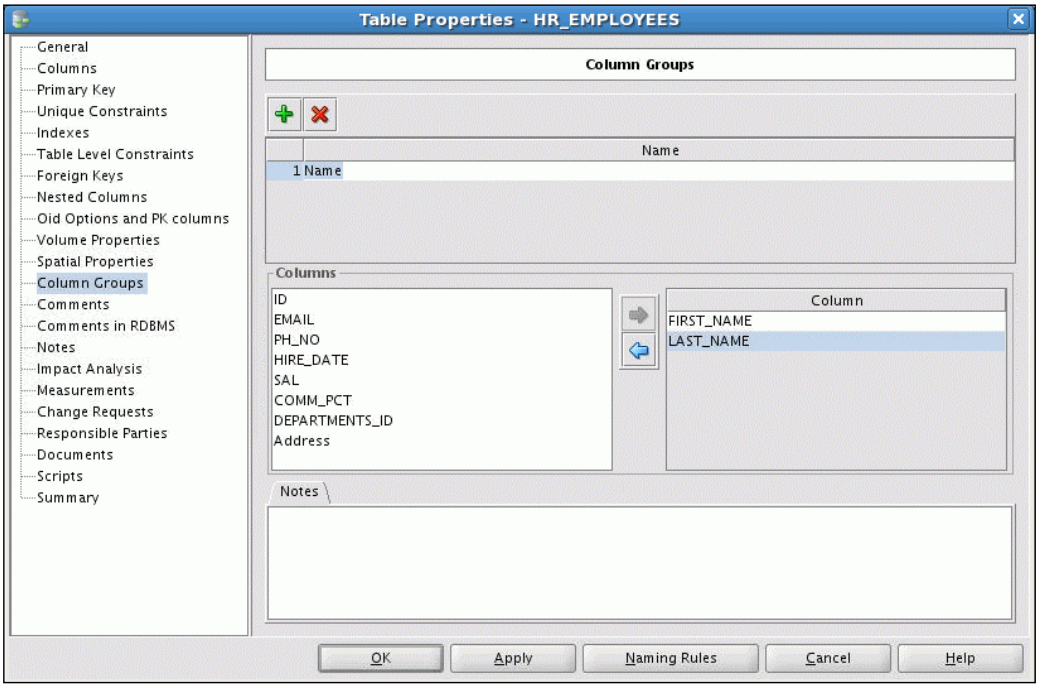
Defining Spatial Properties



Defining Spatial Properties (continued)

To generate the Spatial-specific DDL for this table, select the “Register as Spatial Table” property.

Defining Column Groups



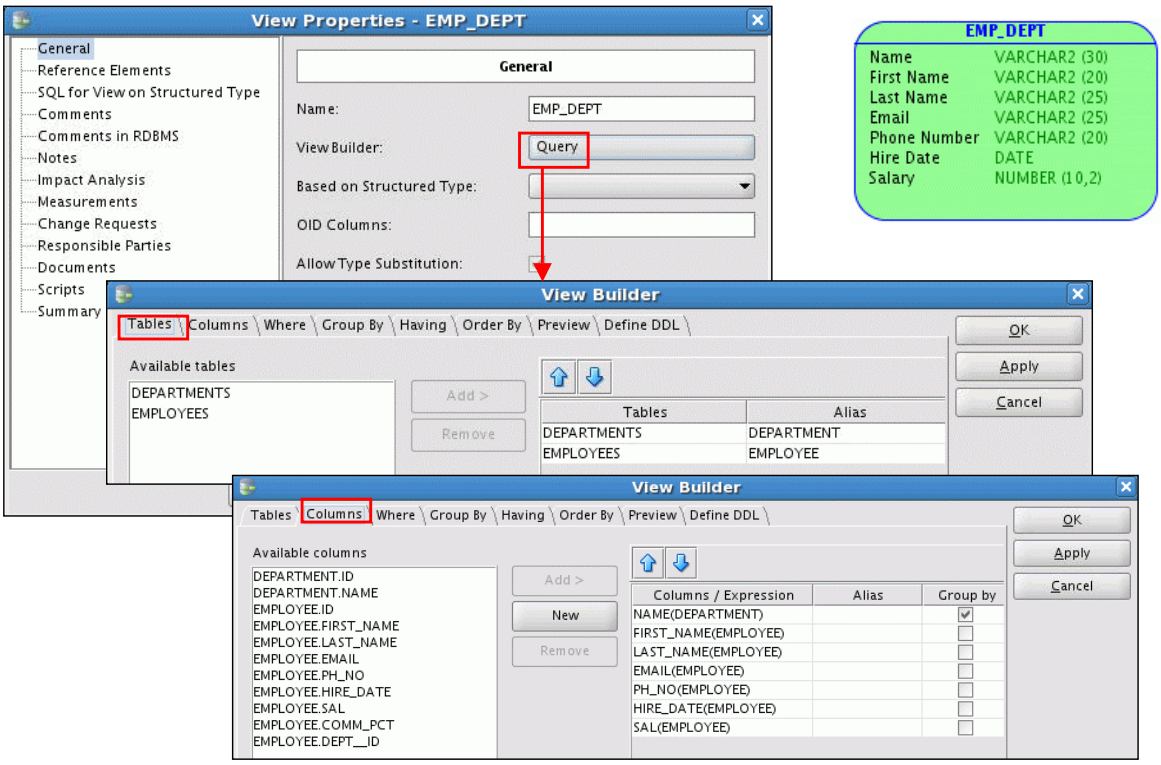
ORACLE

Defining Column Groups

Column groups allow you to group related columns for possible use in generating a user interface. For example, a column group named Name could include first_name and last_name columns, and a column group named Address could include street_address, city, state, and postal_code columns.

To add a column group, click the Add (+) icon, specify the column group name, and then select the desired columns in the Columns list and move them to the Column list. Optionally, enter descriptive text in the Notes box. To delete a column group, select its entry and click the Remove (X) icon.

Analyzing Your View



Analyzing Your View

A view is a window into your database. It is defined by a SELECT statement that is named and stored in the database. A view has no data of its own, it only relays information from underlying tables.

You can use a view to accomplish the following:

- Restrict access to a set of columns or rows in a table.
- Present data in a more understandable way. For example, a view can present calculated data built from basic information that is stored in tables.
- Isolate applications based on views rather than tables. If a view is used, the application would need no maintenance if the underlying tables changed.
- Save complex queries and simplify commands to allow users to create queries over multiple tables without having to know how to join tables together.

Advantages include:

- Presenting derived data to end users without having to store this subset of data in the database
- Showing data that depends on which user you are or what day it is. For example, you can only access a view during certain hours of the day.

Analyzing Your View (continued)

Disadvantages include:

- Views are somewhat slower, because the parse time is slightly longer. After a table and its columns are found, the query can be immediately executed.
- You cannot insert, update, or delete directly into a view.

Quiz

You have a table that is very large and you want to increase query access performance. Which of the following would you do to handle this requirement?

a. Add a table constraint
b. Create a view
c. Create an index
d. Set the volume properties very high

Answer: c

Quiz

In order to make it easier to select data from multiple tables, which of the following would you do?

a. Create an index
b. Add a column group
c. Add a unique constraint
d. Create a view

ORACLE

Answer: d

Summary

In this lesson, you should have learned how to:

- Modify table properties according to requirements
- Determine when to create an index
- Determine when to create a view

ORACLE

Summary

In this lesson, you should have learned how to modify table properties and when to create an index or view.

Practice 16-1 Overview: Analyze Your Relational Model

This practice covers the following topics:

- Evaluating your relational model
- Adding additional design components based on a set of requirements.

ORACLE

Practice 16-1 Overview: Analyze Your Relational Model

In this practice, you evaluate a relational model and add additional design components based on a set of requirements.

17

Denormalizing Your Design to Increase Performance

Objectives

After completing this lesson, you should be able to:

- Recognize when denormalization techniques can be used in your relational model

Objectives

In this lesson, you learn some denormalization techniques that can optimize your relational model.

What Is Denormalization?

- Denormalization:
  - Is the process of adding redundancy to the data to improve performance
  - Starts with a “normalized” model
  - Adds “redundancy” to the design
  - Reduces the “integrity” of the design
- Application code is added to compensate.

ORACLE

What Is Denormalization?

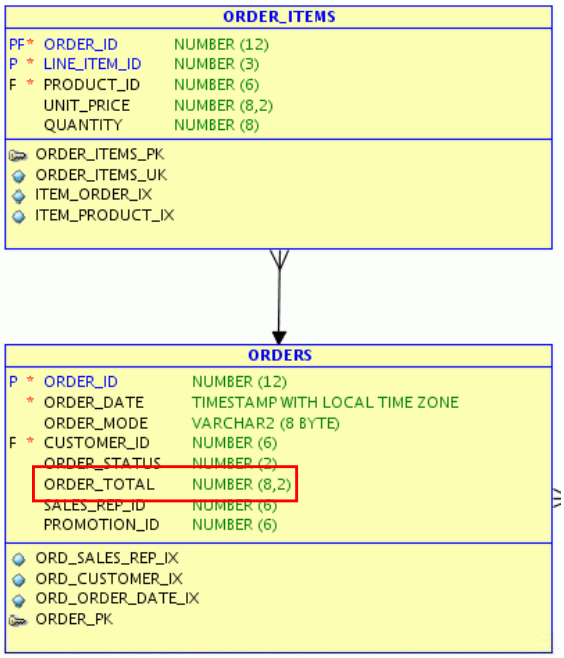
Denormalization aids the process of systematically adding redundancy to the database to improve performance after other possibilities, such as indexing, have failed. Denormalization can improve certain types of data access dramatically, but there is no success guaranteed and there is always a cost. The data model becomes less robust, and it will always slow Data Manipulation Language (DML) down. It complicates processing and introduces the possibility of data integrity problems. It always requires additional programming to maintain the denormalized data.

Hints for Denormalizing

- Always create a conceptual data model that is completely normalized.
- Consider denormalization as the last option to boost performance. Never presume that denormalization is required.
- Do denormalization during the database design.
- After performance objectives have been met, do not implement any further denormalization.
- Fully document all denormalization, stating what was done to the tables, and what application code was added to compensate for the denormalization.

Storing Derivable Values

Add the ORDER_TOTAL column to the ORDERS table to store the derived value of an order.



Storing Derivable Values

When a calculation is frequently executed during queries, it can be worthwhile storing the results of the calculation. If the calculation involves detail records, store the derived calculation in the master table. Be sure to write application code to recalculate the value each time that DML is executed against the detail records. In all situations of storing derivable values, make sure that the denormalized values cannot be directly updated. They should always be recalculated by the system.

This denormalizing technique is appropriate when:

- The source values are in multiple records or tables.
- Derivable values are frequently needed and when the source values are not.
- The source values are infrequently changed.

Advantages for using this technique include:

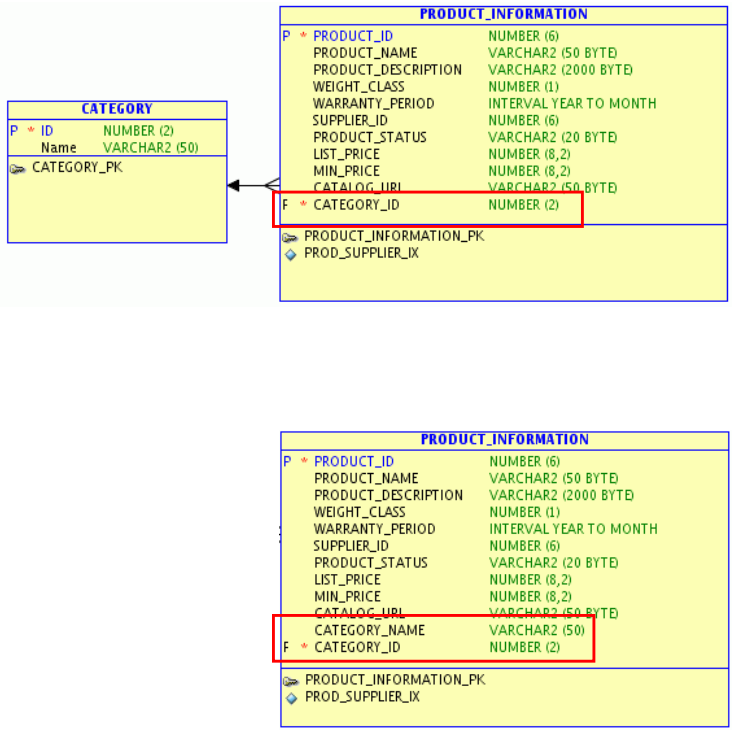
- Source values do not need to be looked up every time the derivable value is required.
- The calculation does not need to be performed during a query or report.

Disadvantages for using this technique include:

- DML against the source data will require recalculation or adjustment of the derivable data.
- Data duplication introduces the possibility of data inconsistencies.

Pre-Joining Tables

Pre-join the PRODUCT_INFORMATION and CATEGORY tables by adding the CATEGORY_NAME column to the PRODUCT_INFORMATION table and deleting the CATEGORY table.



Pre-Joining Tables

You can pre-join tables by including a nonkey column in a table, when the actual value of the primary key, and consequentially the foreign key, has no business meaning. By including a nonkey column that has business meaning, you can avoid joining tables, thus speeding up specific queries.

You must include application code that updates the denormalized column each time that the "master" column value changes in the referenced record.

This denormalizing technique is appropriate when:

- Frequent queries against many tables are required.
- Slightly stale data is acceptable.

Advantages for using this technique include:

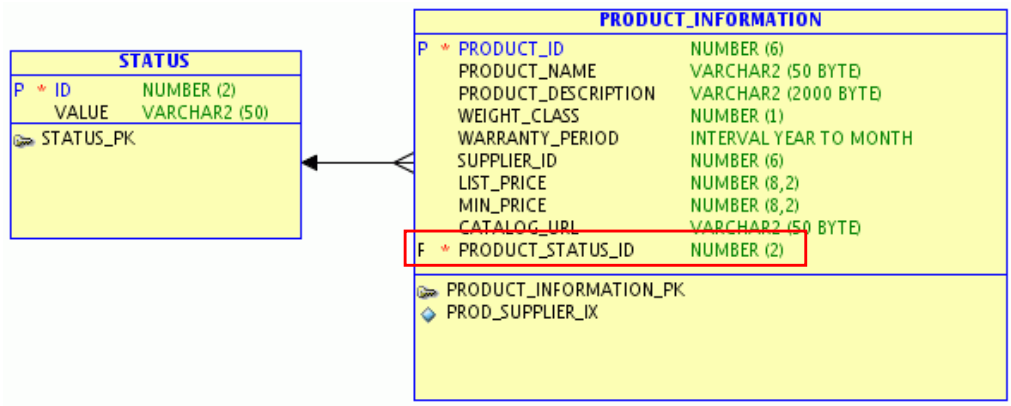
- Time-consuming joins can be avoided.
- Updates may be postponed when stale data is acceptable.

Disadvantages for using this technique include:

- Extra DML is needed to update the original nondenormalized column.
- Extra columns and possibly larger indexes require more working and disk space.

Hard-Coded Values

Remove the foreign key and hard code the allowable values in a check constraint.



ORACLE

Hard-Coded Values

If a reference or lookup table contains records that remain constant, you can consider hard-coding those values into the application code. This will mean that you will not need to join tables to retrieve the list or reference values. This is a special type of denormalization, when values are kept outside the table in the database. In the example in the slide, the PRODUCT_STATUS column contains values that are constant. In this case, you can create a check constraint on the PRODUCT_STATUS column that will validate the values. Note that a check constraint, though it resides in the database, is still a form of hard coding. Whenever a new value of PRODUCT_STATUS is needed the constraint must be modified.

This denormalizing technique is appropriate when:

- The set of allowable values can reasonably be considered to be static during the life cycle of the system
- The set of possible values is small, perhaps less than 30.

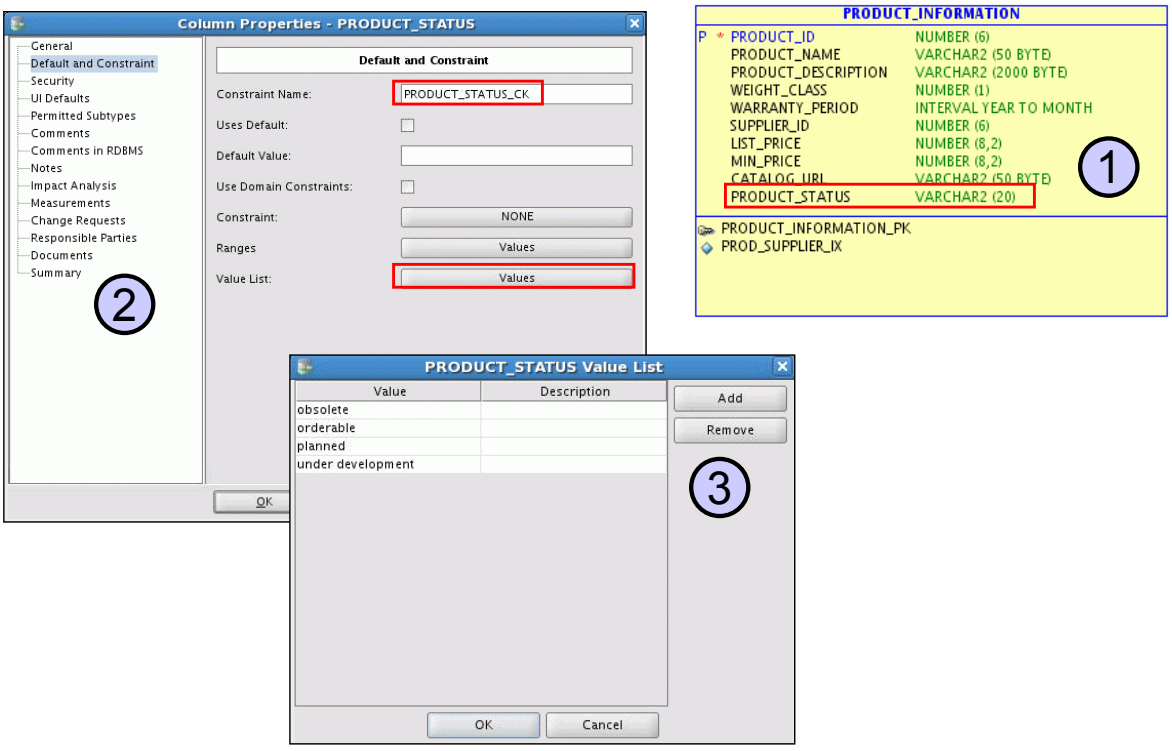
Advantages for using this technique include:

- Implementing a look-up table is not necessary.
- Joins to a look-up table are not necessary.

Disadvantages for using this technique include:

- Changing look-up values requires recoding and retesting.

Hard-Coded Values

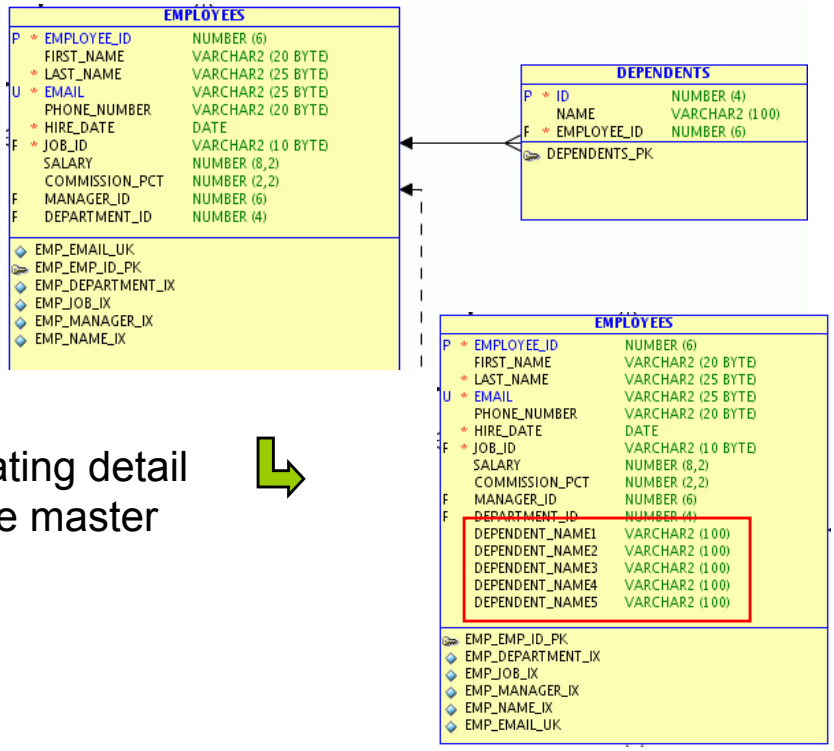


Hard-Coded Values (continued)

Continuing with the example from the previous slide, you delete the foreign key with the STATUS table and create a PRODUCT_STATUS column in the PRODUCT_INFORMATION table. To define the check constraint for the PRODUCT_STATUS column, perform the following steps:

1. Double-click the table, select Columns in the left navigator, and double-click the PRODUCT_STATUS column.
2. Select “Default and Constraints” in the left navigator. Enter a constraint name and click the Value list button.
3. Click Add to add each value to the list.

Keeping Details with the Master Table



Add the repeating detail columns to the master table.

Keeping Details with the Master Table

In a situation where the number of detail records per master is a fixed value (or has a fixed maximum) and where usually all detail records are queried with the master, you may consider adding the detail columns to the master table. This denormalization technique works best when the number of records in the detail table is small. This way you will reduce the number of joins during queries.

This denormalizing technique is appropriate when:

- The number of detail records for all masters is fixed and static.
- The number of detail records multiplied by the number of columns of the detail is small, perhaps less than 30.

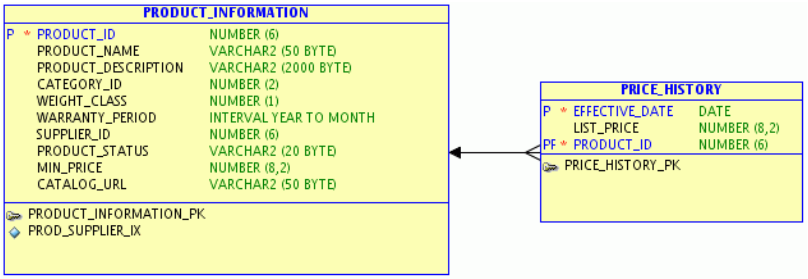
Advantages for using this technique include:

- No joins are required.
- Saves space, because keys are not propagated.

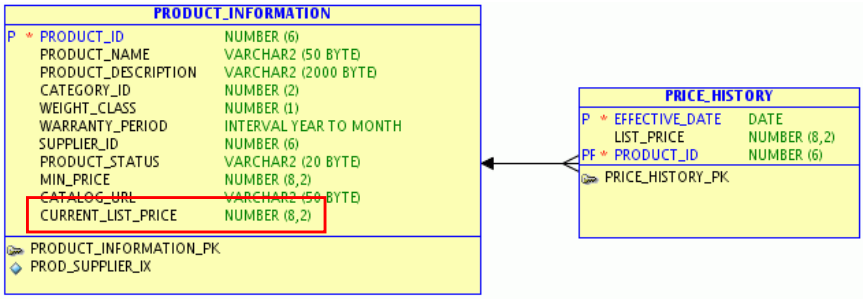
Disadvantages for using this technique include:

- Increases complexity of DML and SELECTs across detail values.
- Checks must be repeated for each repeating column.

Repeating Current Detail with the Master Table



Add a column to the master to store the most current details.



ORACLE

Repeating Current Detail with the Master Table

Often when the storage of historical data is necessary, many queries required only the most current record. You can add a new foreign key column to store this single detail with it master. Make sure you add code to change the denormalized column any time a new record is added to the history table. Additional code must be written to maintain the duplicated single detail value for the master record. In the example in the slide, the price for a product is maintained in the `PRICE_HISTORY` table by the `EFFECTIVE_DATE` column. The `CURRENT_LIST_PRICE` column was added to the `PRODUCT_INFORMATION` table to store the current detail information.

This denormalizing technique is appropriate when:

- Detail records per master have a property such that one record can be considered “current” and others “historical.”
- Queries frequently need this specific single detail, and only occasionally need the other details.
- The Master often has only one single detail record.

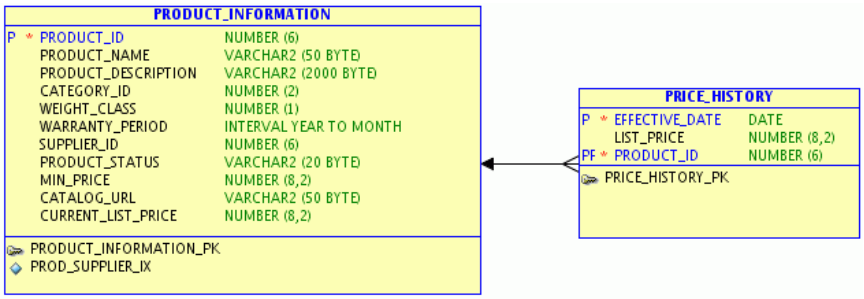
Advantages for using this technique include:

- No join is required for queries that only need the specific single detail.

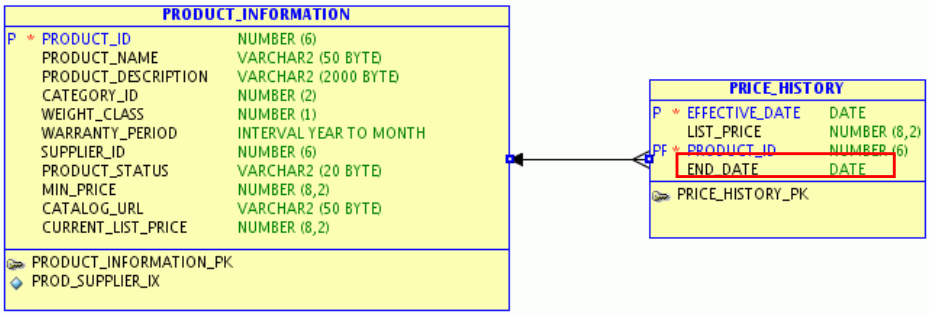
Disadvantages for using this technique include:

- Detail value must be repeated, with the possibility of data inconsistencies.

End Date Columns



Add an end date column so that the between operator can be used.



ORACLE

End Date Columns

One of the most common denormalization techniques is to store the end date for periods that are consecutive. As a result, the end date for a period can be derived from the start date of the previous period. Using this technique you can find a detail record for a particular date without using a complex query. In the example in the slide, the END_DATE column was added to the PRICE_HISTORY table so that if you want to check the price between a certain set of dates you could create a query using the between clause.

This denormalizing technique is appropriate when:

- Queries are needed from tables with long lists or records that are historical and you are interested in the most current record.

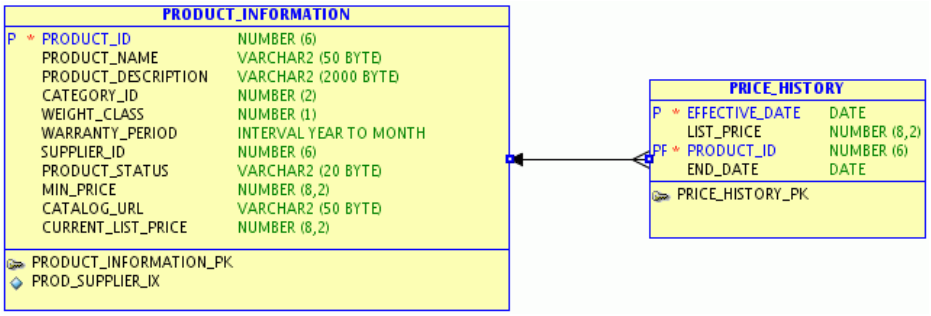
Advantages for using this technique include:

- Using the between operator for date selection queries instead of potentially time-consuming synchronizing subquery.

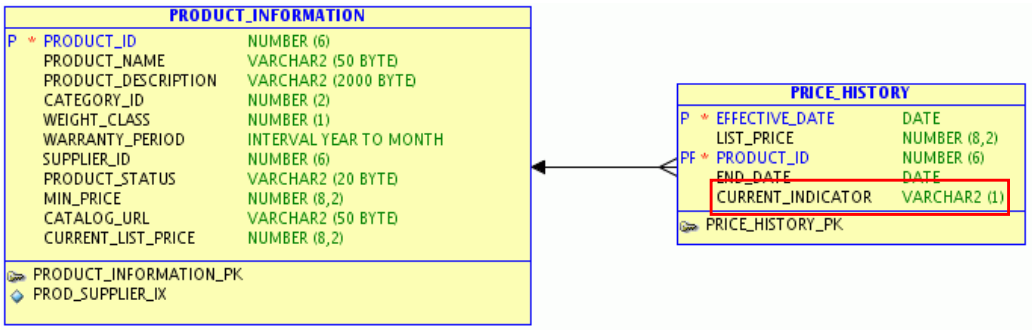
Disadvantages for using this technique include:

- Extra code is needed to populate the end date column with the value found in the previous start date record.

Current Indicator Column



Add a column to represent the most current record in a list of records.



ORACLE

Current Indicator Column

This denormalization technique allows you to quickly find the most current detail record by adding a new indicator column to the details table to represent the currently active row. You would need to add code to update the indicator column each time that you insert a new record. In the example in the slide, the `CURRENT_INDICATOR` column was added to the `PRICE_HISTORY` table to allow you to query the currently active row more easily.

This denormalizing technique is appropriate when:

- The situation requires retrieving the most current record from a long list.

Advantages for using this technique include:

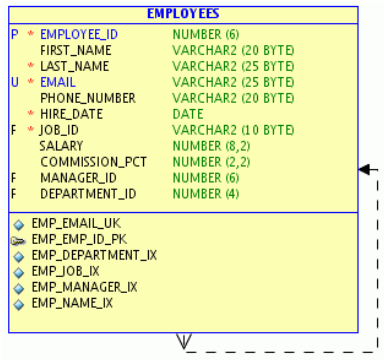
- Queries and subqueries are less complicated.

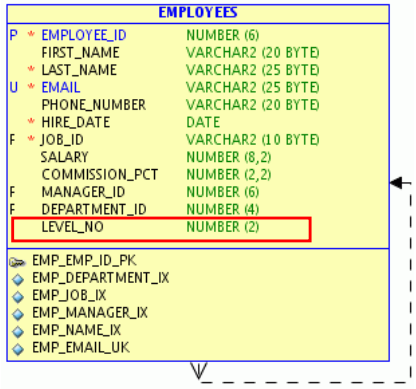
Disadvantages for using this technique include:

- Extra column and application code is needed to maintain the column.
- The concept of “current” makes it impossible to make data adjustments ahead of time.

Hierarchy Level Indicator

Create a column to represent the hierarchy level of a record.





ORACLE

Hierarchy Level Indicator

Suppose that there is a business limit to the number of levels that a particular hierarchy may contain. Or suppose that in many situations you need to know which records have the same level in a hierarchy. In both these situations, you need to use a connect-by clause to traverse the hierarchy. This type of clause can be costly to performance. You could add a column to represent the level of a record in the hierarchy, and then just use the value instead of the connect-by clause in SQL.

This denormalizing technique is appropriate when:

- There are limits to the number of levels within a hierarchy, and you do not want to use a connect-by search to see if the limit has been reached.
- You want to find records located at the same level in the hierarchy.
- The level value is often used for particular business reasons.

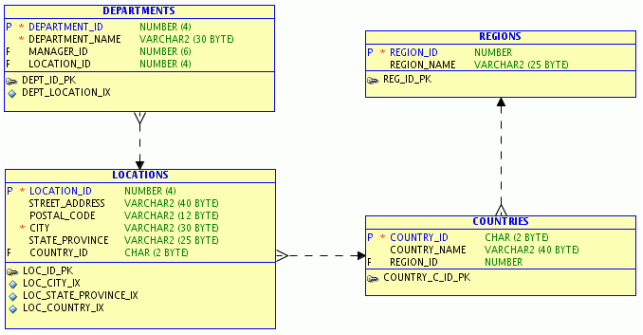
Advantages for using this technique include:

- No need to use the connect-by clause in your query.

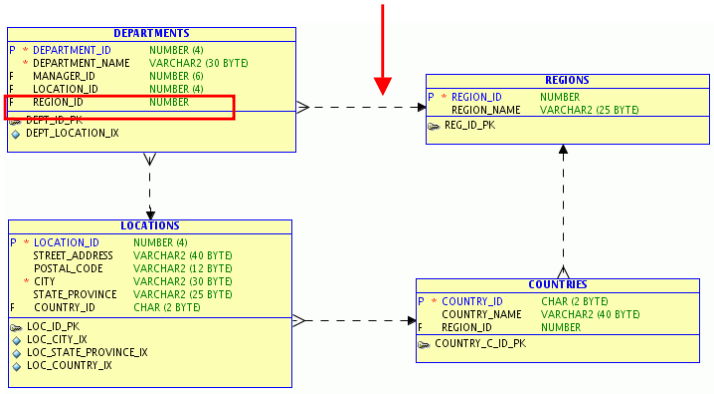
Disadvantages for using this technique include:

- Each time a foreign key is updated, the level indicator needs to be recalculated, and you may need to cascade the changes.

Short Circuit Keys



Create a new foreign key from the lowest detail to the highest master.



ORACLE

Short Circuit Keys

For database designs that contain three (or more) levels of master detail where there is a need to query only the lowest and highest level records, consider creating a short circuit key. These new foreign key definitions directly link the lowest level detail records to higher level grandparent records. The result can produce fewer table joins when queries execute. In the example in the slide, you frequently want to know which region a particular department is in. As a result, you would create a foreign key between the `DEPARTMENTS` and `REGIONS` table.

This denormalizing technique is appropriate when:

- Queries frequently require values from a grandparent and grandchild, but not from the parent.

Advantages for using this technique include:

- Queries join fewer tables together.

Disadvantages for using this technique include:

- Extra foreign keys are required.
- Extra code is required to make sure that the value of the denormalized column is consistent with the value that you would find after a join.

Quiz

You have a column that has a small set of static values. Currently you have a table that contains the list of values with a foreign key to the table that needs to use the values. What can you do to increase performance?

a. Keep all the values in the table that needs to use the values as separate columns.
b. Create a check constraint on the column that uses the set of values.
c. Add a short-circuit key to access the set of values more readily.
d. Create a table with all the columns so that you can access all the data without doing a join.

ORACLE

Answer: b

Quiz

Your database design contains the COURSES and COURSE_HISTORY tables to show the list of current courses as well as those offered in the past. In order to find the currently offered courses, what denormalization techniques could you use?

a. Add a current indicator column to the COURSE_HISTORY table with a bitmap index.
b. Add a check constraint to the COURSE_TYPE column in the COURSES table.
c. Add START_DATE and END_DATE columns to the COURSE_HISTORY table so that you can use the between operator in your query.
d. Add a LEVEL_NO column to the COURSES table.

ORACLE

Answer: a

Summary

In this lesson, you should have learned how to:

- Recognize when denormalization techniques can be used in your relational model

ORACLE

Summary

In this lesson, you should have learned some denormalization techniques and when to use them.

Practice 17-1 Overview: Denormalize Your Relational Model

This practice covers the following topics:

- Evaluating a relational model and denormalizing it based on a set of requirements

ORACLE

Practice 17-1 Overview: Denormalize Your Relational Model

In this practice, you evaluate a relational model and denormalize it based on a set of requirements.

18

Defining Your Physical Model

Objectives

After completing this lesson, you should be able to:

- Create objects in a physical model
- Refine relational model objects in the physical model

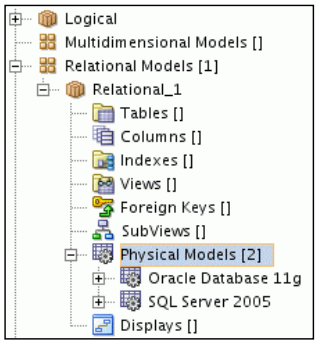
ORACLE

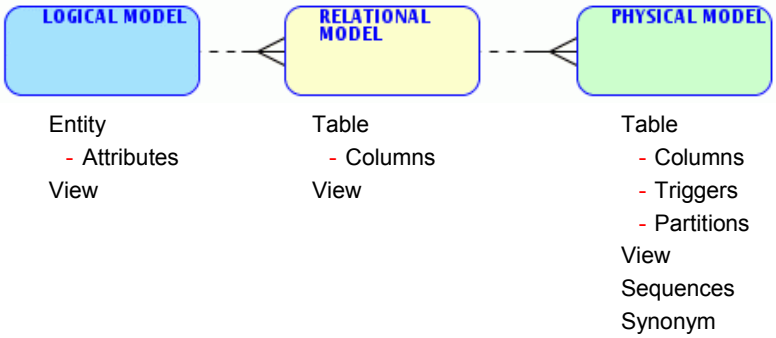
Objectives

In this lesson, you learn how to explain what a physical model is and how to create objects in the physical model.

What Is a Physical Model?

- A physical model:
  - Is an extension of a relational model that describes how the objects should be implemented in specific database
  - Includes database objects (tables, views, triggers) based on a relational model
- A relational model can have many physical models.





ORACLE

What Is a Physical Model?

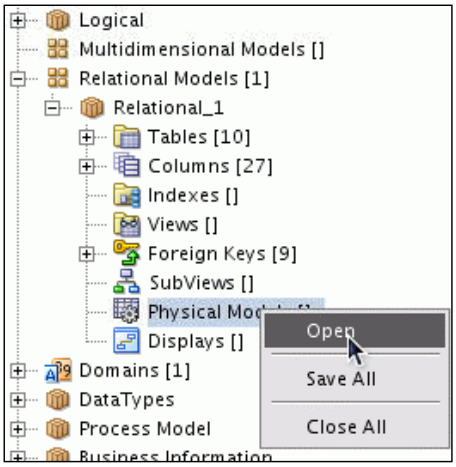
A physical model is an extension of a relational model that describes how the model should be implemented in specific databases. A physical model includes database objects (such as tables, views, and triggers) that are based on a relational model. Each relational model can have one or more physical models, one for each RDBMS that you are going to deploy to.

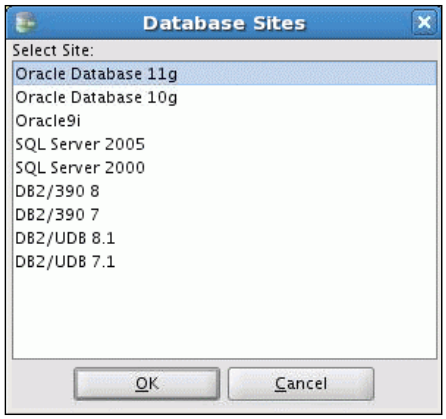
Physical models do not have graphical representation; instead, they are displayed in the object browser hierarchy. To create and manage objects in the physical model, use the Physical menu or the context (right-click) menu in the object browser.

You do not need a physical model to generate DDL, but the objects that you create in the physical model will be generated as well.

Creating a Physical Model

Use the object browser hierarchy to manage your physical model.



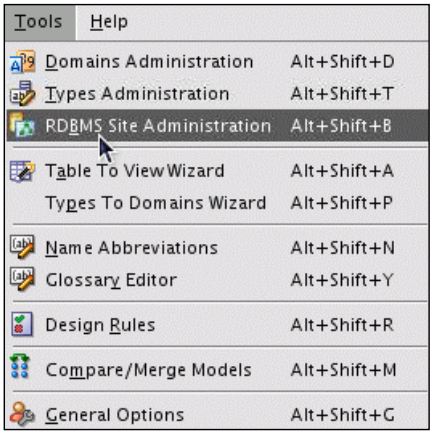


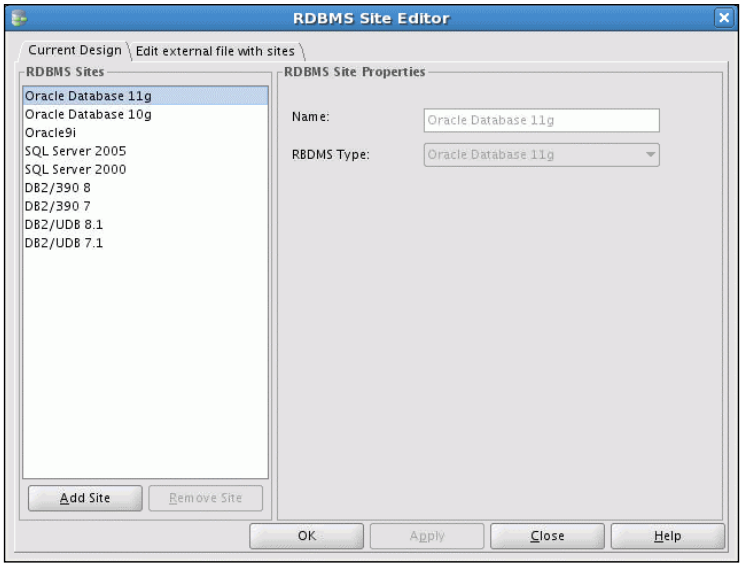
Creating a Physical Model

The physical model is managed from the object browser hierarchy. There is no graphic representation of it. To create a physical model, perform the following steps:

1. Expand your relational model (in this case, Relational_1), right-click Physical Models and select Open.
2. Select the database site that you want to create a physical model for, and click OK.

RDBMS Administration



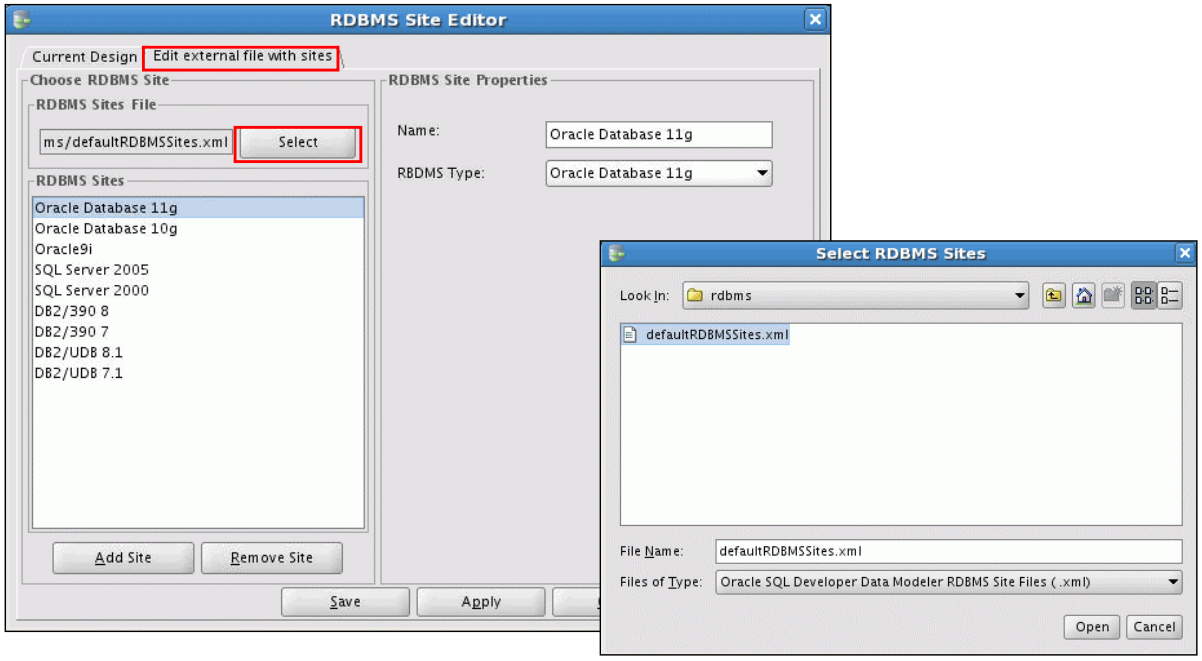


ORACLE

RDBMS Administration

Each physical model is based on an RDBMS site object. An RDBMS site is a name associated with a type of database supported by Oracle SQL Developer Data Modeler. Several RDBMS sites are predefined (for example, for Oracle Database 11*g* and Microsoft SQL Server 2005). You can also use the RDBMS Site Editor to create user-defined RDBMS sites as aliases for supported types of databases; for example, you might create sites named Test and Production, so that you will be able to generate different physical models and then modify them.

RDBMS Administration: Changing the Default RDBMS Sites



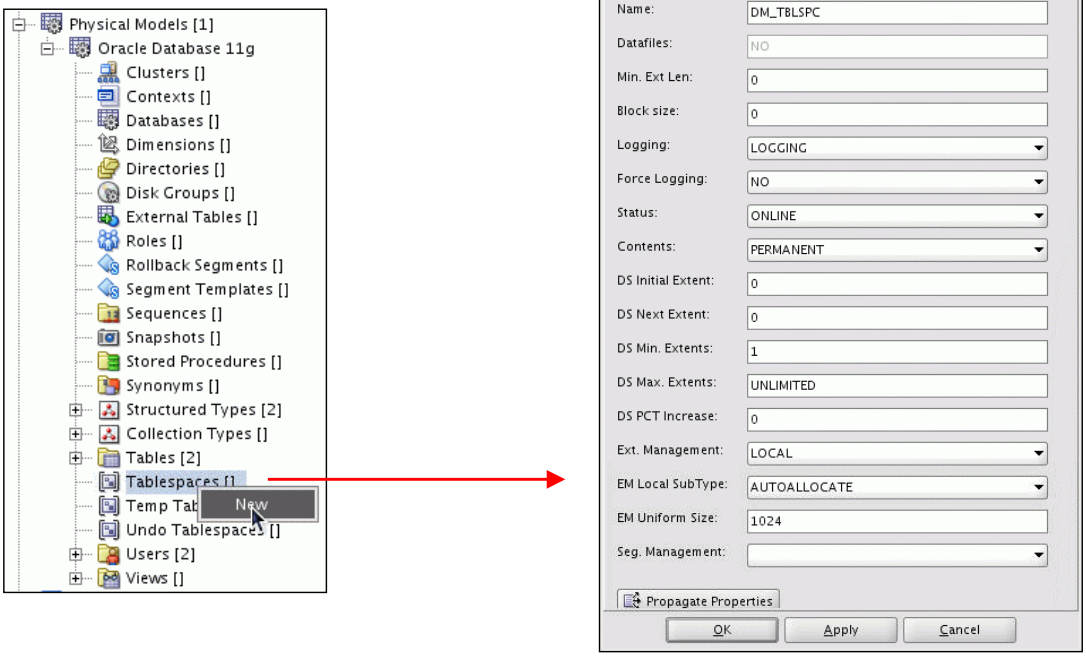
RDBMS Administration: Changing the Default RDBMS Sites

Several RDBMS sites are predefined (for example, for Oracle Database 11*g* and Microsoft SQL Server 2005). You can modify the predefined list by performing the following steps:

1. In the RDBMS Site Editor, click the “Edit external file with sites” tab.
2. Click the Select button.
3. Search for the `defaultRDBMSSites.xml` file in the `<datamodeler home>/RDMBS` directory and click Open.
4. You can modify or remove existing sites or add additional ones.

Creating Physical Model Objects

Physical Model Objects



Creating Physical Model Objects

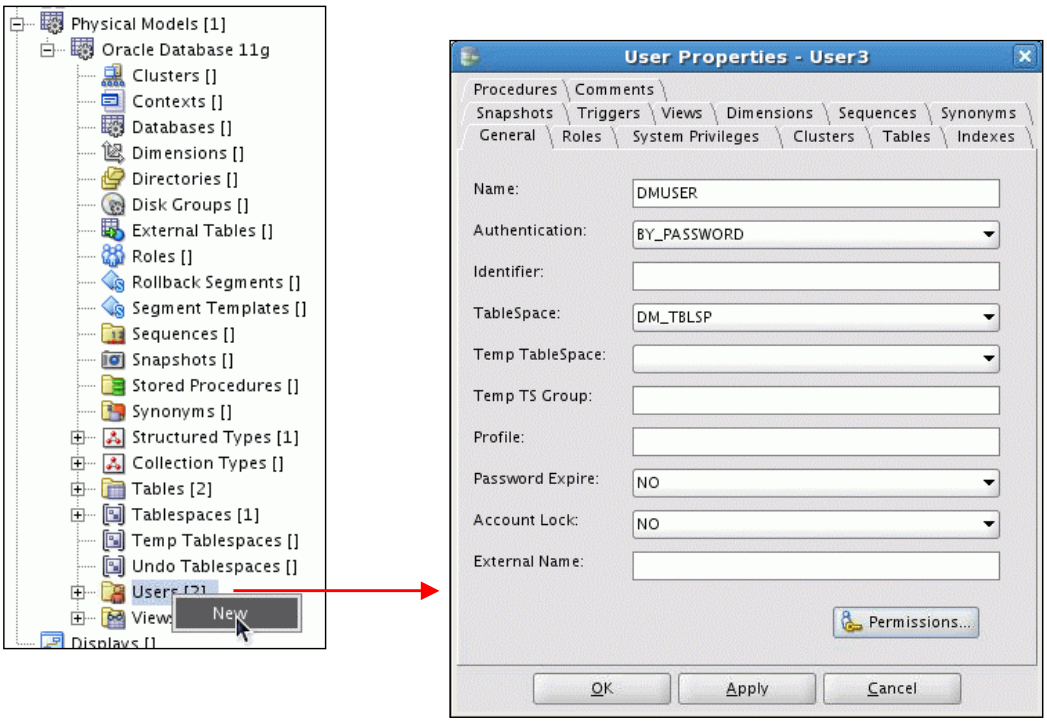
You can create many types of physical model objects. In addition, when you import from the data dictionary (discussed in a later lesson), the physical model objects in your database will be created in the physical model in Oracle SQL Developer Data Modeler. For an Oracle Database, the following objects are definable:

- **Clusters:** A schema object that contains data from one or more tables
- **Contexts:** A set of application-defined attributes that validates and secures an application
- **Databases:** A collection of data treated as a unit. The purpose of a database is to store and retrieve related information.
- **Dimensions:** A parent-child relationship between pairs of column sets, where all the columns of a column set must come from the same table. However, columns in one column set (called a level) can come from a different table than columns in another set. The optimizer uses these relationships with materialized views to perform query rewrite. The SQL Access Advisor uses these relationships to recommend creation of specific materialized views.
- **Directories:** An alias for a directory (called a folder on Windows systems) on the server file system where external binary file LOBs (BFILEs) and external table data are located.

Creating Physical Model Objects (continued)

- **Disk Groups:** A group of disks that Oracle Database manages as a logical unit, evenly spreading each file across the disks to balance I/O. Oracle Database also automatically distributes database files across all available disks in disk groups and rebalances storage automatically whenever the storage configuration changes.
- **External Tables:** A table that is stored outside the database but is available for access as though it were a table in the database
- **Roles:** A set of privileges that can be granted to users or to other roles. You can use roles to administer database privileges. You can add privileges to a role and then grant the role to a user. The user can then enable the role and exercise the privileges granted by the role.
- **Rollback Segments:** An object that the Oracle Database uses to store data necessary to reverse, or undo, changes made by transactions. Note, however, that Oracle strongly recommends that you run your database in automatic undo management mode instead of using rollback segments.
- **Segment Templates:** A set of extents that contains all the data for a logical storage structure within a tablespace.
- **Sequences:** An object used to generate unique integers. You can use sequences to automatically generate primary key values.
- **Snapshots**: Used to create a materialized view.
- **Stored Procedures:** A schema object that consists of a set of SQL statements and other PL/SQL constructs, grouped together, stored in the database, and run as a unit to solve a specific problem or perform a set of related tasks.
- **Synonyms:** An alternative name for a table, view, sequence, procedure, stored function, package, user-defined object type, or other synonym. Synonyms can be public (available to all database users) or private (only to the database user that owns the synonym).
- **Structured Types:** A non-simple data type that associates a fixed set of properties with the values that can be used in a column of a table
- **Collection Types:** Represent arrays or collections of elements (basic type, distinct type, structured type, or another collection) and are mapped to the Oracle VARRAY and nested table types
- **Tables:** Holds data. Each table typically has multiple columns that describe attributes of the database entity associated with the table, and each column has an associated data type. You can choose from many table creation options and table organizations (such as partitioned tables, index-organized tables, and external tables), to meet a variety of enterprise needs.
- **Tablespaces:** An allocation of space in the database that can contain persistent schema objects. Objects in permanent tablespaces are stored in data files.
- **Undo Tablespaces:** A type of permanent tablespace used by Oracle Database to manage undo data if you are running your database in automatic undo management mode
- **Temporary Tablespaces:** Contains schema objects only for the duration of a session. Objects in temporary tablespaces are stored in temp files.
- **Users:** An account through which you can log in to the database. Each database user has a database schema with the same name as the user.
- **Views:** A virtual table that selects data from one or more underlying tables

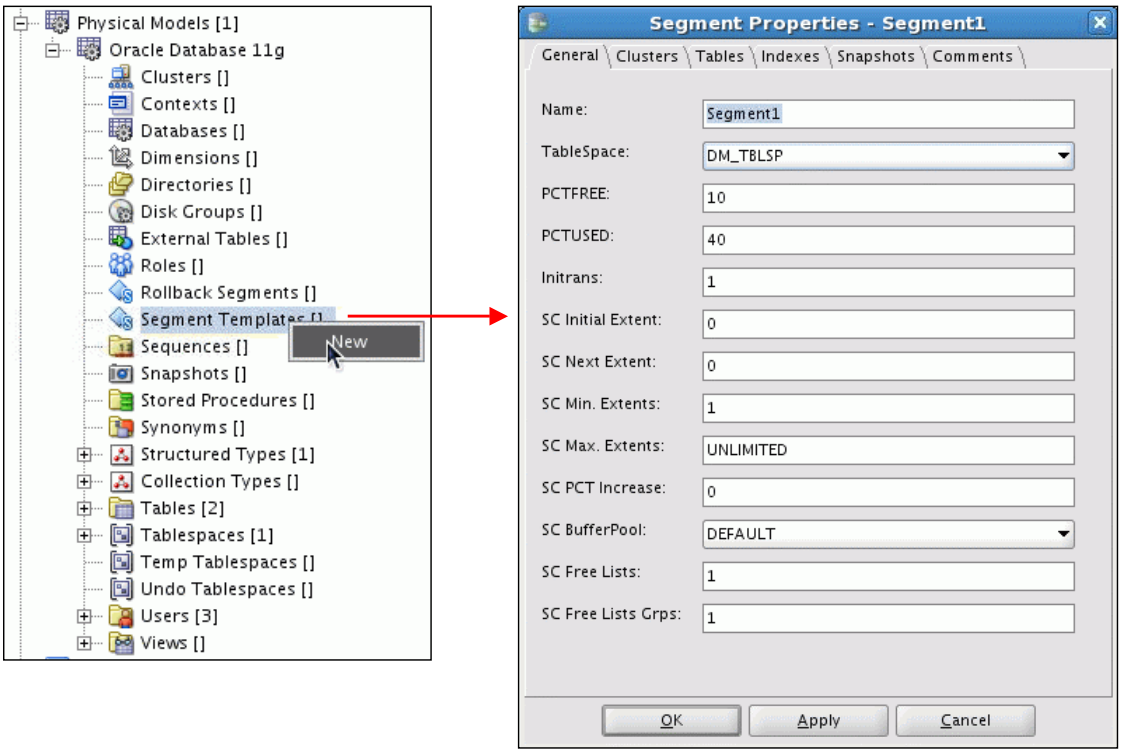
Adding a User



Adding a User

In order to generate the relational model objects into a particular user schema, you need to create the user in Oracle SQL Developer Data Modeler. In the example in the slide, a new user, DMUSER, is created and the tablespace created in the previous slide is selected in the Tablespace field. You can specify any roles or privileges here and then click OK to create the user.

Adding Segment Templates (Storage)

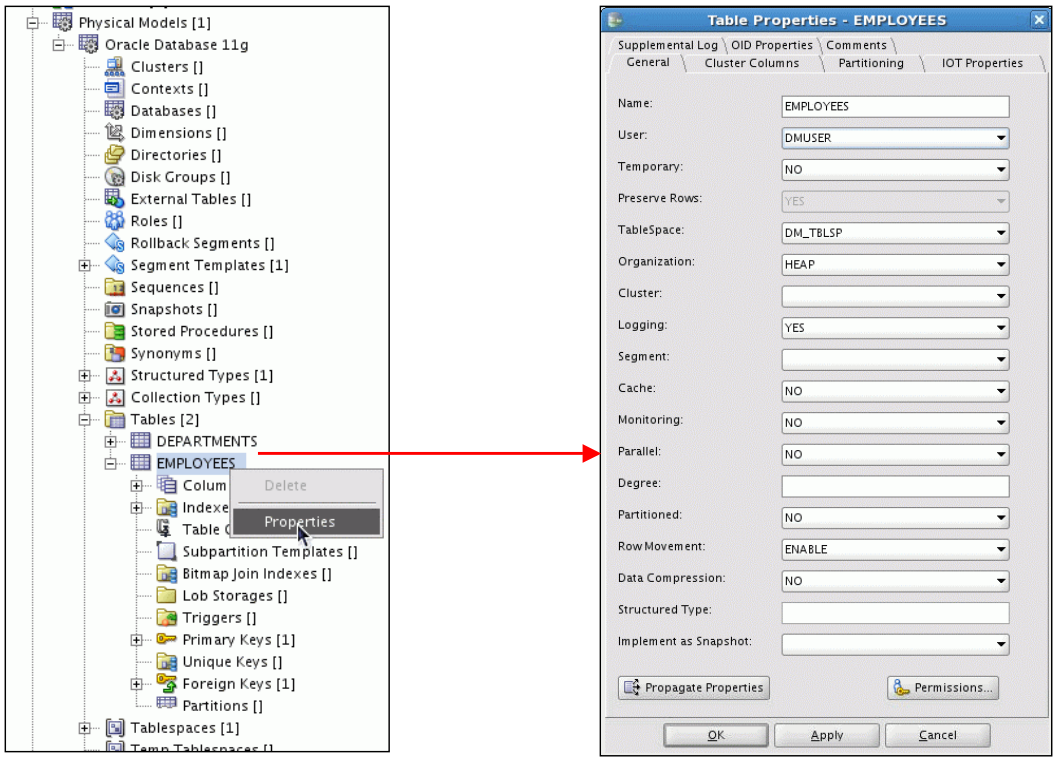


ORACLE

Adding Segment Templates (Storage)

Segment templates define the set of extents that contains all the data for a logical storage structure within a tablespace. This object is needed to figure out what the storage requirements will be for the objects in the database.

Associating Physical Objects with Your Table

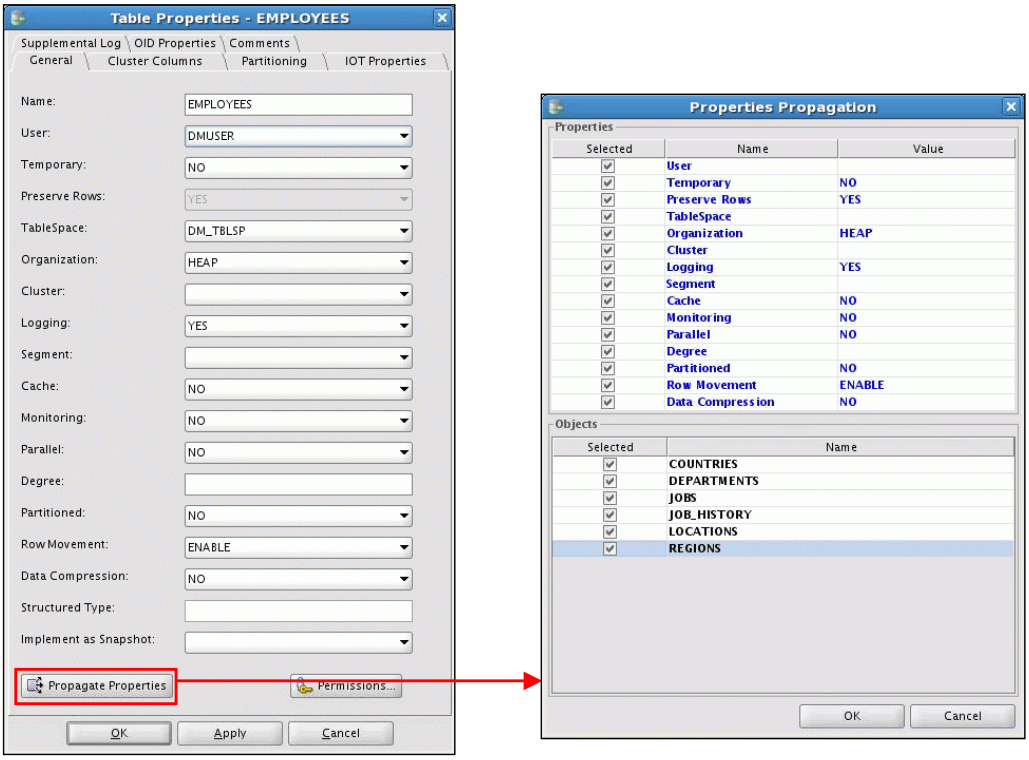


Associating Physical Objects with Your Table

The objects that you created in the previous slides can now be associated with your table in the physical model by performing the following steps:

1. Under your physical mode, expand Tables, right-click the table that you want to define, and select Properties. In the example in the slide, the properties for the EMPLOYEES table are shown.
2. Specify all the properties for this table and click OK. In this case, you select DM_TBLSP for Tablespace and DMUSER for User.

Propagating Properties to Other Physical Objects

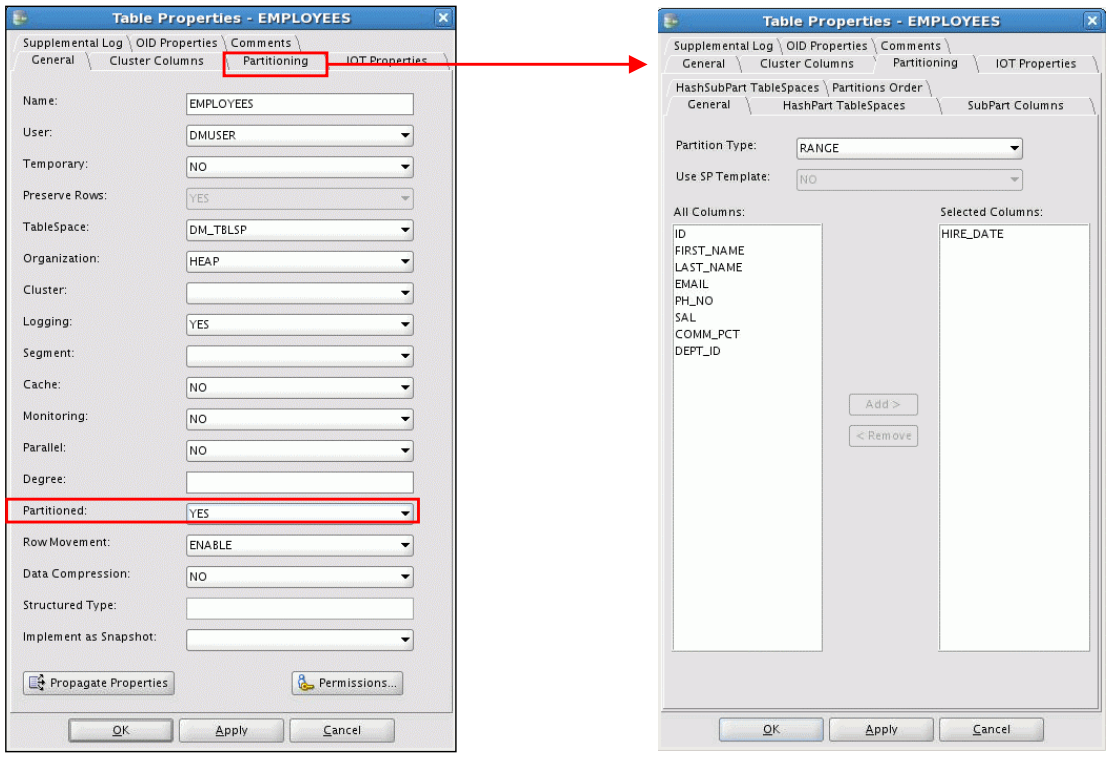


ORACLE

Propagating Properties to Other Physical Objects

When the Propagate Properties button in any object properties window is clicked, a list of properties for that object appears with a list of the objects that you can propagate the properties to. This capability is very useful when you want to assign, for example, the same user and tablespace to all tables in your physical model.

Partitioning a Table



Partitioning a Table

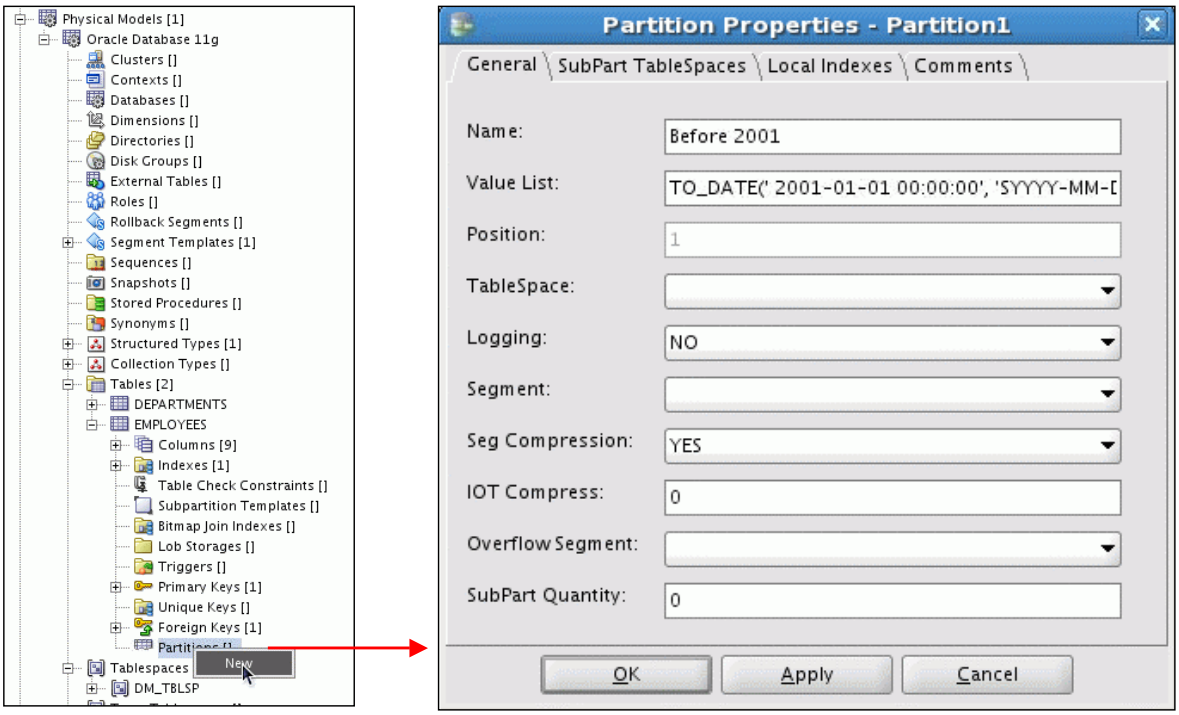
Partitioning a table improves data load and retrieval performance for large tables. In the example in the slide, you may want to partition the EMPLOYEES table on the HIREDATE column so that you can retrieve employee information for a range of dates on which employees were hired.

To create a partitioned table, perform the following steps:

1. In the Table Properties window, select Yes for Partitioned.
2. Click the Partitioning tab.
3. Select the type of partitioning that you want to perform from the drop-down list. In the example in the slide, you select RANGE.
4. Select the column that you want to partition and click Add.
5. Click OK.

These steps create the definition of the partition. In the next slide, you determine the partition definitions themselves.

Partitioning a Table

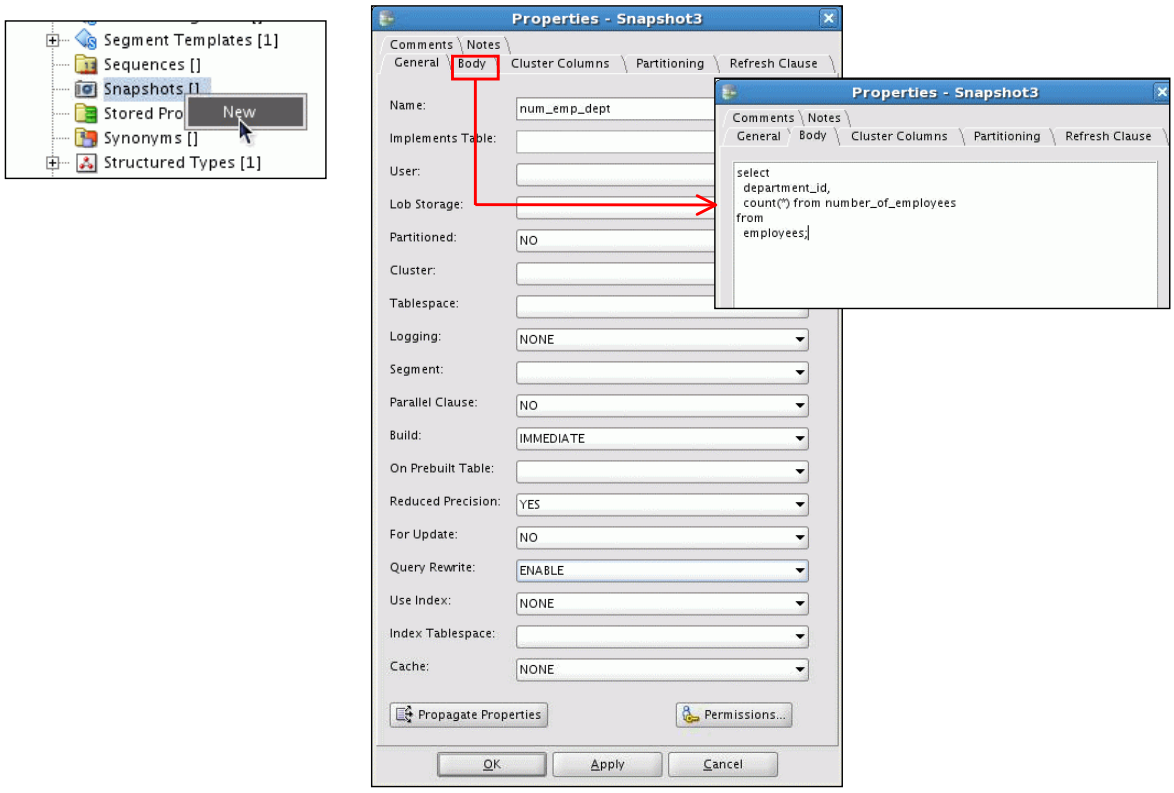


Partitioning a Table (continued)

After you specify that you want to partition the table and what type of partition you want, you create each of your partitions by performing the following steps:

1. In the object browser, under the physical model for your table, right-click Partitions.
2. In the Partition Properties dialog box, enter the partition criteria in the Value List field, complete the other fields, and click OK.

Creating a Materialized View



ORACLE

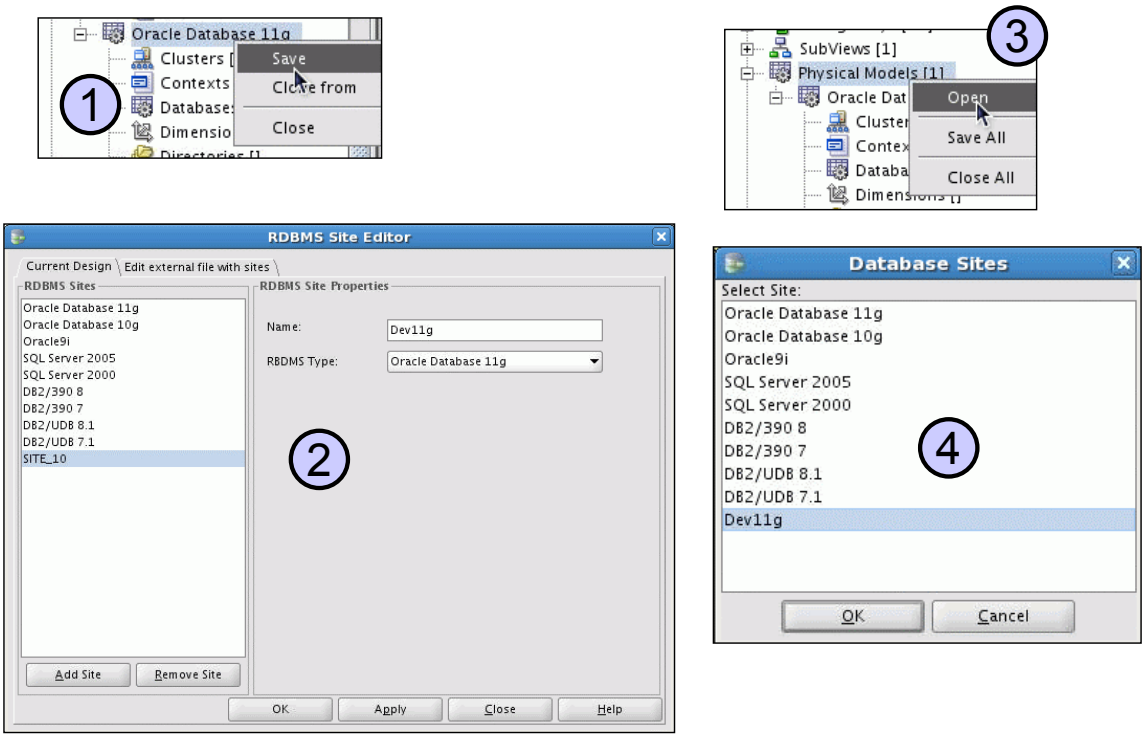
Creating a Materialized View

Materialized views are query results that have been stored or “materialized” in advance as schema objects. In your physical model, you can create a snapshot that will generate DDL for a materialized view (discussed in the next lesson).

To create a materialized view, perform the following steps:

1. Right-click Snapshots and select New
2. Complete the Name and other fields on the General tab, and click the Body tab.
3. Enter the select statement that you want to create the materialized view for, and click OK.

Cloning a Database

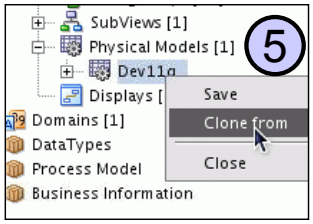


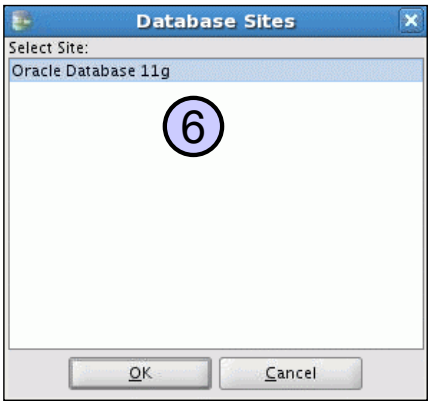
Cloning a Database

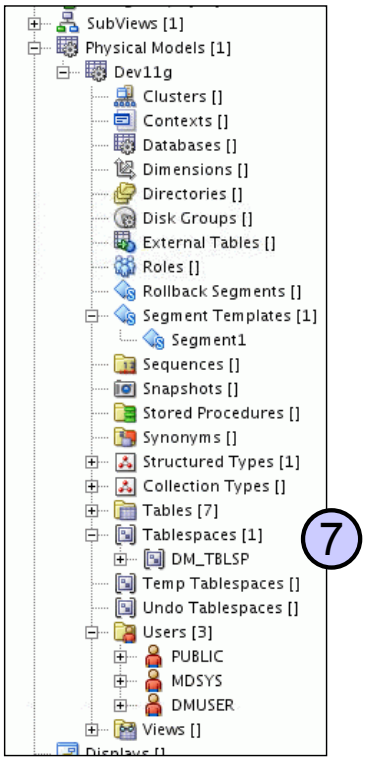
If you want to upgrade a database or copy a database to another physical model (that, for example, you want to use for development [versus production] purposes), you can clone your physical model to another physical model so that it has all the same physical objects. In Oracle SQL Developer Data Modeler, this is called cloning your database. To clone a database, you perform the following tasks:

1. Create your physical model as discussed previously in this lesson. Right-click your physical model and select Save.
2. You must create a new RDBMS site. Select Tools > RDBMS Site Administration. Click Add, enter a name, select the RDBMS type, and click OK.
3. Right-click Physical Model in your object browser.
4. Select the RDBMS that you just created, and click OK.

Cloning a Database







ORACLE

Cloning a Database (continued)

5. Right-click the physical model that you just opened, and select “Clone from.”
6. Select the other physical model that you want to clone from, and click OK.
7. Expand the nodes in your object browser. Notice that the objects you created in the other physical model are now created in this physical model.

Quiz

A physical model: (Select all that apply.)

a. Must be complete before you can generate the database
b. Is created for each database implementation
c. Captures detail that the DBA will maintain
d. Contains objects that can optimize the performance of your database

ORACLE

Answer: b, c, d

Summary

In this lesson, you should have learned how to:

- Create objects in a physical model
- Refine relational model objects in the physical model

ORACLE

Summary

In this lesson, you should have learned how to create a physical model, create some physical model objects, and refine some relational model objects in the physical model.

Practice 18-1 Overview: Create a Physical Model

This practice covers the following topics:

- Creating a physical model
- Creating and refining some objects in the physical model

19

Generating Your Database

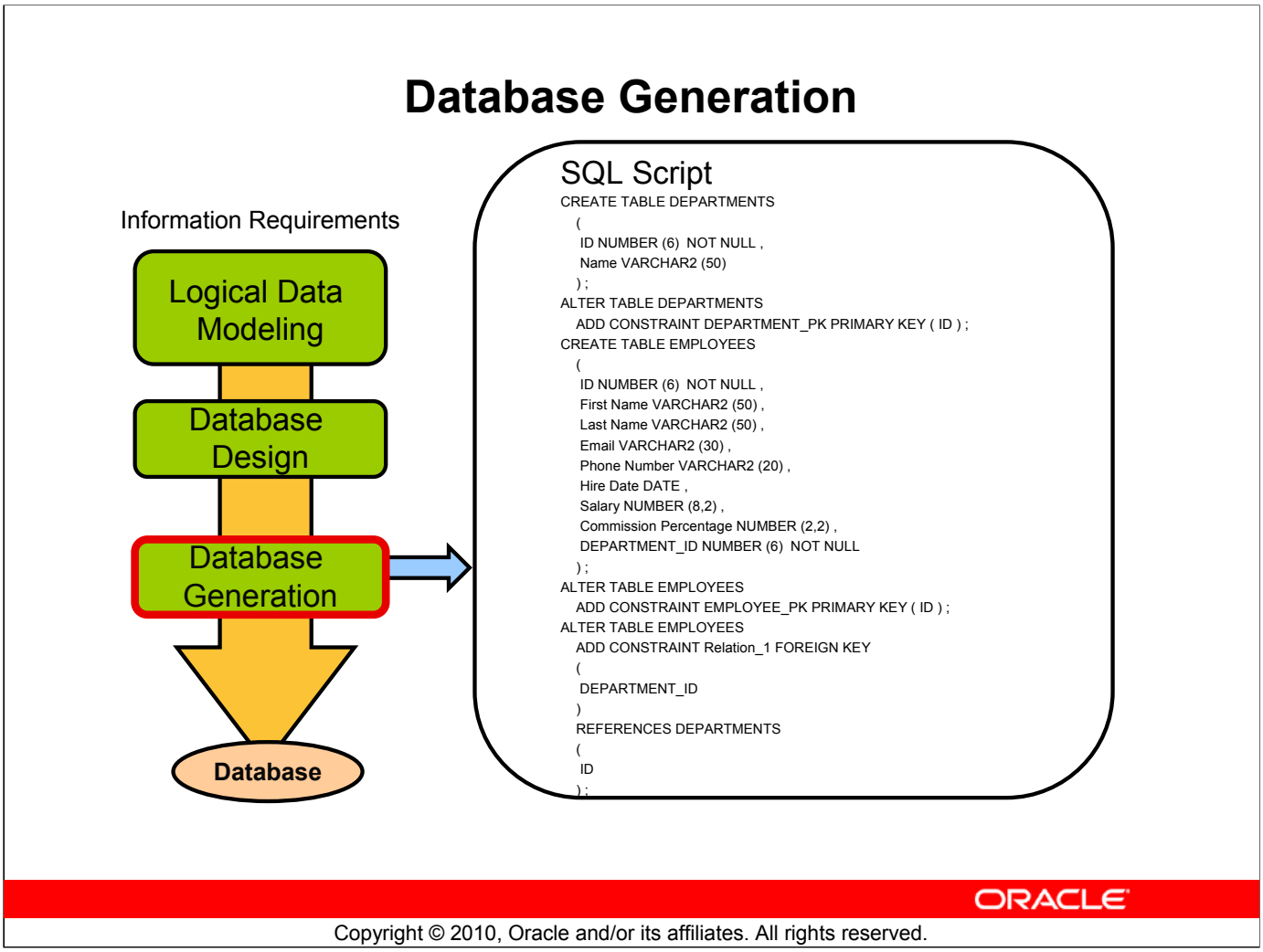
Objectives

After completing this lesson, you should be able to:

- Generate DDL for your database

Objectives

In this lesson, you generate DDL for your database.

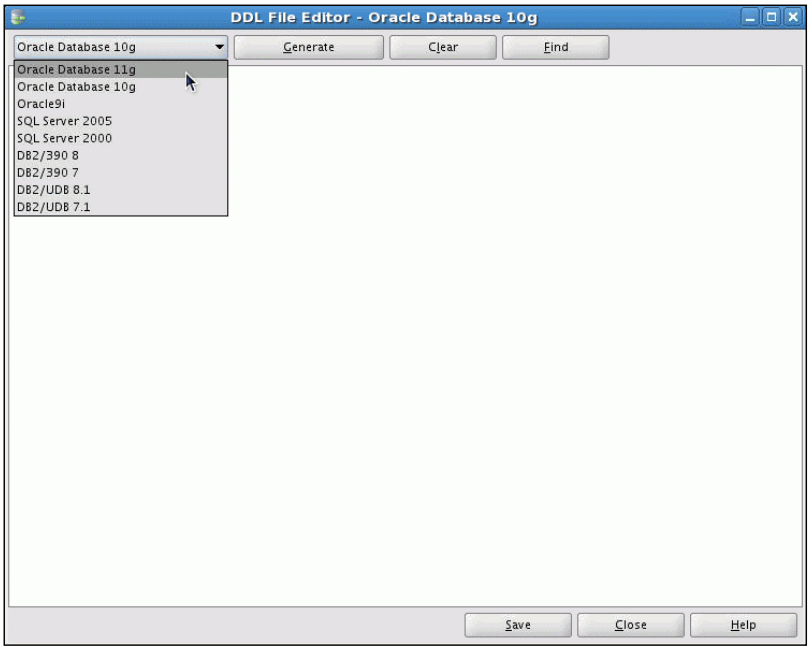


Database Generation

The Database Generation process involves completing the physical model and generating the SQL script that will contain the SQL statements necessary to create the database. The purpose of the physical model is to describe a database in terms of Oracle Database objects (tablespaces, tables, views, triggers, and so on) that are based on a relational model. Each relational model can have one or more physical models. Each physical model is based on an RDBMS site object. An RDBMS site is a name associated with a type of database supported by data modeling such as the Oracle Database 11*g*.

You will Oracle SQL Developer Data Modeler to generate the SQL script that you use to generate your database. In the slide is a portion of the SQL script that was generated for the DEPARTMENTS and EMPLOYEES tables.

Generating DDL: Selecting a Database



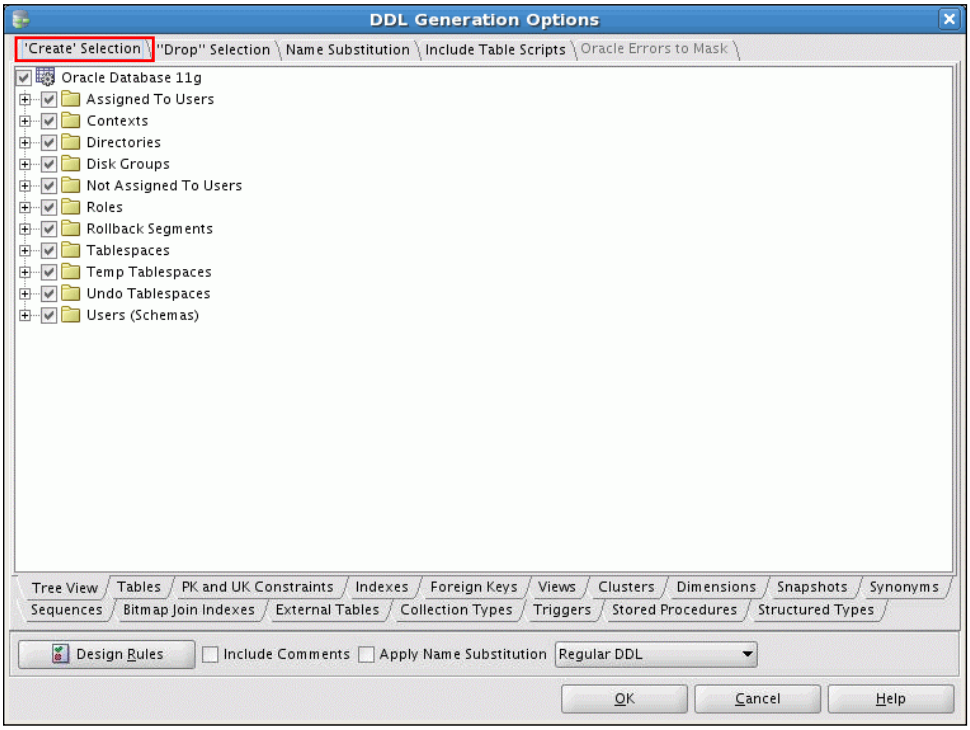
ORACLE

Generating DDL: Selecting a Database

To generate the DDL, you perform the following steps:

1. Click the Generate DDL icon in the toolbar. The DDL File Editor window appears.
2. From the drop-down list, select the database that you want to generate to.
3. Click the Generate button.

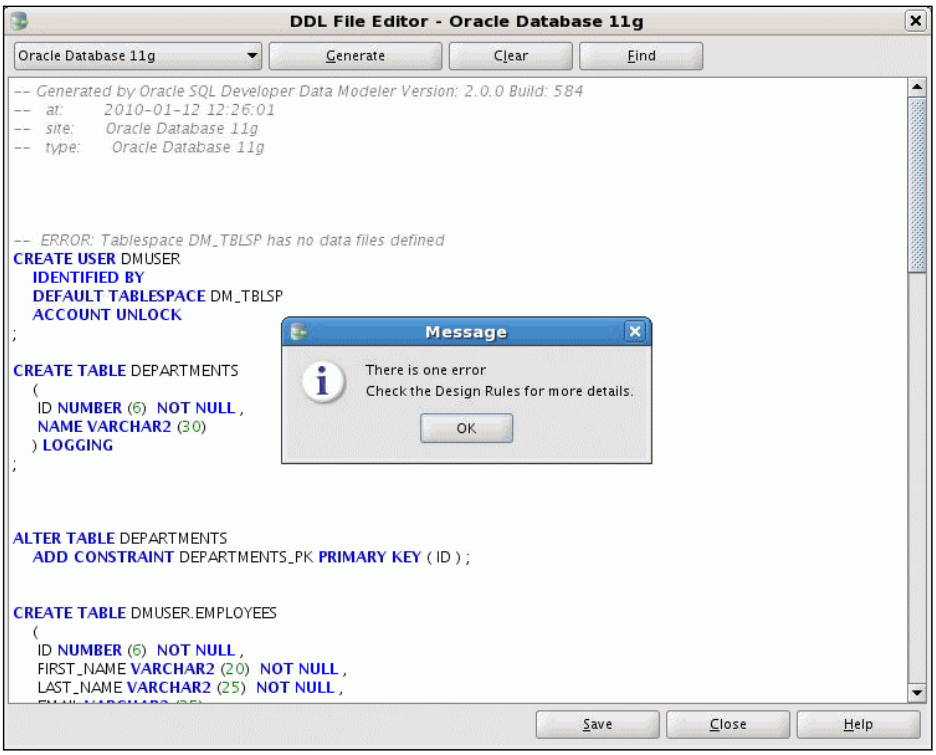
Generating DDL: 'Create' Selection



Generating DDL: 'Create' Selection

By default, all the objects are selected. If you do not have a physical model, you will not see many of the objects that are shown in the screenshot in the slide. When you are ready to generate the DDL script, click OK.

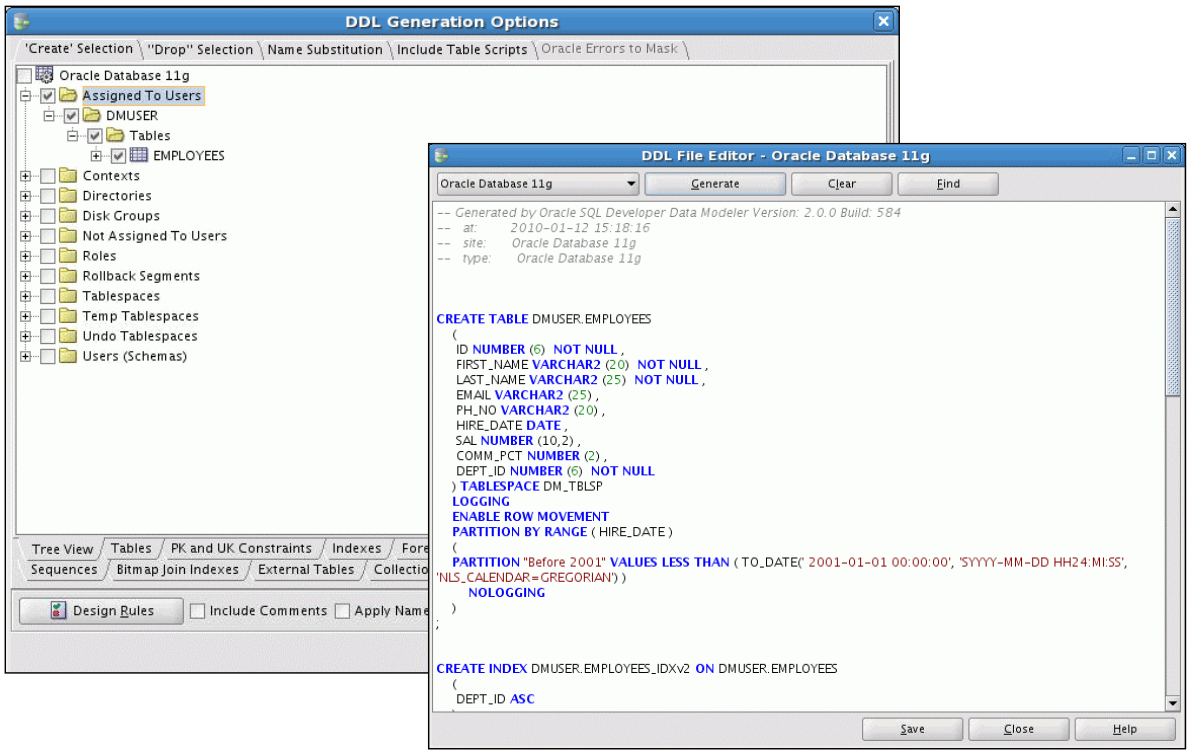
Generating DDL: DDL Script



Generating DDL: DDL Script

The DDL script is generated with all the objects that you selected on the previous slide. In this case, there is a design error because the tablespace associated with the table did not have any data files defined. To correct the error, close this window, define the data files for the tablespace, and then generate the DDL again.

Generating DDL: Assigned to Users

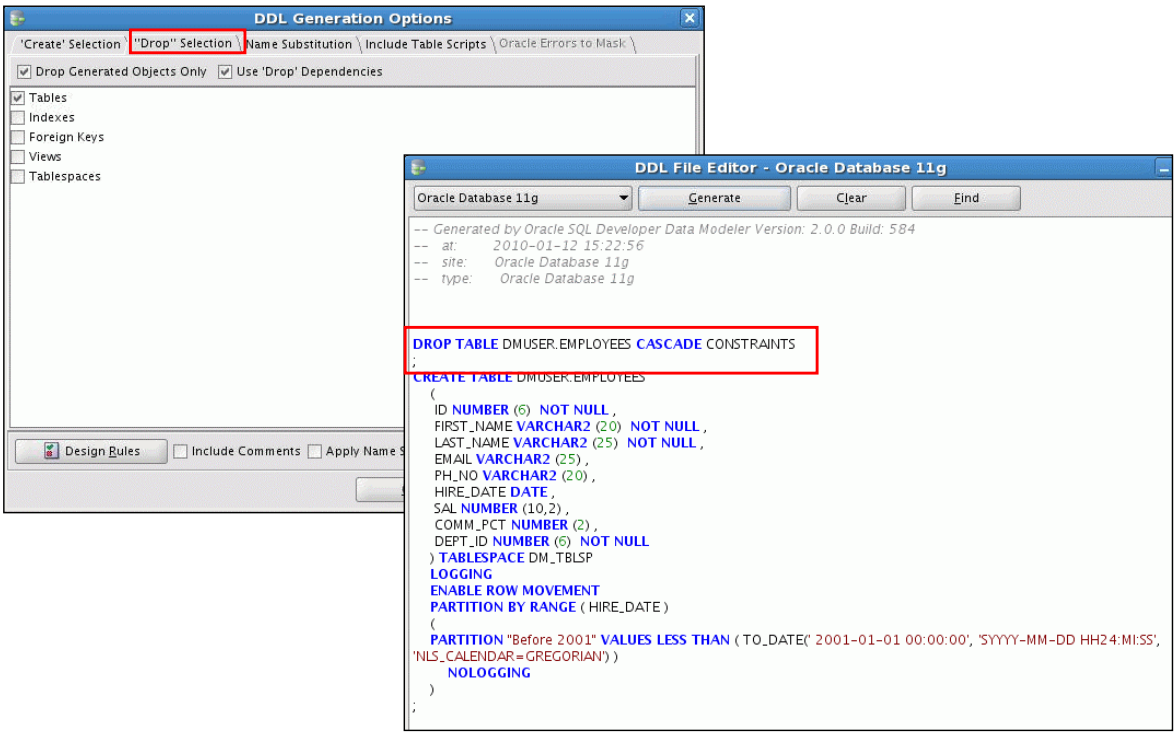


ORACLE

Generating DDL: Assigned to Users

To generate only objects that are assigned to a specific user, deselect the high-level node (in this case Oracle Database 11*g*), select the check box next to Assigned To Users, and then click OK.

Generating DDL: “Drop” Selection

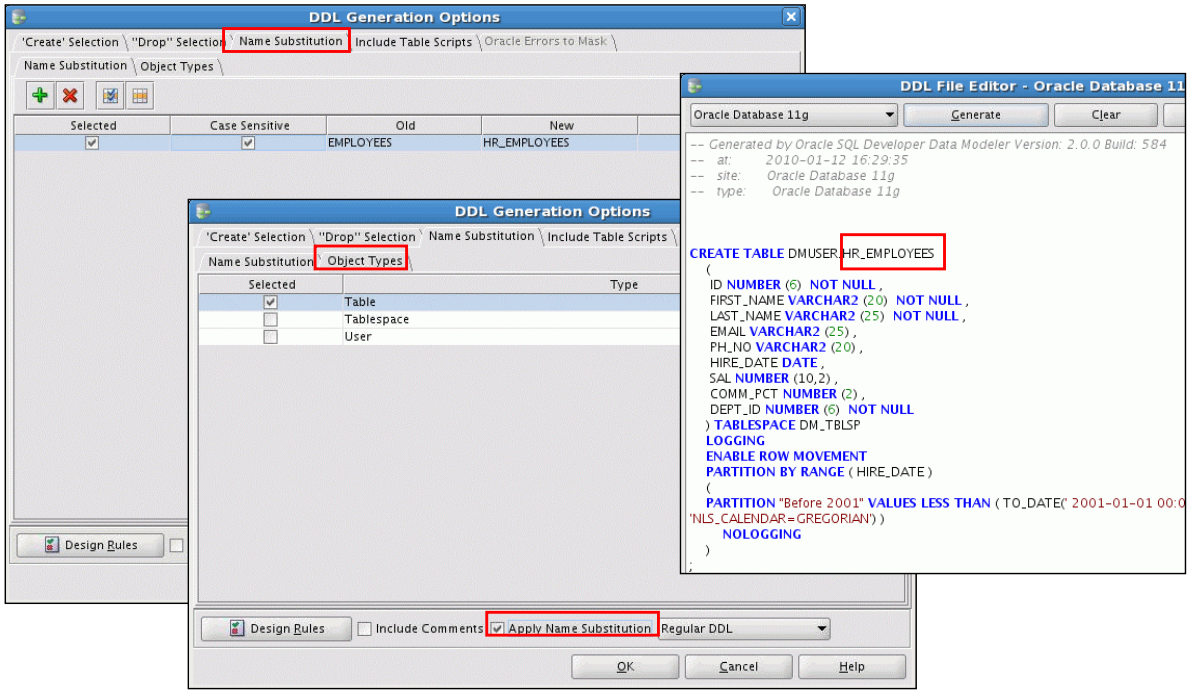


ORACLE

Generating DDL: “Drop” Selection

You can also generate drop statements before the create statement in case the table already exists. Be careful not to do this with production systems because if the drop is executed, the existing table will be dropped before a new empty table will be created.

Generating DDL: Name Substitution



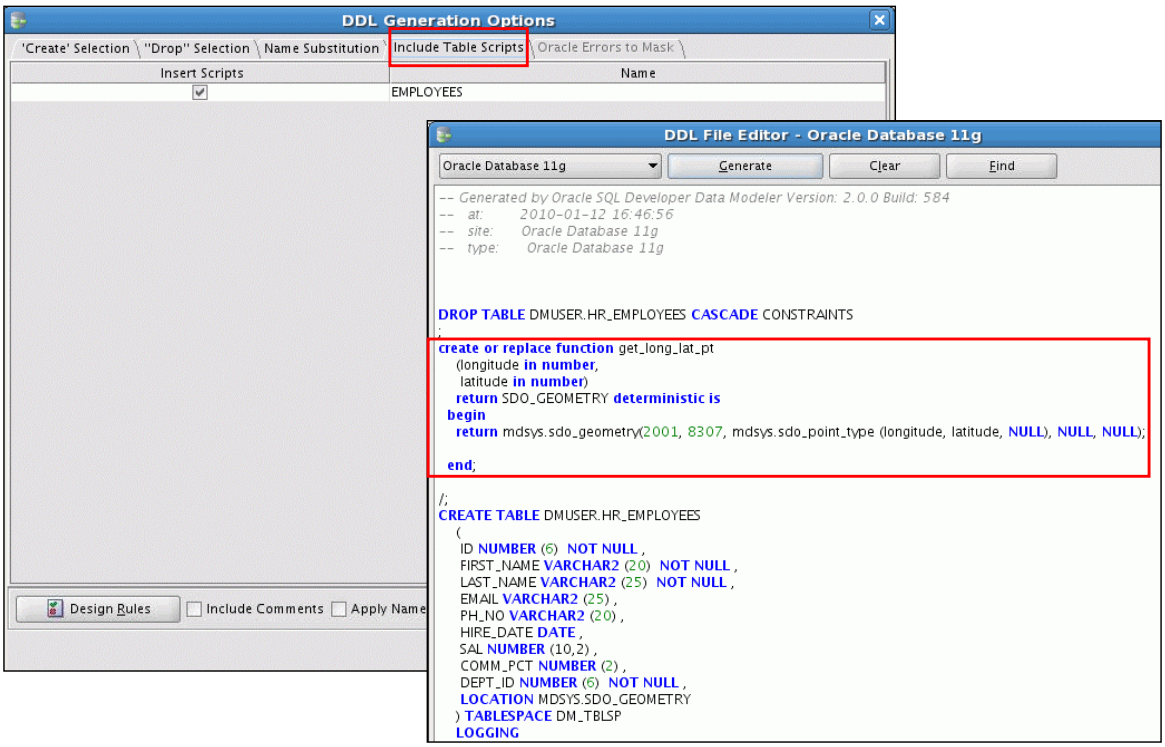
ORACLE

Generating DDL: Name Substitution

When generating the DDL you can specify a name substitution for tables, tablespaces and users. Perform the following steps:

1. Select the Name Substitution tab.
2. Under the Name Substitution subtab, click the Add '+' icon.
3. Specify the old and new names and whether it is case-sensitive.
4. Select the Object Types subtab.
5. Select the check box for the object types that you want to apply the name substitution to.
6. Select the Apply Name Substitution check box and click OK.

Generating DDL: Including Table Scripts



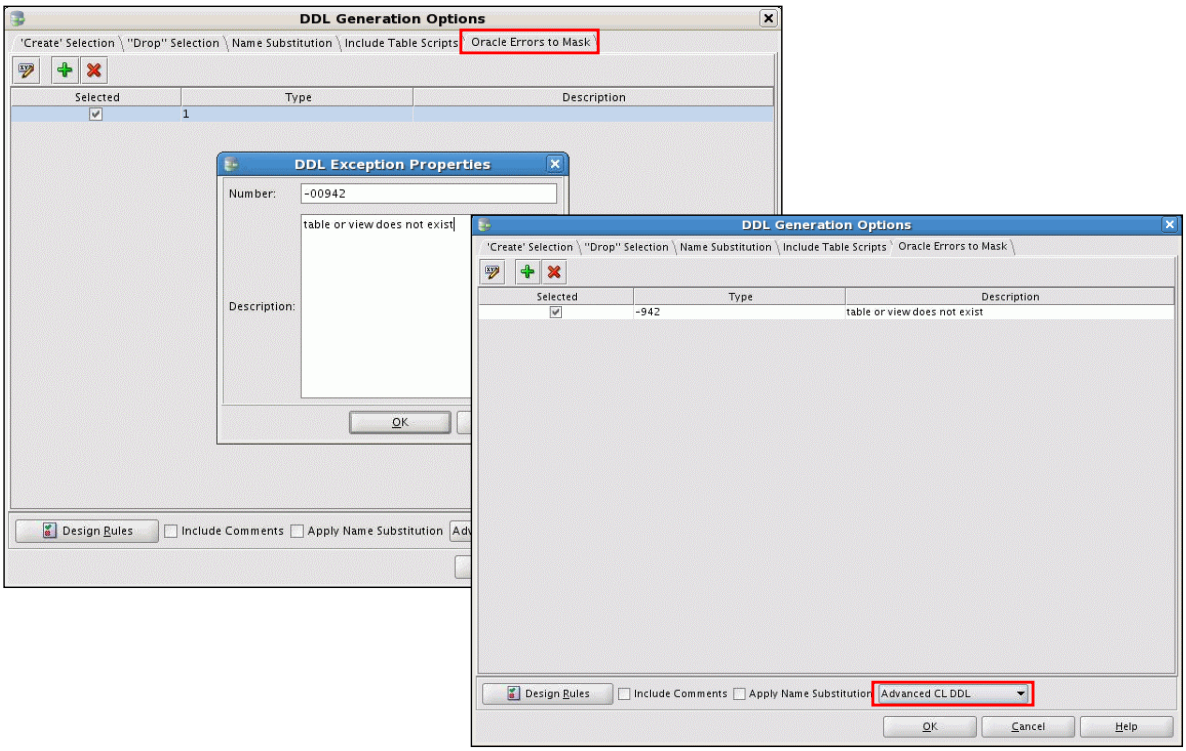
ORACLE

Generating DDL: Including Table Scripts

In the lesson where you reviewed your relational model, you created a new spatial column and added a script to create a function. When generating DDL, you can specify whether you want the script to be generated. Perform the following steps:

1. Select the Include Table Scripts tab.
2. To specify that the script be included, select the Insert Scripts check box and then click OK.

Generating DDL: Masking Oracle Errors

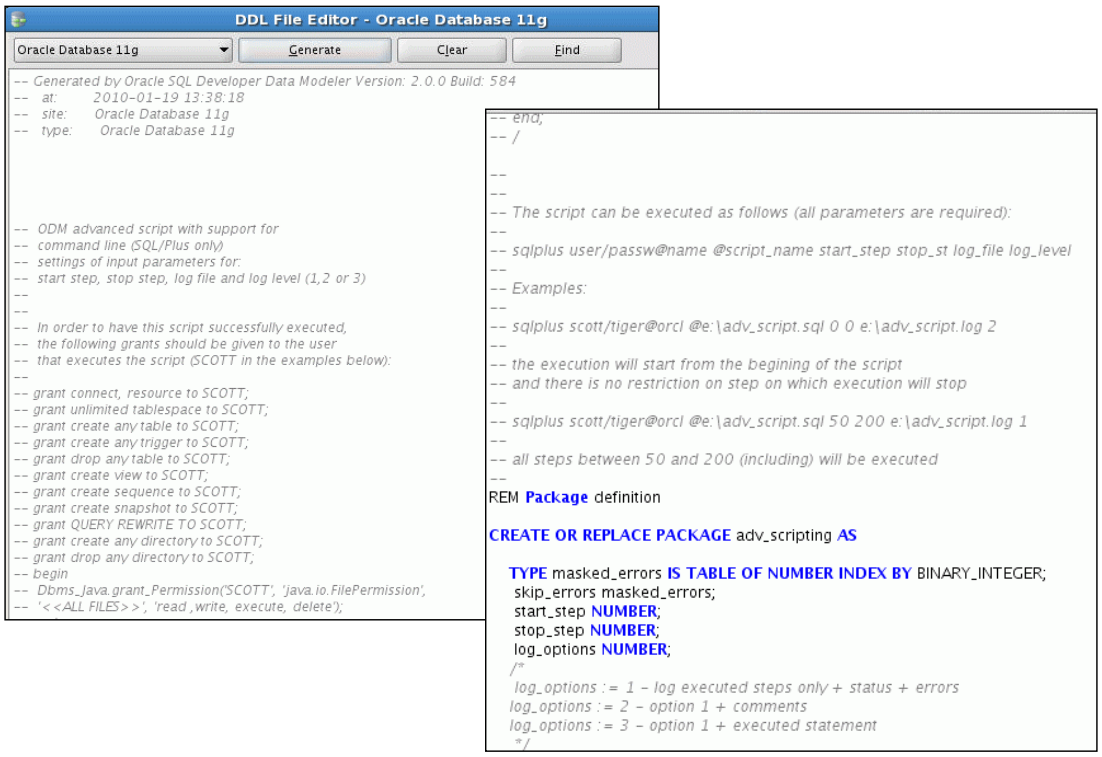


Generating DDL: Masking Oracle Errors

You can specify that certain Oracle Errors to be ignored during script execution. This option is only available if you are generating Advanced Interactive DDL or Advanced CL DDL. Perform the following steps:

1. From the drop-down list at the bottom of the DDL Generation window, select Advanced Interactive DDL or Advanced CL DDL.
2. Select the “Oracle Errors to Mask” tab.
3. Click the Add icon.
4. Double-click the number 1 to open the DDL Exception Properties window.
5. Specify the error number in the Number field. Do not include the ORA prefix.
6. Enter a description, and click OK.
7. You see the errors to mask in the list. Click OK.

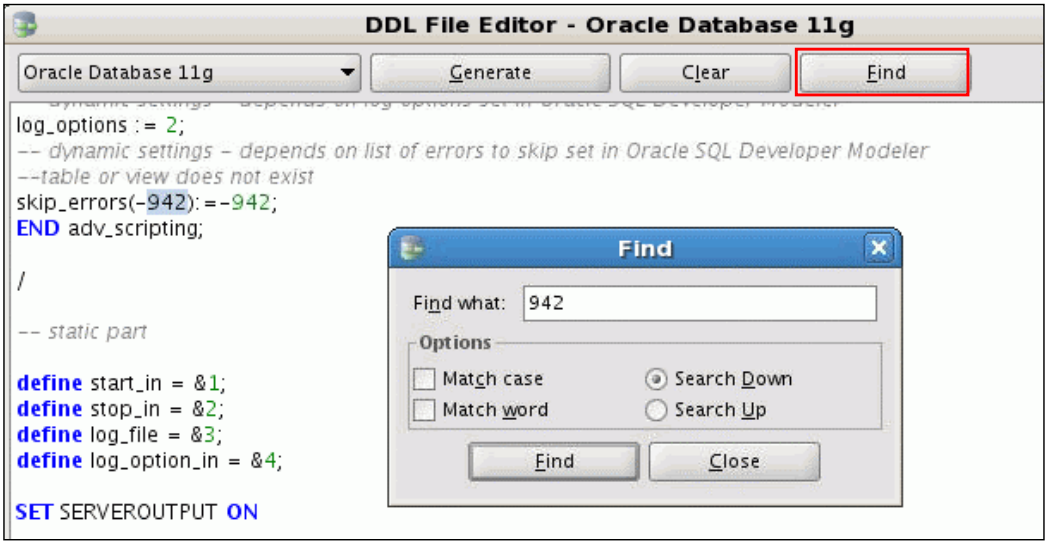
Generating DDL: Masking Oracle Errors



Generating DDL: Masking Oracle Errors (continued)

The DDL is displayed. There are instructions at the top of the script that explain the privileges and steps that need to be performed to execute the script. A number of packages are generated to handle the logic necessary to mask the identified errors.

Generating DDL: Using Find

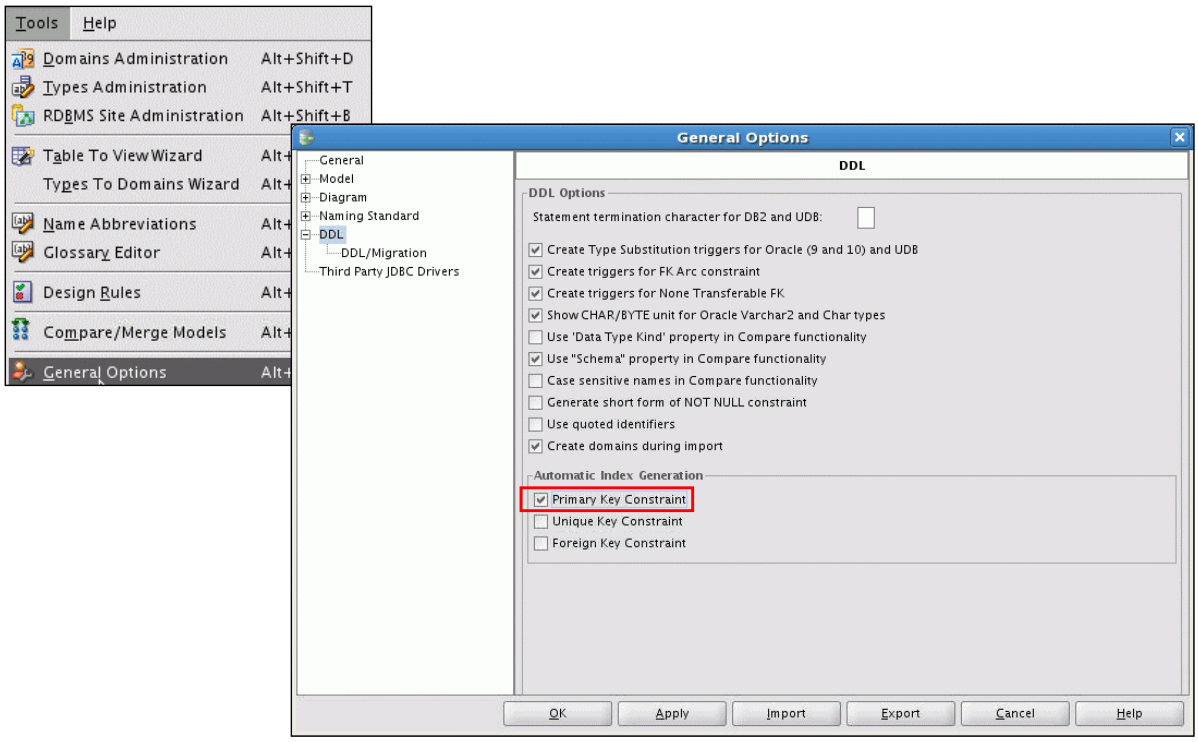


ORACLE

Generating DDL: Using Find

The Find option allows you to search for any word in the DDL script that is generated. In the example in the slide, you search for error number 942.

DDL General Options



ORACLE

DDL General Options

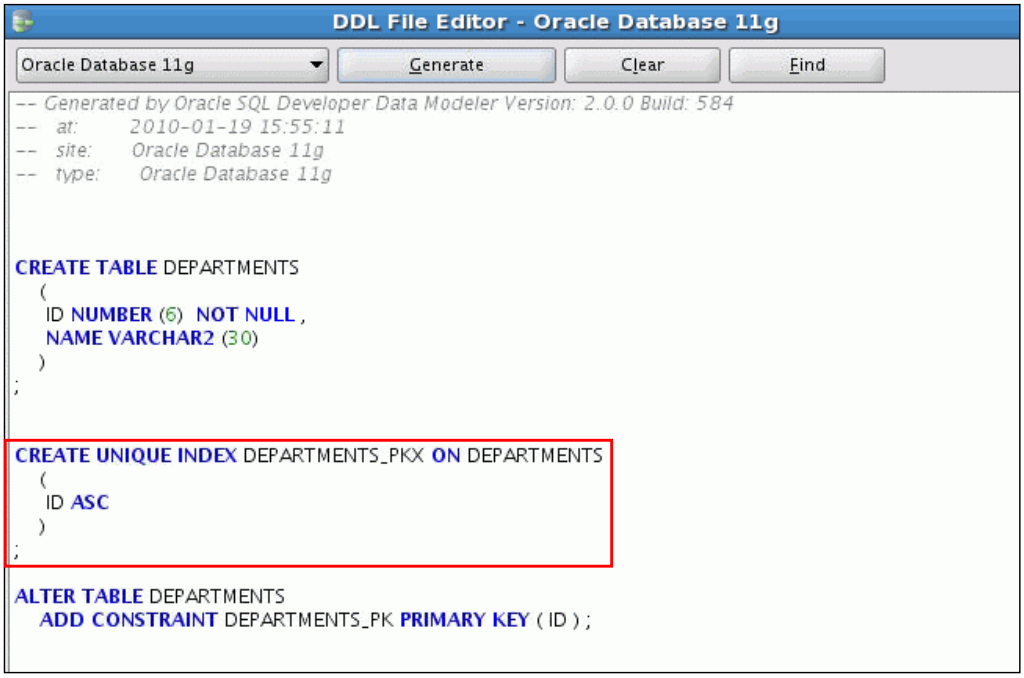
You can specify the following defaults for DDL generation. Select Tools > General Options and select DDL in the object browser hierarchy.

- **Statement termination character for DB2 and UDB:** The termination character for DDL for IBM DB2 and UDB databases
- **Create Type Substitution triggers for Oracle and UDB:** Controls whether triggers are created for type substitutions in Oracle and IBM UDB physical models
- **Create Triggers for FK Arc constraint:** Controls whether triggers are created in generated DDL code to implement foreign key arc constraints
- **Create triggers for Non Transferable FK:** Controls whether triggers are created for non-transferable foreign key relationships. Whether a foreign key relationship is transferable is controlled by the Transferable (Updateable) option.
- **Show CHAR/BYTE unit for Oracle Varchar2 and Char types:** Controls whether, for attributes of Oracle type CHAR or VARCHAR2, the unit (CHAR or BYTE) associated with the attribute length is included for columns based on the attribute in relational model diagrams and in generated CREATE TABLE statements

DDL General Options (continued)

- **Use 'Data Type Kind' property in Compare functionality:** Controls whether the data type kind (such as domain, logical type, or distinct type) should be considered to prevent types of different kinds from generating the same native data type (for example, preventing a domain and a logical type from resulting in Number(7,2))
- **Use “Schema” property in Compare functionality**
- **Case sensitive names in Compare functionality**
- **Generate short form of NOT NULL constraint**
- **Use quoted identifiers**
- **Create domains during import**
- **Automatic Index Generation:** You can specify whether you want indexes to be created automatically if one of the following constraints is defined.
  - Primary Key Constraint
  - Unique Key Constraint
  - Foreign Key Constraint

DDL General Options

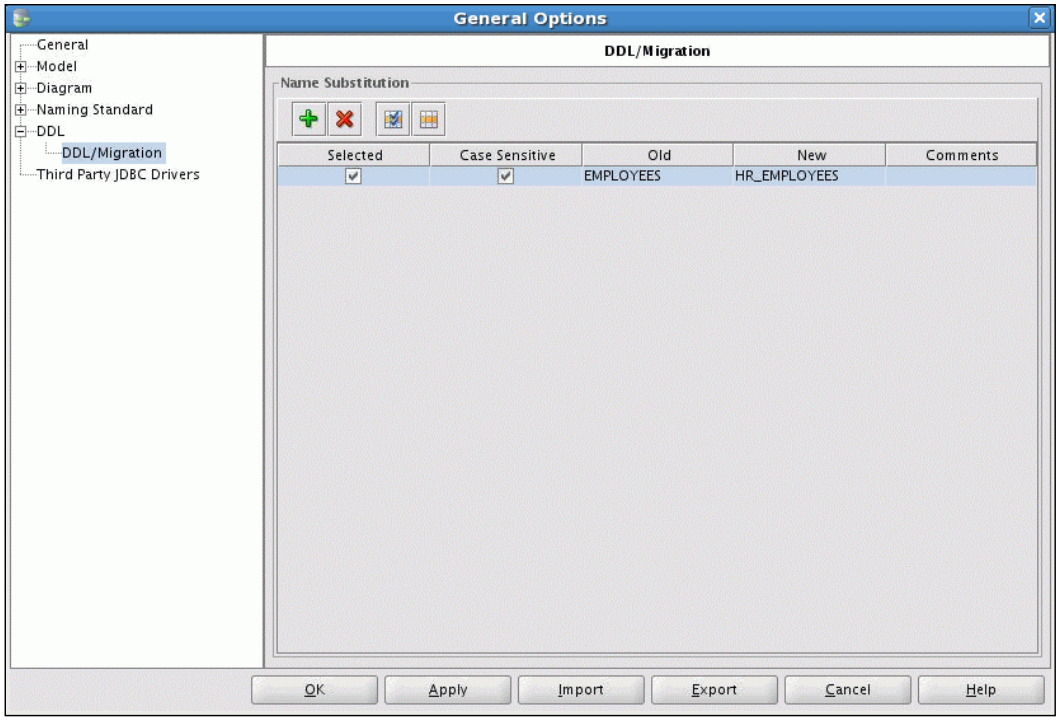


ORACLE

DDL General Options (continued)

In the example in the slide, notice that an index was automatically created for the primary key because you specified the parameter under General Options.

DDL/Migration General Options



ORACLE

DDL/Migration General Options

You can specify the following defaults for DDL/Migration generation. Select Tools > General Options, expand DDL in the object browser hierarchy and select DDL/Migration. This option allows you to specify one or more pairs of string replacements to be made when DDL statements are generated. Each pair specifies the old string and the new string with which to replace the old string. The options are:

- **Selected:** Controls whether the specified replacement is enabled or disabled
- **Case Sensitive:** Controls whether the replacement is done only if the case of the old string in the DDL exactly matches the case specifies for the old string

Note that these defaults will be automatically used under the Name Substitution tab in the Generate DDL window.

Summary

In this lesson, you should have learned how to:

- Generate DDL for your database

ORACLE

Summary

In this lesson, you should have learned how to generate the DDL for your database from a model.

Practice 19-1 Overview: Generate DDL

This practice covers the following topics:

- Generating the DDL script for your database

VII

Other Needs for Modeling

Overview

In this unit, you examine the following areas:

- Lesson 20: Altering an Existing Design
- Lesson 21: Creating a Multidimensional Model

20

Altering an Existing Design

ORACLE

Objectives

After completing this lesson, you should be able to:

- Import from the data dictionary
- Reverse engineer to create the logical model
- Compare and merge models
- Export your model
- Analyze your model by running Data Modeler reports

ORACLE

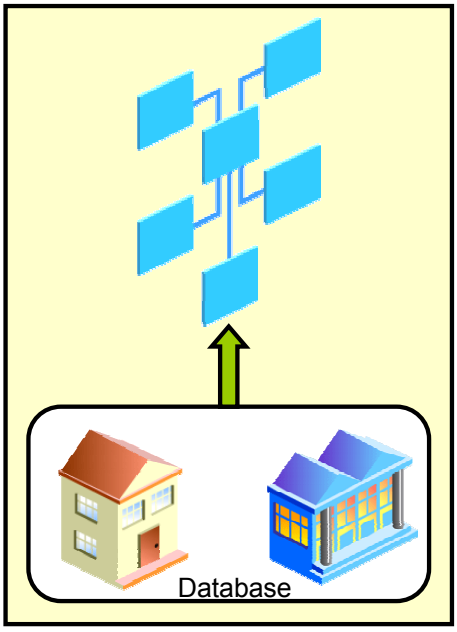
Objectives

In this lesson, you learn how to import an existing database, reverse engineer to create the logical model, compare and merge models, export your model, and analyze your model within SQL Developer.

Approaches to Modeling

Bottom-up modeling: Modifying an existing database

1. Produce a relational model.
2. Modify the relational model and create additional relational models.
3. Reverse engineer the logical model from a relational model.
4. Modify and check design rules for the logical model.
5. Generate modified DDL code.

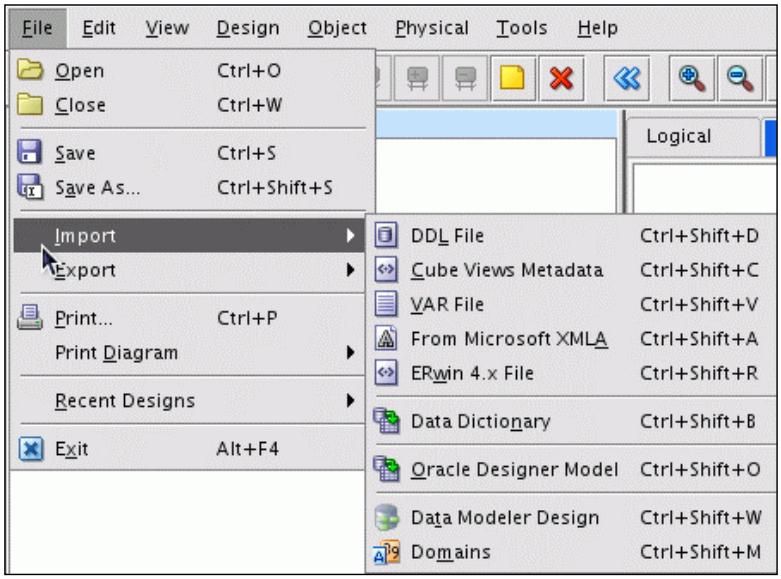


ORACLE

Approaches to Modeling

The bottom-up modeling approach modifies an existing database definition. Bottom-up modeling builds a database design based on either metadata extracted from an existing database or a file with DDL code that implements an existing database. The resulting database is represented as a relational model and a physical model, and you reverse engineer the logical model from the relational model. Bottom-up modeling can involve the steps listed in the slide, but you can abbreviate or skip some steps as appropriate for your needs.

Using Import



ORACLE

Using Import

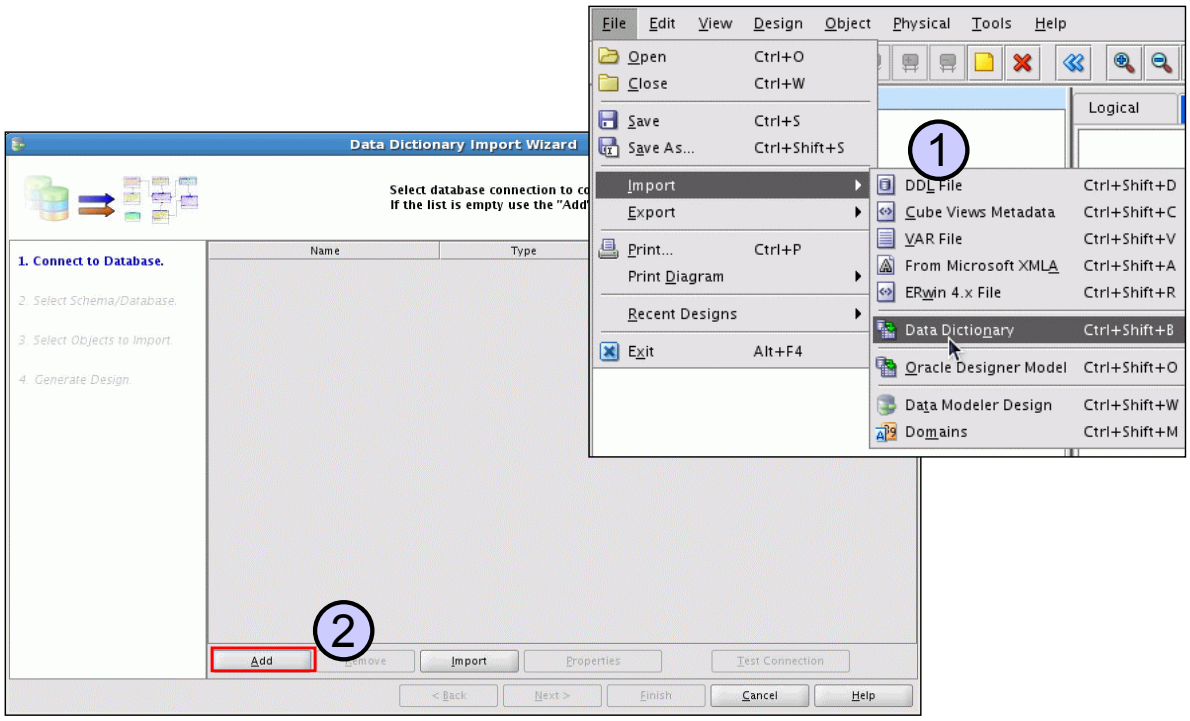
There are a number of objects that can be imported to create a model. These include:

- **DDL File:** Create a relational model based on a DDL file. DDL files can originate from any supported database type and version. The file to be imported usually has the extension .ddl or .sql. The import process creates a new relational model with the name of the imported DDL file and opens a physical model reflecting the source site.
- **Cube Views Metadata:** Creates a multidimensional model based on an existing implementation, as reflected in a specified XML file
- **VAR File:** Creates a logical and/or relational model based on an exported Bachman formatted metadata file
- **From Microsoft XMLA:** Creates a multidimensional model stored in the Microsoft XMLA file format
- **ERwin 4.x File:** Captures logical and relational models from the ERwin modeling tool. Specify the XML file containing definitions of the models to be imported.
- **Data Dictionary:** Creates a relational model and a physical model based on an existing database implementation. The data dictionary can be from any supported database type and version.

Using Import (continued)

- **Oracle Designer Model:** Creates a relational model and a physical model based on an existing Oracle Designer model
- **Data Modeler Design:** Captures the logical model and any relational and data type models from a design previously exported from Oracle SQL Developer Data Modeler
- **Domains:** Creates domains based on a domain file

Importing an Existing Database



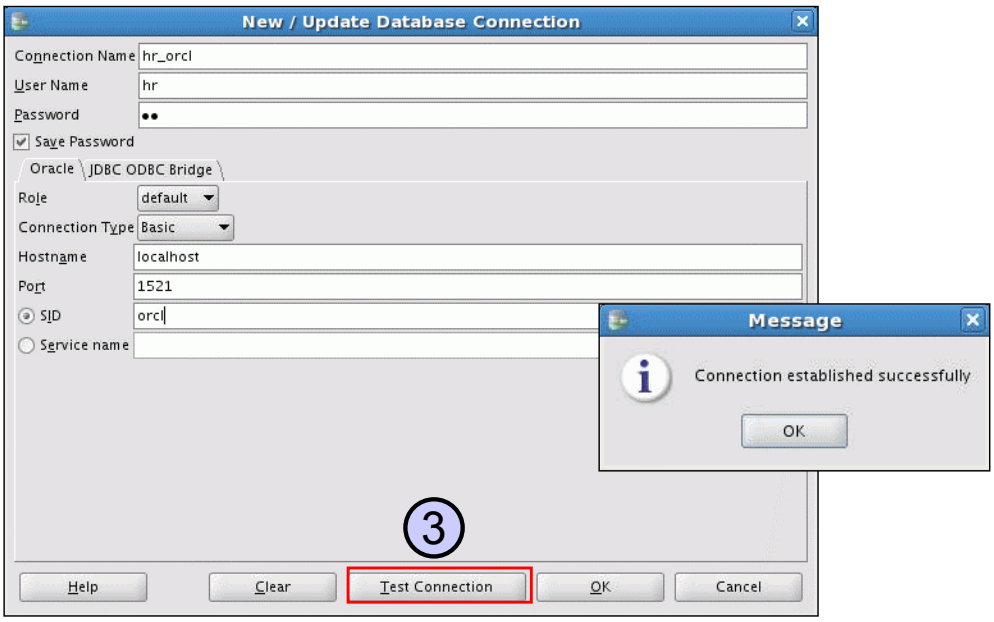
ORACLE

Importing an Existing Database

You can create a relational model of what exists in the database (or data dictionary) by using import. Perform the following steps:

1. Select File > Import > Data Dictionary.
2. In the Data Dictionary Import Wizard window, click Add to add a connection to your database.

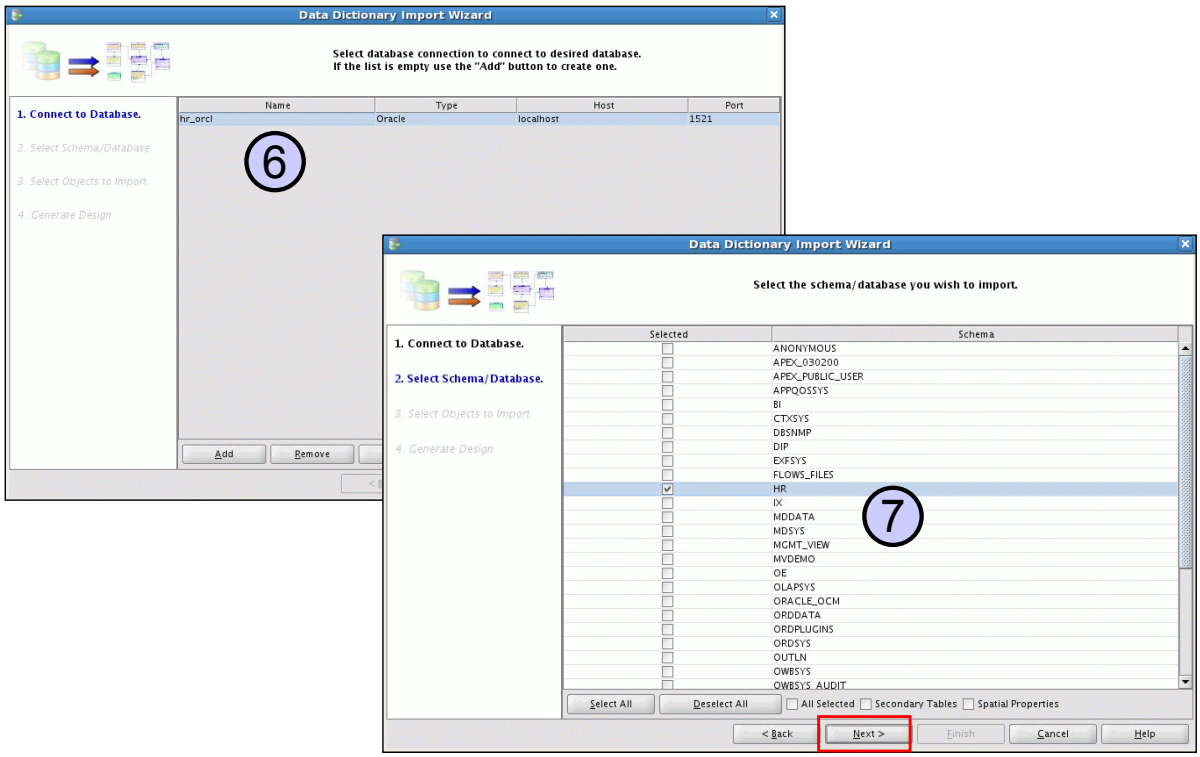
Importing an Existing Database



Importing an Existing Database (continued)

3. Enter your connection information and click Test Connection.
4. Click OK when the message indicates that the connection was successful.
5. Click OK in the Update Database Connection window to create the connection.

Importing an Existing Database



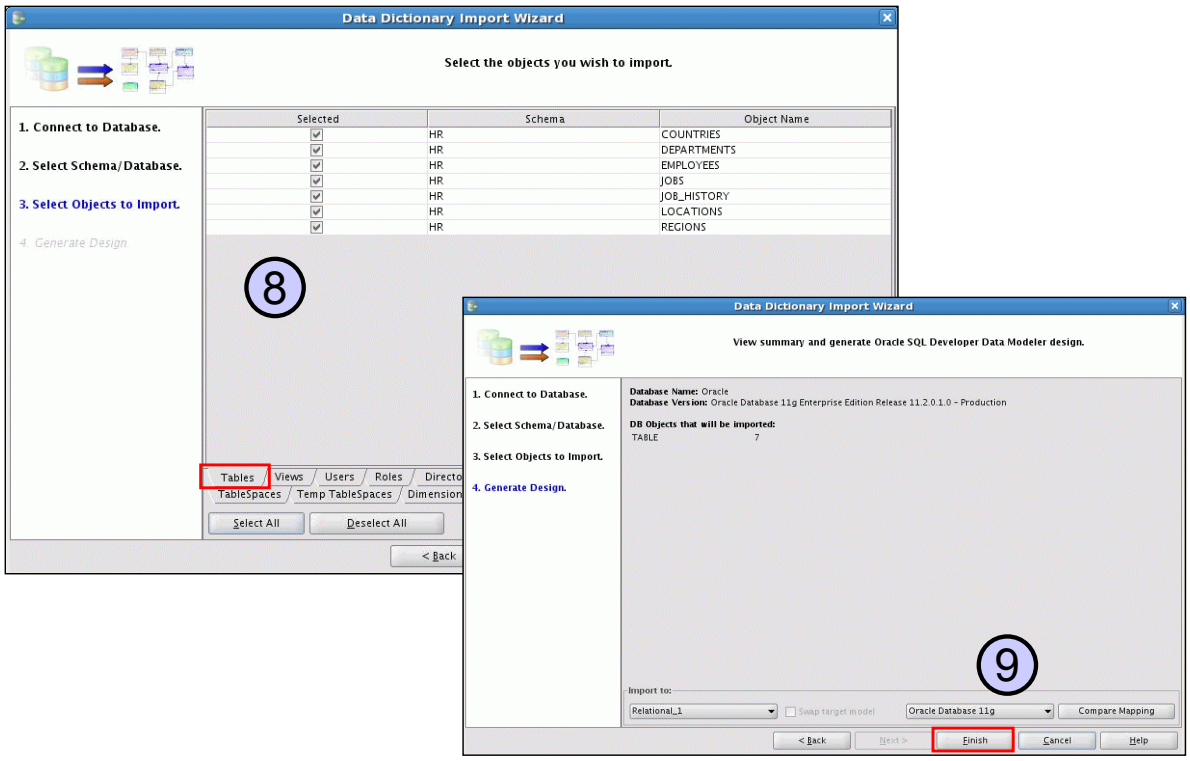
ORACLE

Importing an Existing Database (continued)

6. Select the connection that you just created, and click Next.
7. A list of schemas is displayed. Select the schemas that you want to import objects from, and click Next. The options available at the bottom of the screenshot in the slide include:
   - **All Selected:** Controls whether all items are initially selected or deselected when you move to the next page (Select Objects to Import).
   - **Secondary Tables:** Includes secondary tables in the metadata extraction
   - **Spatial Properties:** Includes Oracle Spatial properties in the metadata extraction

Note that if you do not have access to the schema, you will not see any objects for that schema (shown in the next slide). For example, if the connection that you created is based on the HR user who does not have access to the OE schema objects, you can select the OE schema but you will not see any objects for the OE schema. On the other hand, if you create a connection for the SYSTEM user, and then select both the HR and OE schemas, you will see the objects for both.

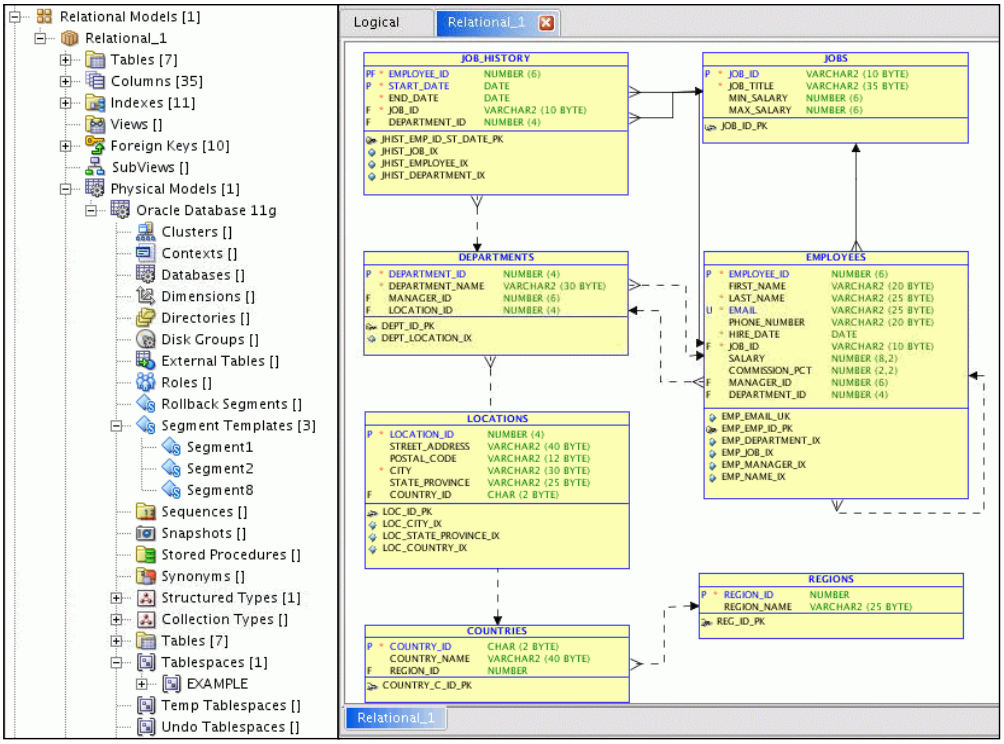
Importing an Existing Database



Importing an Existing Database (continued)

8. Select each object tab and select the objects that you want to import. Then click Next.
9. The object type and the number of objects that will be imported will be displayed. Click Finish.

Importing an Existing Database

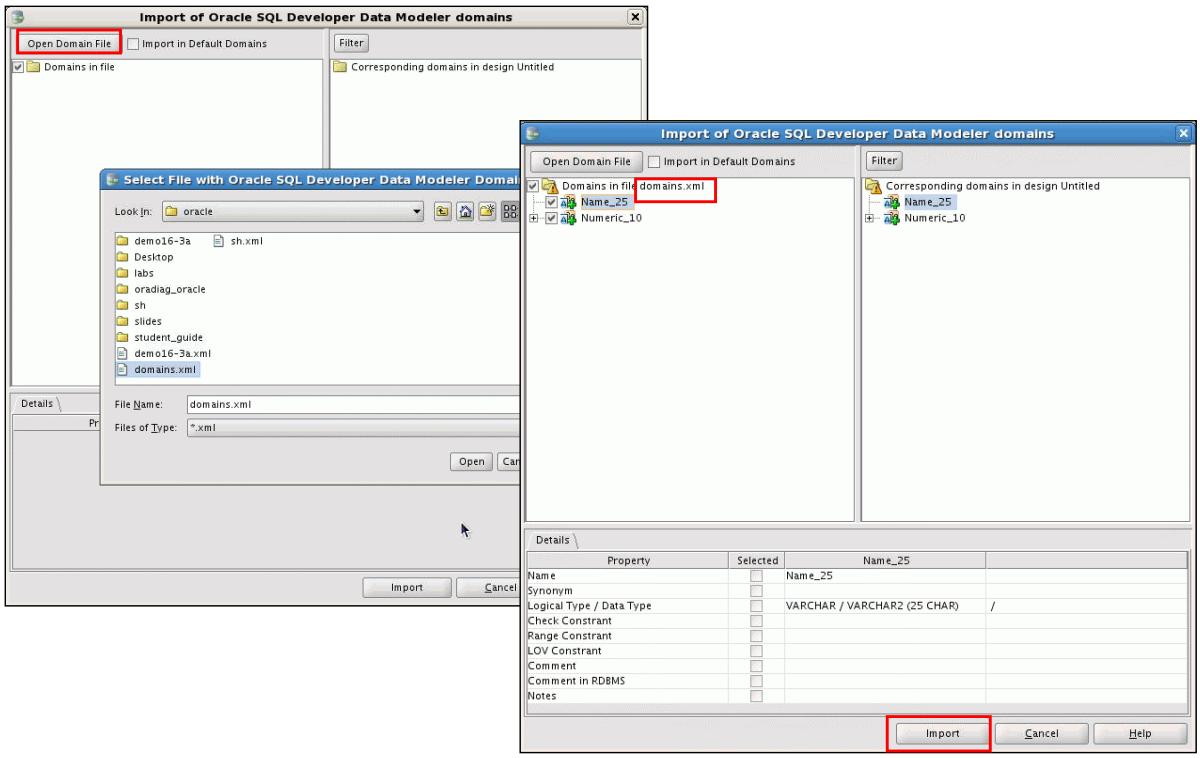


ORACLE

Importing an Existing Database (continued)

The relational model diagram is created. In the object browser, you see that the physical model was created and all the physical objects (such as Tablespace, and Segment Templates) from the database were created as well.

Importing Domains



ORACLE

Importing Domains

You can establish a standard domain file that you use to specify the data type for each column or attribute. The file is created from the Tools > Domains Administration page. To import the file, perform the following steps:

1. Select File > Import > Domains.
2. Click Open Domain File
3. Select the .xml file that has all the domains, and click Open.
4. You see the list of domains and a comparison of what will be created in the model you are importing them into. In the example in the slide, the untitled model is empty so the two domains will be created.
5. Click Import.

Quiz

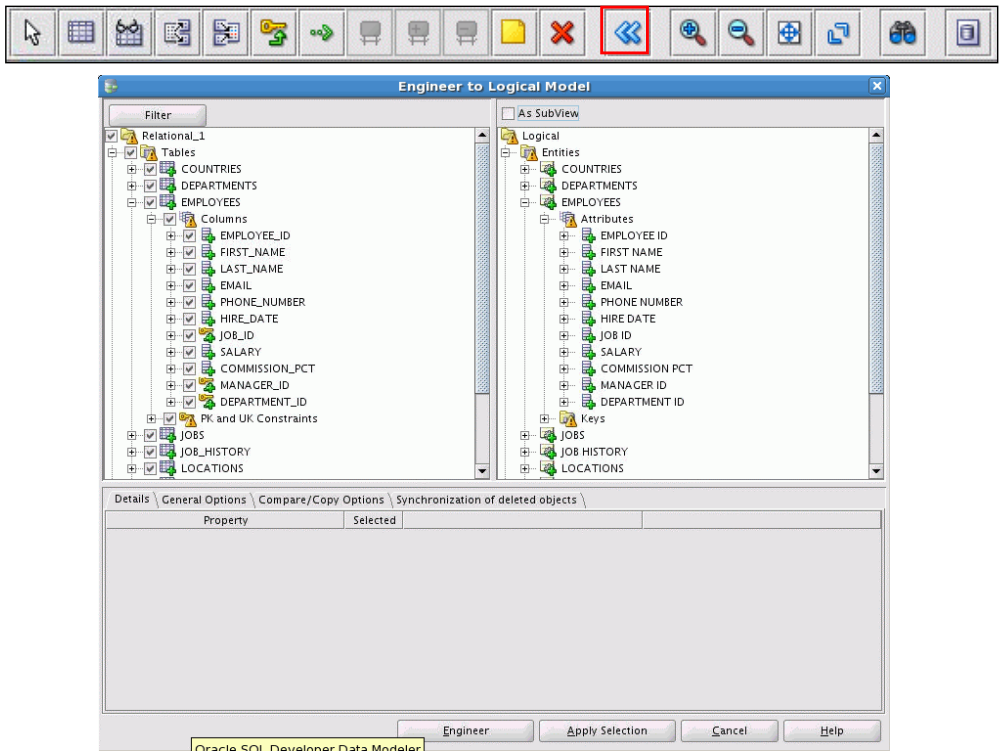
XYZ Company wants to make some changes to their existing Order Entry database. What should they do? (Select all that apply.)

a. Import the Data Modeler design.
b. Open the model in Oracle SQL Developer Data Modeler.
c. Import the DDL for the Order Entry database.
d. Import the Order Entry schema from the data dictionary.

ORACLE

Answer: c, d

Creating a Logical Data Model from Your Relational Model



Creating a Logical Data Model from Your Relational Model

After your relational model is created, you can make modifications to the relational model and/or reverse engineer to create a logical data model. Perform the following steps:

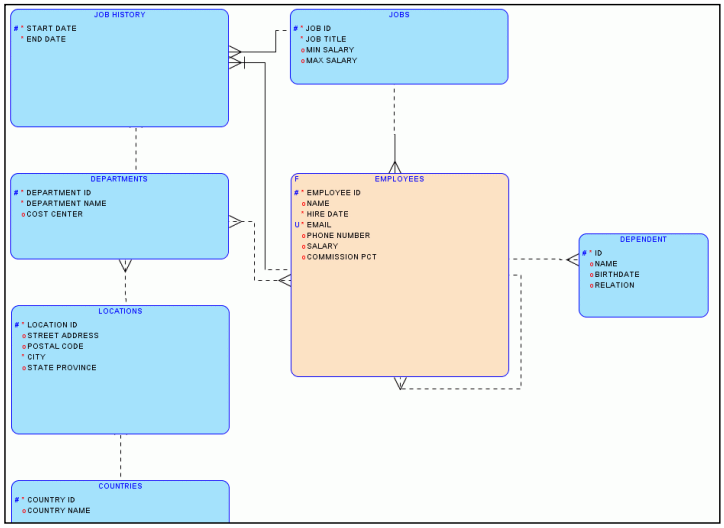
1. Select the Engineer icon in the toolbar. Notice that the arrows are pointing to the left (rather than to the right when you forward engineer).
2. The “Engineering to Logical Model” window appears. Expand the objects to review what will be engineered and then click Engineer.

Note that you can review what is going to happen when you click Engineer and if the translation is not correct, you can cancel, make changes to the relational model, and then rerun Engineer.

Reviewing and Making Changes to Your Logical Model

Example changes:

- Added a new DEPENDENT entity and a relationship with the EMPLOYEES entity
- Assigned a fact type to the EMPLOYEES entity
- Reordered the attributes in the EMPLOYEES entity
- Added a new COST CENTER attribute
- Changed FIRST_NAME and LAST_NAME to one NAME attribute



ORACLE

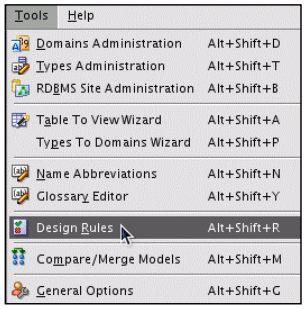
Reviewing and Making Changes to Your Logical Model

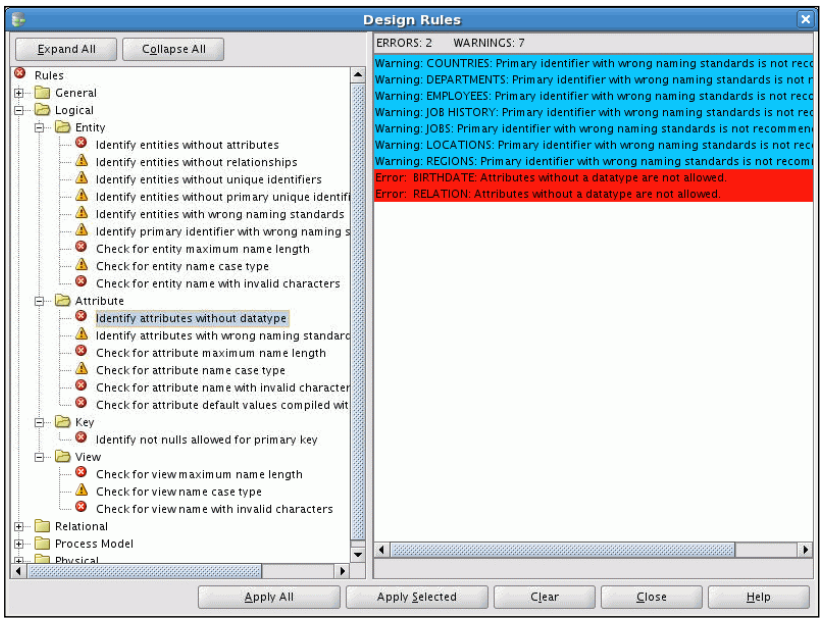
You can review your logical model for completeness and make any changes. In the example in the slide, the following changes were made to the logical model:

- A new DEPENDENT entity was created with attributes. In addition, a 1:M relationship was created between EMPLOYEES and DEPENDENT.
- The type FACT was assigned to the EMPLOYEES entity. As a result, the color of the entity changed and an F appears in the upper-left corner of the entity.
- The attributes were reordered in the EMPLOYEES entity.
- A new COST CENTER attribute was added to the DEPARTMENTS entity.
- A new NAME attribute was added to the EMPLOYEES entity, and the FIRST_NAME and LAST_NAME attributes were deleted.

Checking Design Rules

You must fix errors; warnings are OK.





ORACLE

Checking Design Rules

Before you synchronize your changes with the relational model, you should check to make sure you have no design rule errors. Warnings are OK.

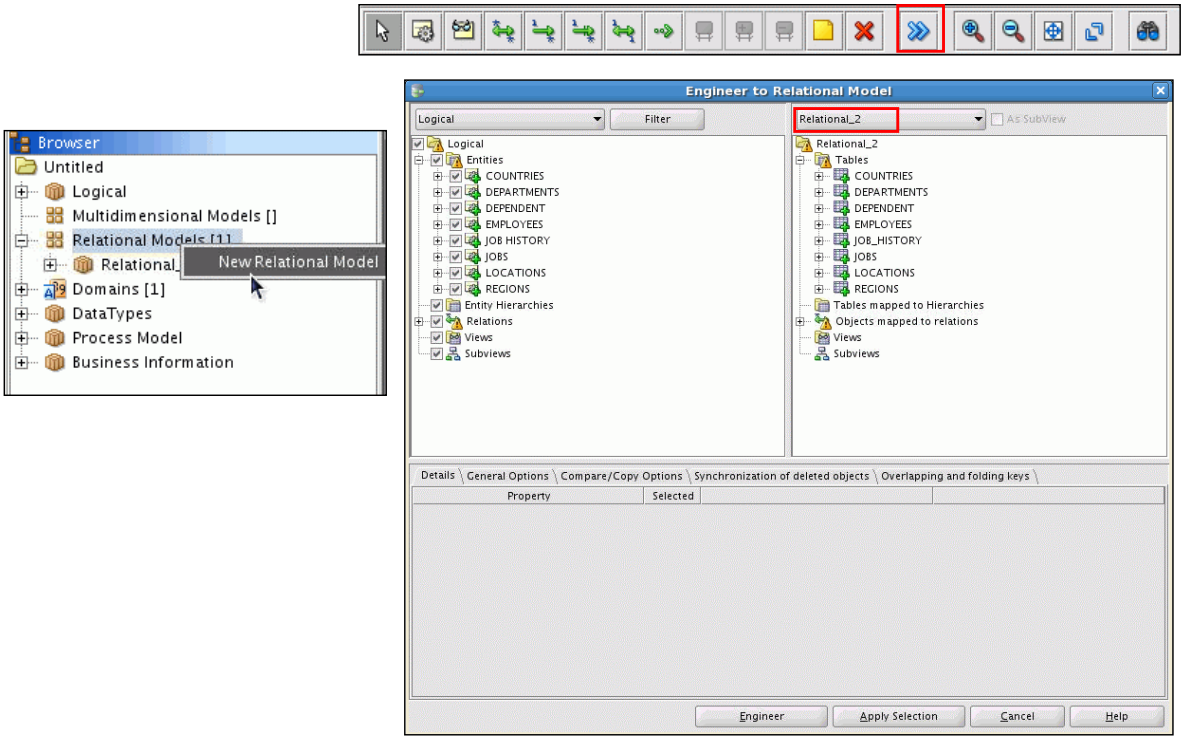
To check your design rules, perform the following:

1. Select Tools > Design Rules.
2. Select the Logical Rule and click Apply Selected.

You see the violations (warnings and errors on the right side of the window).

You may also expand each node and select the specific design rules that you want to run.

Forward Engineering to a New Relational Model



ORACLE

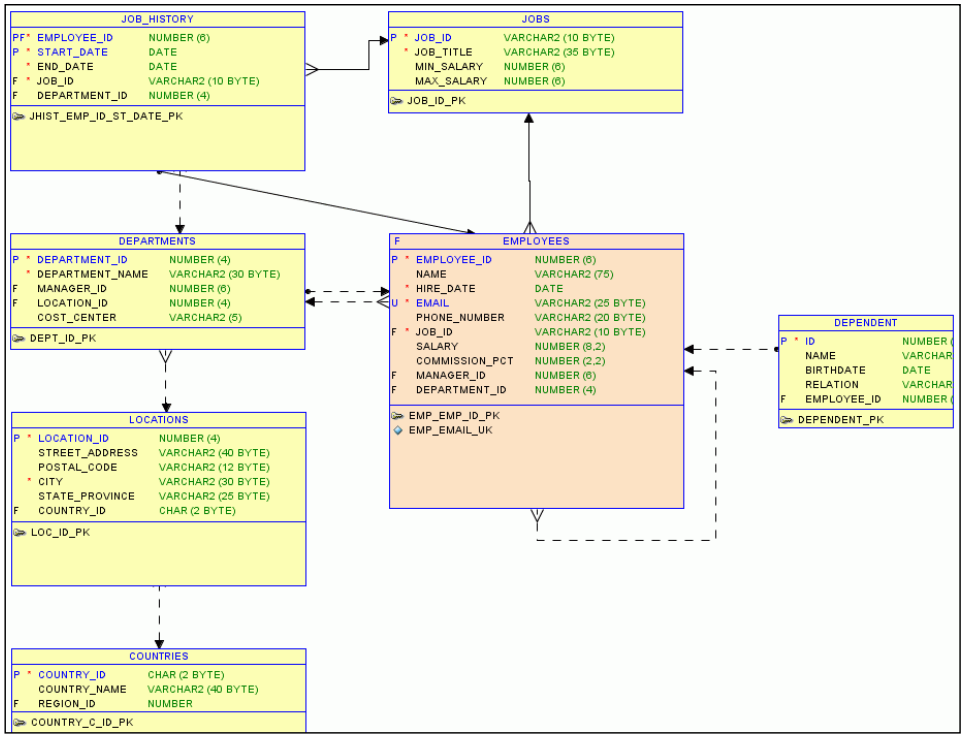
Forward Engineering to a New Relational Model

You can have multiple relational models for one logical model. If you do not want to overlay the existing relational model with the changes that you made to the logical model, you can create a new relational model and forward engineer the logical model to the new relational model. Perform the following steps:

1. In the object browser, right-click Relational Models and select New Relational Model.
2. The new relational model will appear in the list. Before you reverse engineer, click the Logical tab.
3. Click the Engineer icon.

   Notice that the relational model listed on the right side of the engineer window is relational_2. If you expand the nodes you see that all the objects will be created in the new relational model.
4. Click Engineer.

Forward Engineering to a New Relational Model



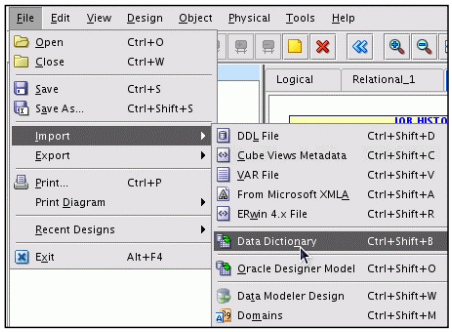
ORACLE

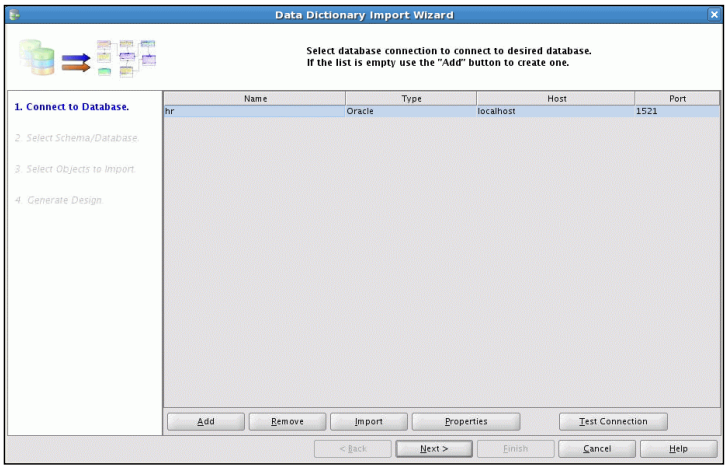
Forward Engineering to a New Relational Model (continued)

The relational model is displayed. Notice that all the changes from the logical model are now engineered into the relational model.

Comparing Your Relational Model Changes with What Is in the Database

Import from the data dictionary.





ORACLE

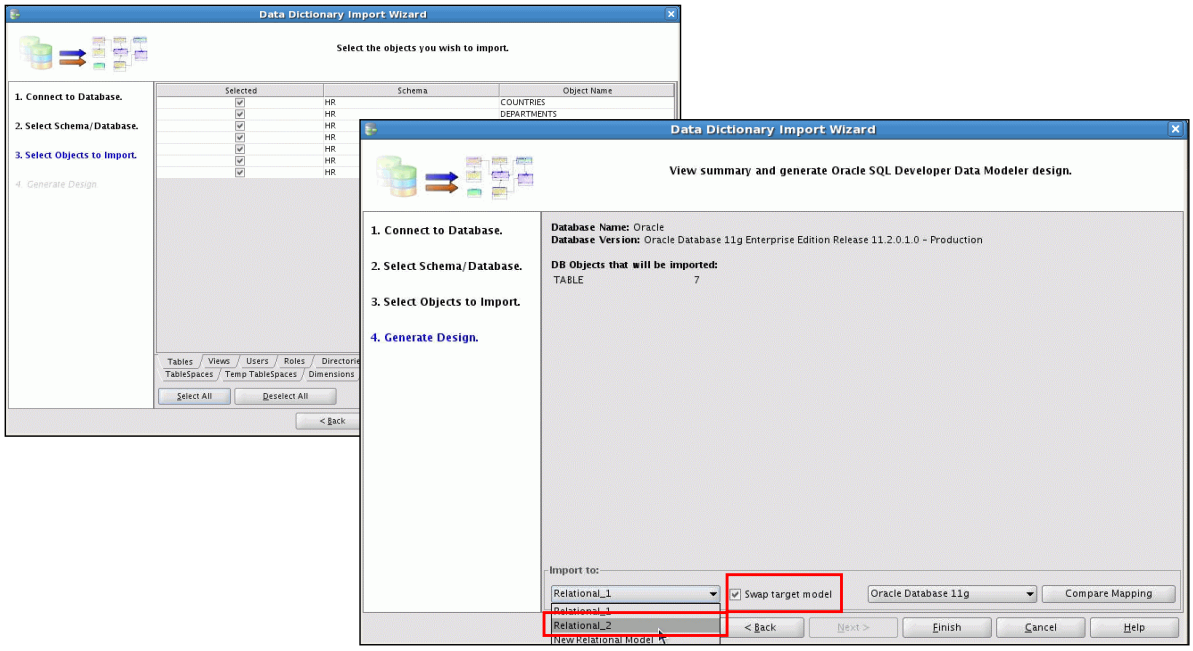
Comparing Your Relational Model Changes with What Is in the Database

At this point, it is useful to determine what has changed between the relational model and what is contained in your database. To do this comparison, perform the following steps:

1. Select File > Import > Data Dictionary.
2. Select the connection that you created previously from the list, and click Next.

Comparing Your Relational Model Changes with What Is in the Database

Swap the target model, and select the modified relational model.

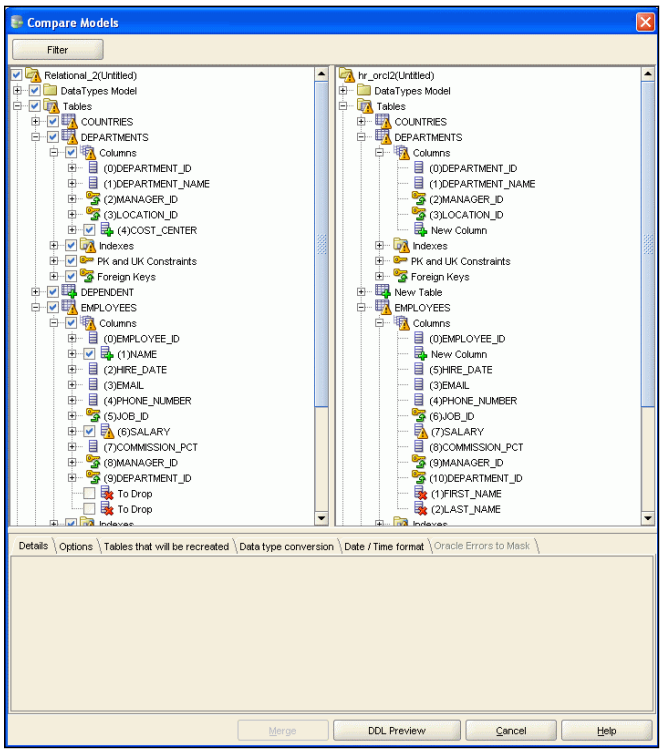


Comparing Your Relational Model Changes with What Is in the Database (continued)

1. Select the schema that contains the objects that you want to compare, and click Next.
2. Select the schema objects that you want to compare, and click Next.
3. In the Generate Design window (step 4 of the wizard), select the relational_2 model from the drop-down list. This is the changed model that you want to compare against.
4. Select the “Swap target model” check box to set the target model to be the database, the one you are importing.
5. Click Finish.

Comparing Your Relational Model Changes with What Is in the Database

- New `COST_CENTER` column in the `DEPARTMENTS` table
- New `DEPENDENT` table
- New `NAME` column in the `EMPLOYEES` table
- `FIRST_NAME` and `LAST_NAME` columns dropped from the `EMPLOYEES` table
- Columns reordered in the `EMPLOYEES` table



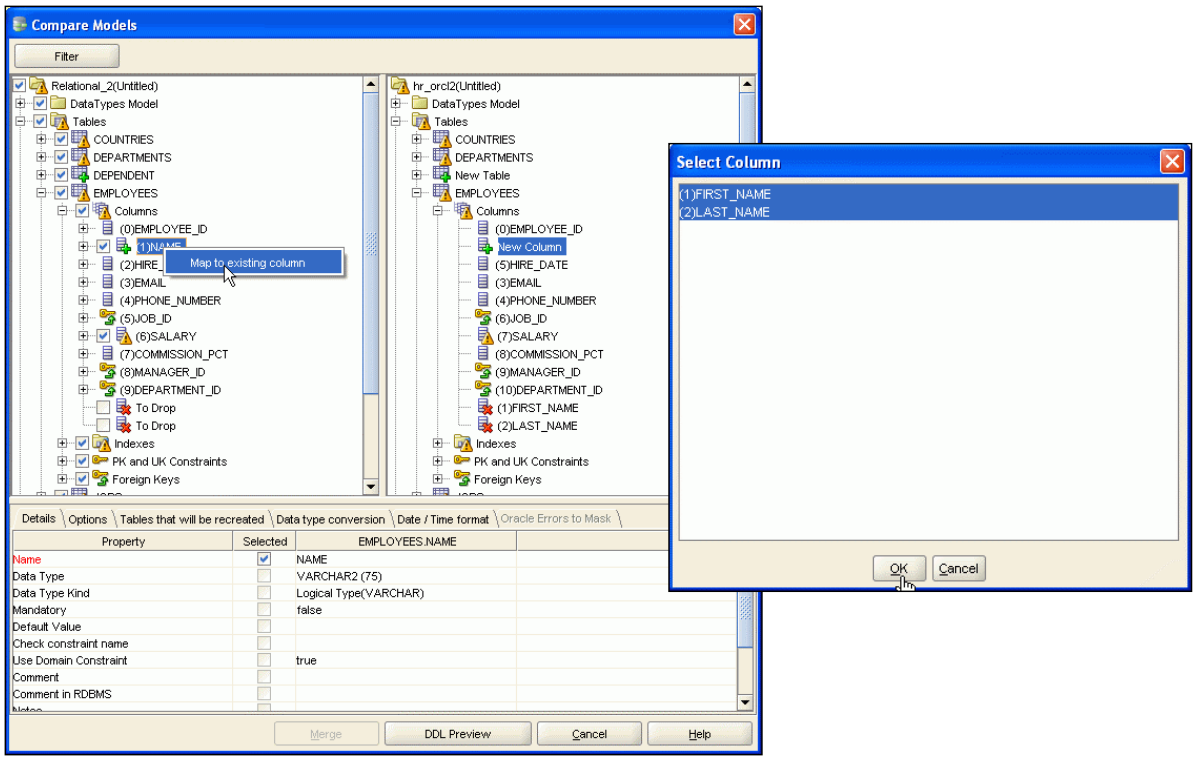
ORACLE

Comparing Your Relational Model Changes with What Is in the Database (continued)

The Compare Models window appears. You can now see what will happen to the database based on the changes that you made to the relational model.

- The COST_CENTER column will be added to the DEPARTMENTS table.
- A new DEPENDENT table will be created.
- A new NAME column in the EMPLOYEES table will replace the FIRST_NAME and LAST_NAME columns that were deleted.
- The columns in the EMPLOYEES table will be reordered so that HIRE_DATE appears after the new NAME column.

Mapping to an Existing Column

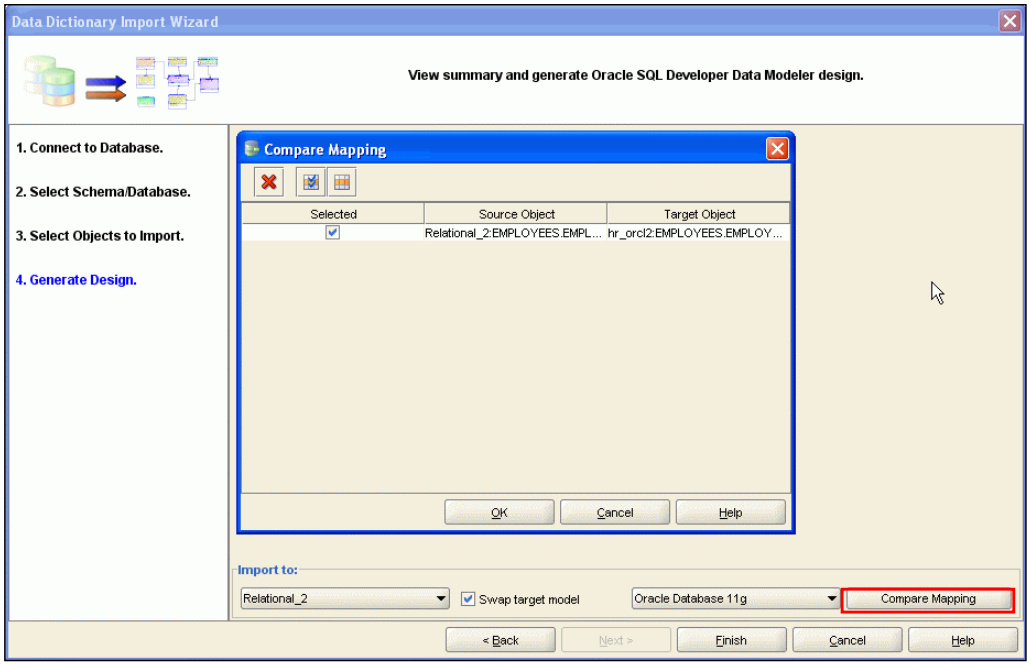


Mapping to an Existing Column

In the case where you have created a new column that replaces an existing column(s) in the target, you can map to the existing column. In the example in the slide, right-click the NAME column, and select “Map to existing column.” Select the columns that you want to map to, in this case, FIRST_NAME and LAST_NAME (which are the columns that will be dropped). Then click OK.

Note that the right-click in this window is currently not available in a Linux environment. This discrepancy has been logged to Oracle SQL Developer Data Modeler development and will be fixed in a future release. The current build for this class is release 2.0 with patch 1 build 584.

Compare Mapping

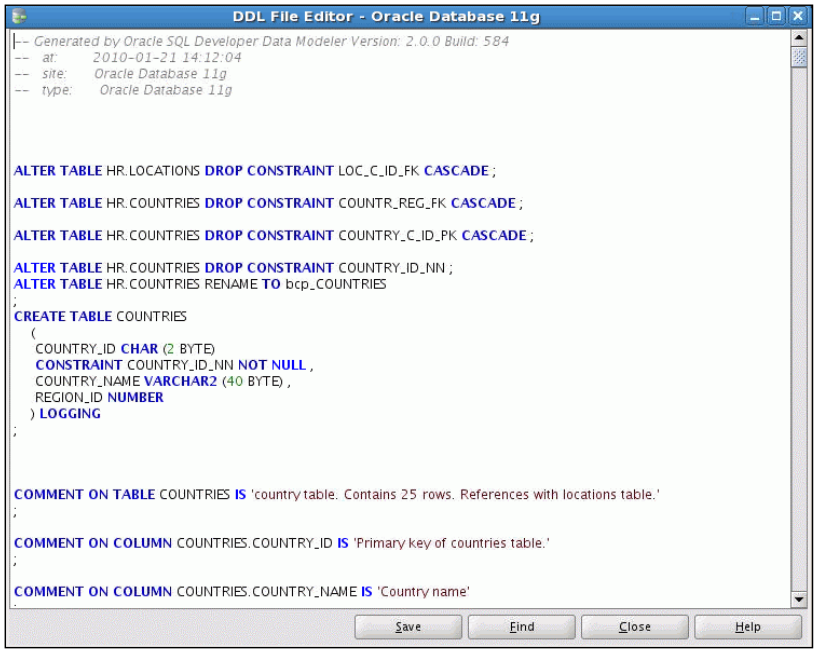


Compare Mapping

Because you have mapped the existing columns, when you run the comparison the next time and click Compare Mapping, the mapping will be displayed.

Previewing the DDL

ALTER statements are generated.

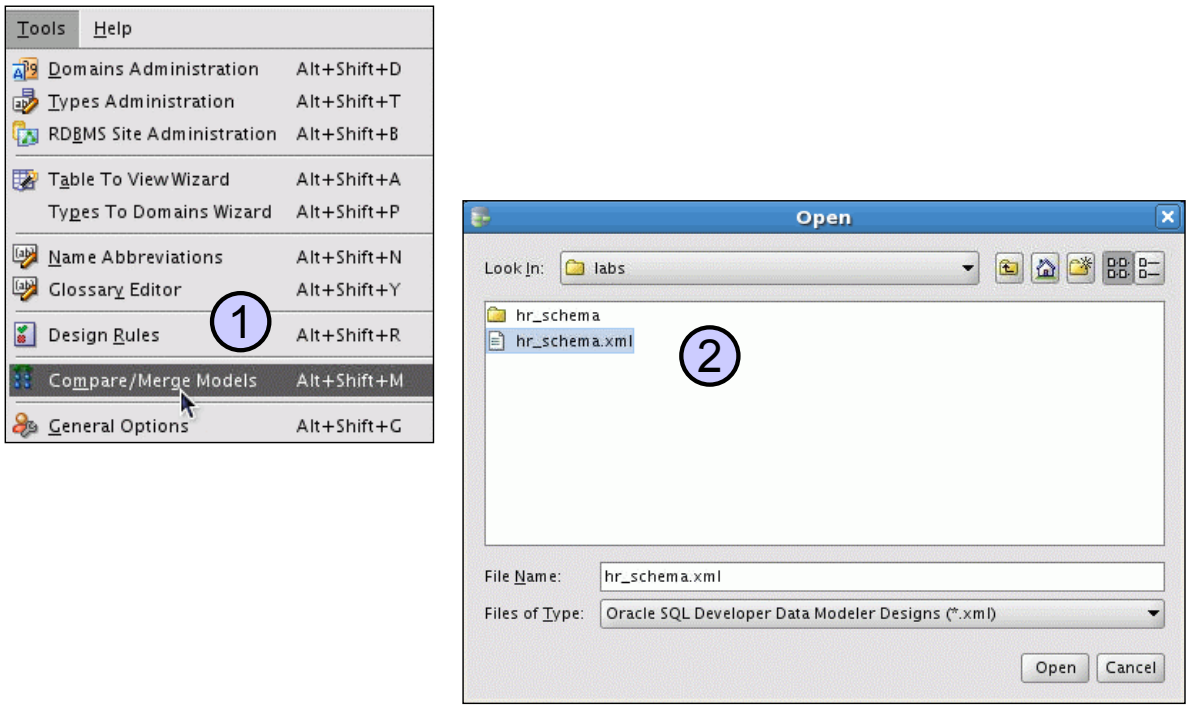


Previewing the DDL

In the Compare Models window, click the DDL Preview button to see the ALTER statements that will generate (when you generate the DDL script) to create the modifications to the database.

Note that the script will rename the current table, create the new table, and then insert the data from the renamed table into the new table.

Comparing and Merging Two Models



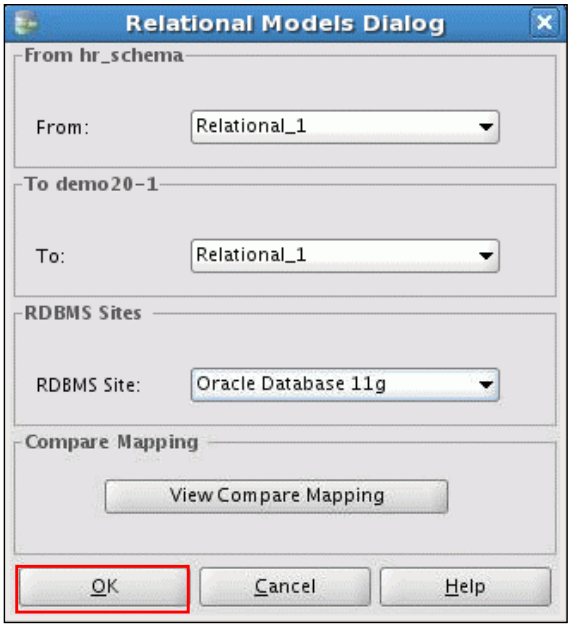
ORACLE

Comparing and Merging Two Models

You can compare an opened model with another model file. To compare and merge models by this method, perform the following steps:

1. Make sure that one of your models is opened. Then select Tools > Compare/Merge Models. The Open dialog box appears.
2. Select your model (another .xml file other than the one that is already open) and click Open.

Comparing and Merging Two Models



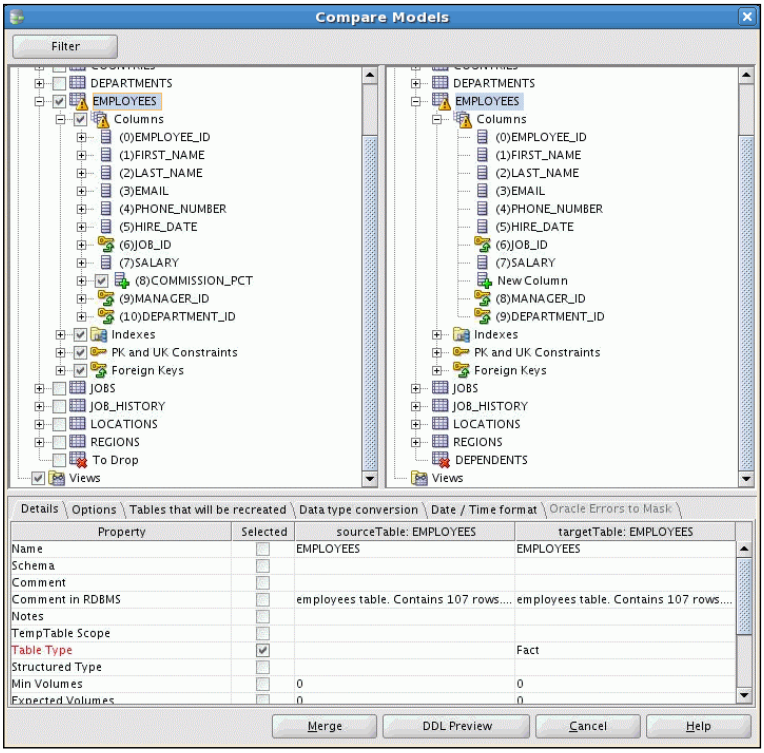
ORACLE

Comparing and Merging Two Models (continued)

The Relational Models Dialog dialog box appears. Notice that the second model that you select is in the From area and the model that you opened first appears in the To area. This means that the model you opened first is the one that will be the end result.

Select the RDBMS site from the RDBMS Site drop-down list, and click OK.

Comparing and Merging Two Models



ORACLE

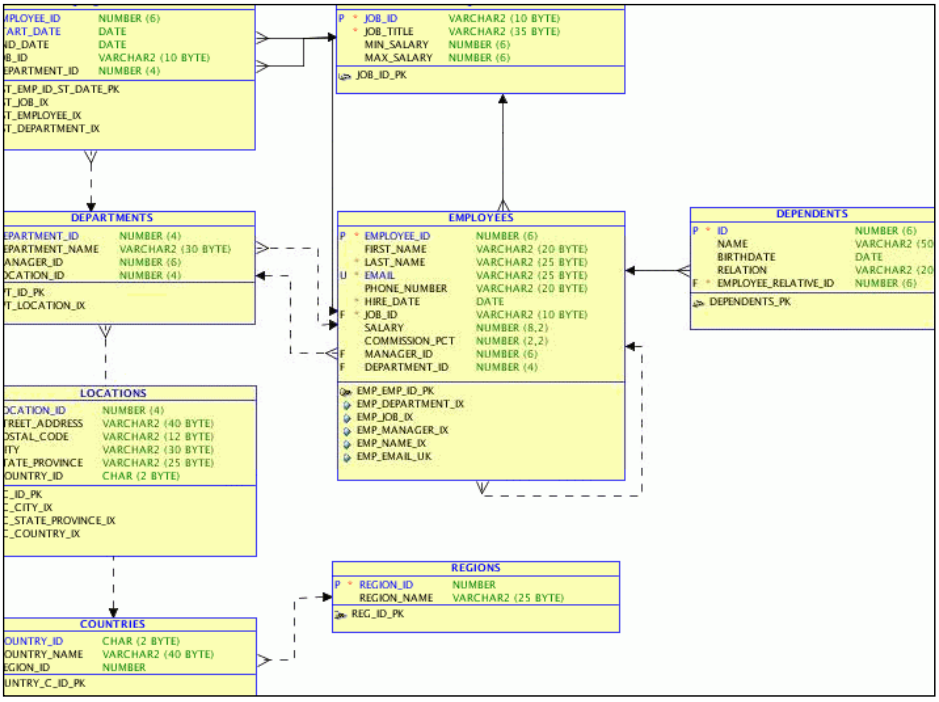
Comparing and Merging Two Models (continued)

The Compare Models window appears. Expand the relevant nodes to see what has changed. In the example in the slide, the following changes will be made:

- Some changes were made to the EMPLOYEES table properties. If you select the table in the list, you see the details below. Notice that the Fact type has been specified in the modified HR Schema model.
- A new column, COMMISSION_PCT, will be added to the modified HR schema. If you do not want this column added, you must deselect the check box next to the column on the left.
- A new table was added to the modified HR Schema model. If you do not want to add this table, you must select the To Drop check box on the left and it will be removed from the merged model. It is deselected by default.

When you are done analyzing the comparison, you can merge the models by selecting Merge.

Comparing and Merging Two Models

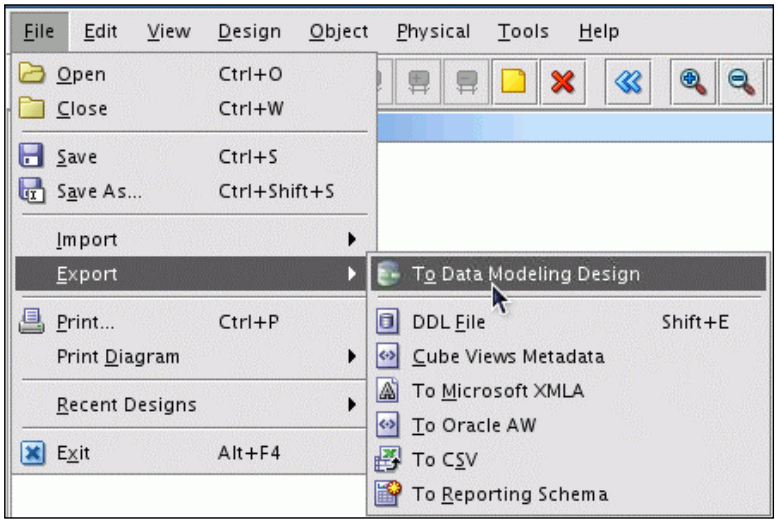


ORACLE

Comparing and Merging Two Models (continued)

Notice that the COMMISSION_PCT column was added back into the modified HR Schema model because you did not deselect the check box to add it back. The table that was added in the modified HR Schema model is still available because you did not check the check box for To Drop.

Exporting Your Model



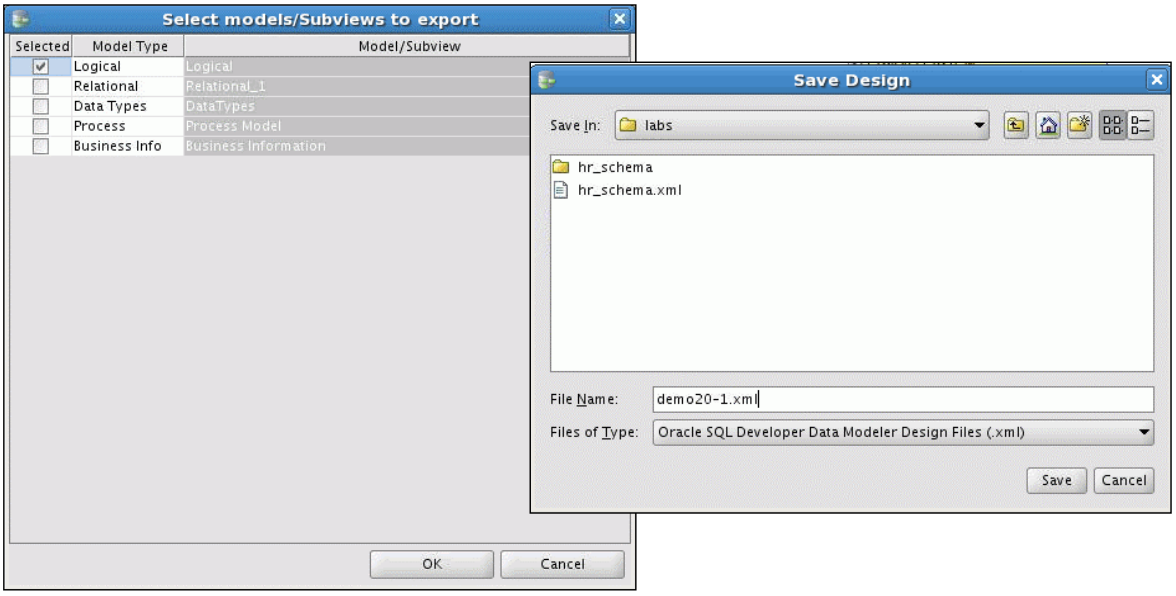
ORACLE

Exporting Your Model

There are a number of ways to export objects in your model. These include:

- **To Data Modeling Design:** Is discussed in detail in this lesson
- **DDL File:** Displays the same window as when you click the Generate DDL icon from the toolbar.
- **Cube Views Metadata:** Creates an XML file of your multidimensional model
- **To Microsoft XMLA:** Creates a file in Microsoft XMLA format of your multidimensional model
- **To Oracle AW:** Creates a file of your relational model that can be used to import into Oracle Analytic Workspace Manager
- **To CSV:** Creates a CSV file that can be opened in Microsoft Excel
- **To Reporting Schema:** Loads model metadata into a schema user that can then be analyzed through a series of reports in Oracle SQL Developer

Exporting to a Data Modeling Design



Exporting to a Data Modeling Design

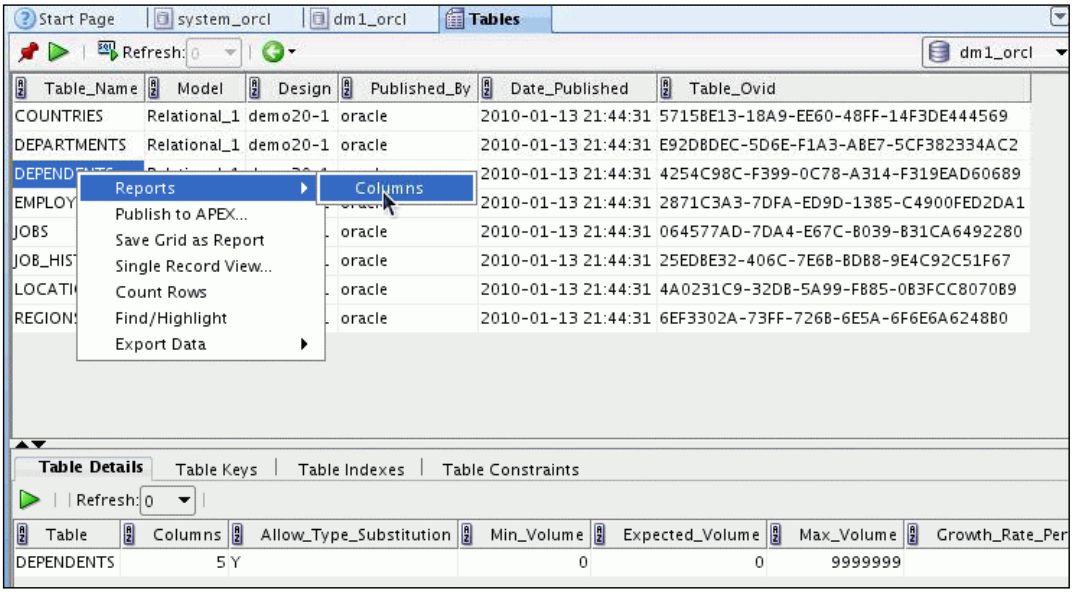
If you want to export only a portion of your design, perform the following steps:

1. Select File > Export > To Data Modeling Design.
2. Select check boxes for the model types that you want to export, and click OK.
3. Enter the name of the XML file that you want to save to.

Note that after the model is exported, you can open it in Oracle SQL Developer Data Modeler just like you would open any other model. The difference here is that only the exported models are included in the export file.

Producing Data Modeling Metadata Reports

Design content and rule reports in Oracle SQL Developer.



ORACLE

Producing Data Modeling Metadata Reports

Predefined data modeler reports are included with the latest release of Oracle SQL Developer. These reports allow you to report on the objects in your model, and they also report design rule violations. In order to utilize the data modeler reports, you need to export your model to a reporting schema in Oracle SQL Developer Data Modeler and then run the reports within Oracle SQL Developer.

Steps to Produce Data Modeler Reports

Oracle SQL Developer

1. Create a SYSTEM connection.
2. Create a new user for reporting.
3. Create connection for the new user.

Oracle SQL Developer Data Modeler

1. Open the model.
2. Export to the reporting schema.

Oracle SQL Developer

1. Run the data modeler reports.

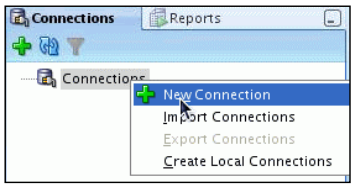
Steps to Produce Data Modeler Reports

You must perform a few tasks before you can do an export:

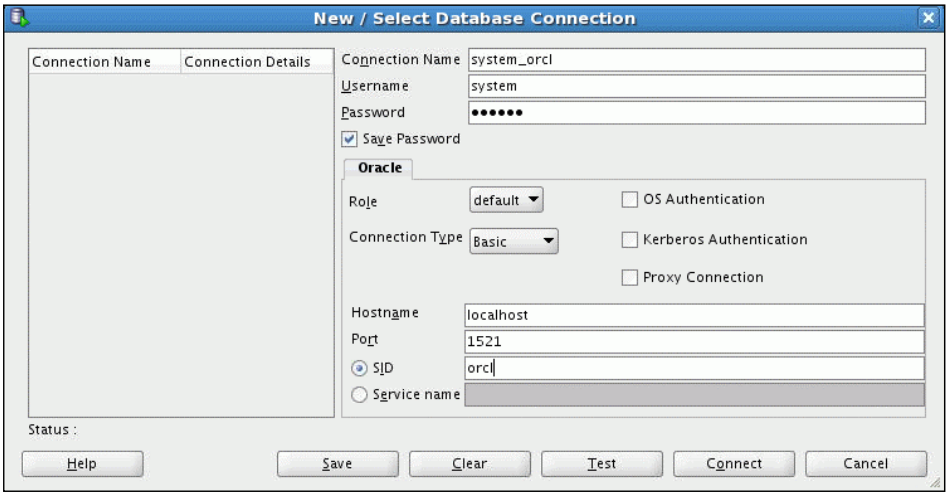
A. Create a new user for reporting purposes in Oracle SQL Developer.
B. Export your model to a reporting schema connected as the user you created.
C. Run the Data Modeler reports available in Oracle SQL Developer.

The next few slides guide you through the process.

Creating a SYSTEM Connection



Tool: Oracle SQL Developer



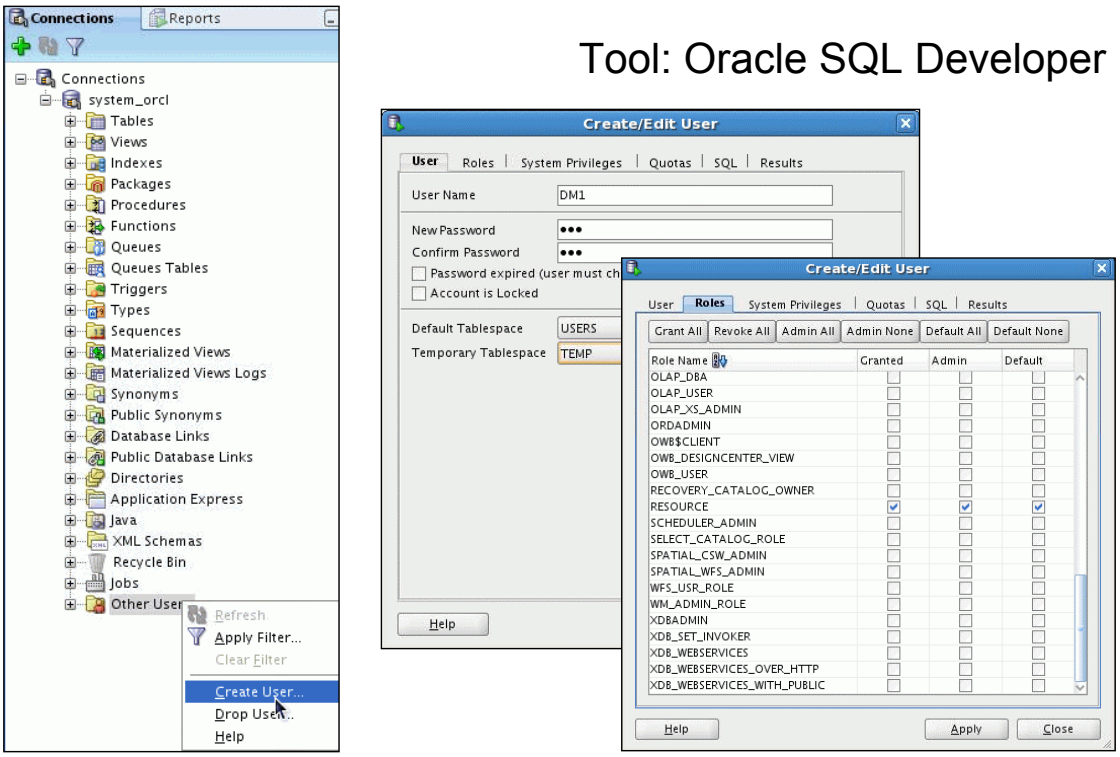
ORACLE

Creating a SYSTEM Connection

This task is performed in Oracle SQL Developer. Create a connection to a user that has the privilege to create additional users. To create a SYSTEM connection, perform the following steps:

1. In Oracle SQL Developer, right-click Connection and select New Connection.
2. Enter the following information and click Connection.
   - Connection Name: `system_<sid>`
   - Username: `system`
   - Password: `<your system password>`
   - Hostname: `<your hostname>`
   - SID: `<your SID>`

Creating a New User for Reporting

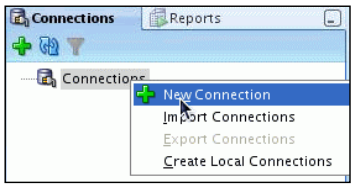


Creating a New User for Reporting

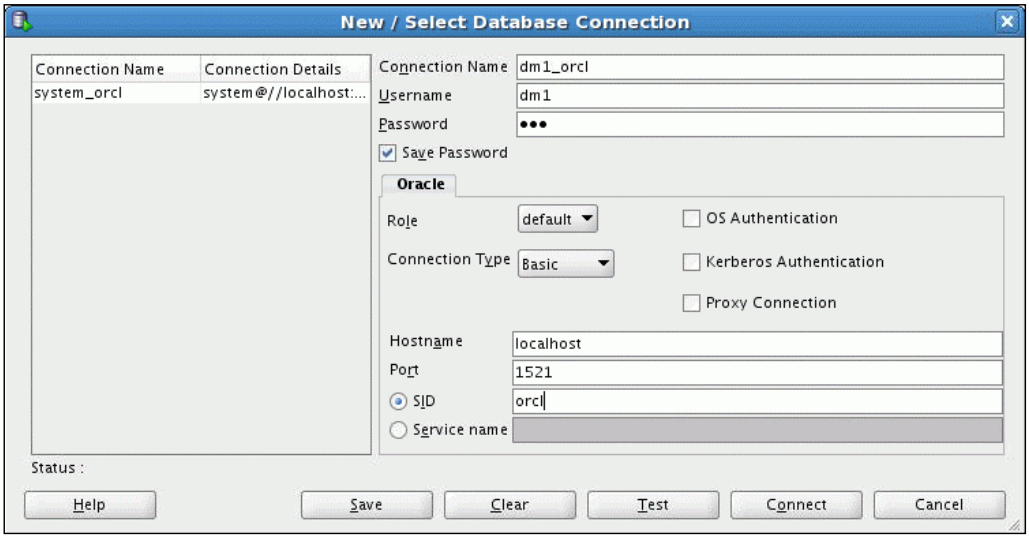
Next create a new user that you will use for reporting purposes. You could use an existing user; however, there will be additional database objects created when you export to the reporting schema and this may not be desirable in an existing user schema. Perform the following steps:

1. Expand your system connection.
2. Right-click Other Users and select Create User.
3. Enter a User Name and Password (in this case DM1), specify a default Tablespace and Temporary Tablespace, and click the Roles tab.
4. Select the Connect and Resource Roles and click System Privileges.
5. Select the Create View privilege and click Apply.
6. After the user is created, click Close.

Creating a Connection for the New Reporting User



Tool: Oracle SQL Developer



ORACLE

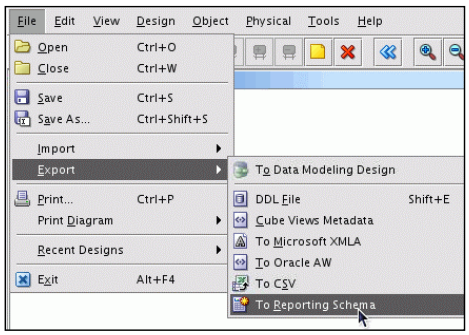
Creating a Connection for the New Reporting User

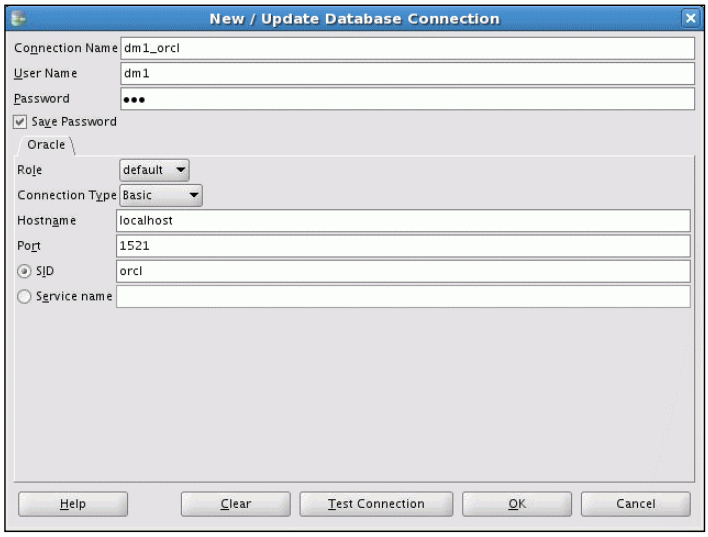
Next you create a connection for the new reporting user that you just created. This is the connection that you will use to run the reports after you export your model from Oracle SQL Developer Data Modeler. Perform the following steps:

1. In Oracle SQL Developer, right-click Connections and select New Connection.
2. Enter the following information and click Connect:
   - Connection Name: `<dmuser>_<sid>`
   - Username: `<dmuser>`
   - Password: `<your dmuser password>`
   - Hostname: `<your hostname>`
   - SID: `<your SID>`

Exporting Your Model to the Reporting Schema

Tool: Oracle SQL Developer Data Modeler



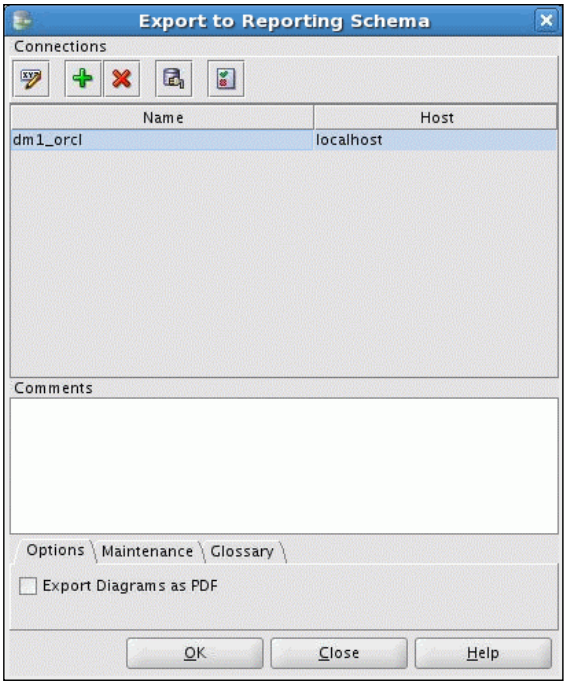


Exporting You Model to the Reporting Schema

Now you are ready to export your model to the reporting schema in Oracle SQL Developer Data Modeler. Perform the following steps:

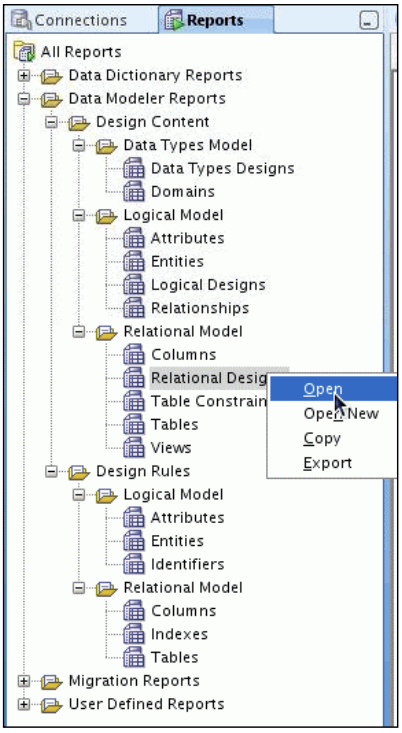
1. In Oracle SQL Developer Data Modeler, open the model that you want to export.
2. Select File > Export and select To Reporting Schema.
3. The “Export to Reporting Schema” dialog box appears. Click the Add ‘+’ icon.
4. Enter the following information and click OK.
   - Connection Name: `<dmuser>_<sid>`
   - Username: `<dmuser>`
   - Password: `<your dmuser password>`
   - Hostname: `<your hostname>`
   - SID: `<your SID>`

Exporting Your Model to the Reporting Schema



Tool: Oracle SQL Developer Data Modeler

Running Data Modeler Reports



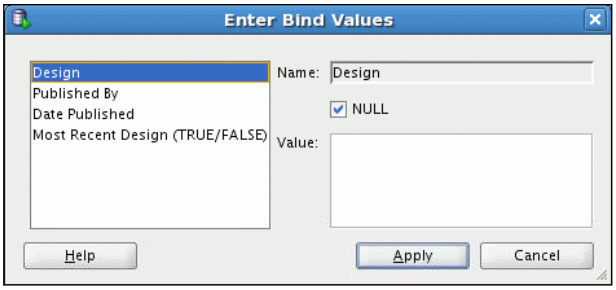
Tool: Oracle SQL Developer

Running Data Modeler Reports

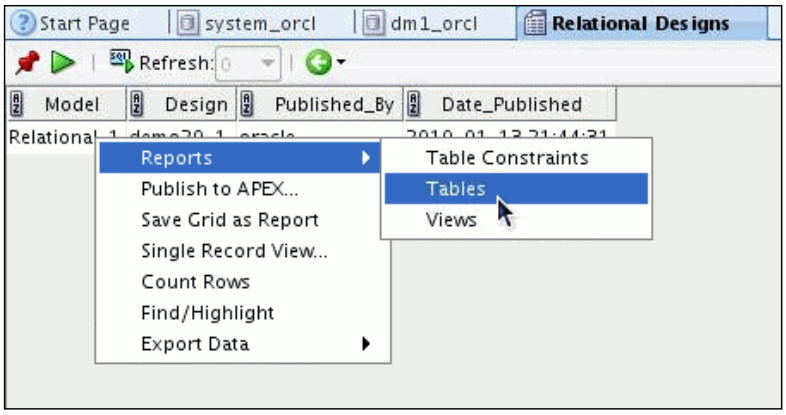
You are now ready to run the data modeler reports in Oracle SQL Developer. Under Design Content, you can run a series of reports that show information about the data types model, logical model, or relational model. You can also view Design Rules reports that show you objects that have violated a series of rules contained in the logical or relational model (shown in the screenshot in the slide). Perform the following steps:

1. In Oracle SQL Developer, click the Reports tab.
2. Expand Data Modeler Reports > Design Content and select Relational Designs, or right-click Relational Designs and select Open.
3. Select the connection that you created for the reporting user and click OK.
   The Enter Bind Values dialog box appears

Running Data Modeler Reports



Tool: Oracle SQL Developer



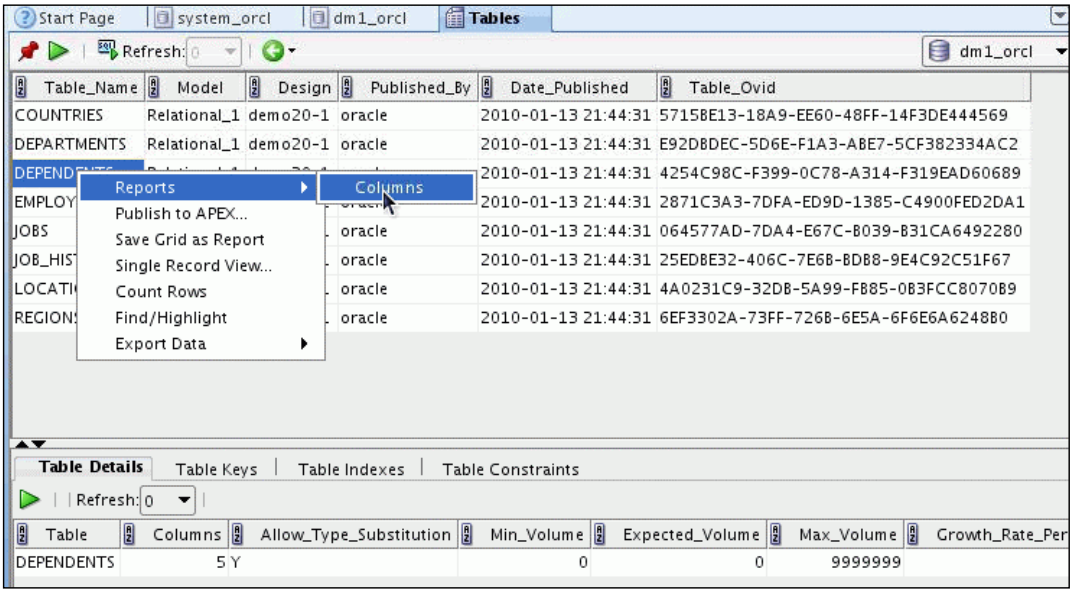
ORACLE

Running Data Modeler Reports (continued)

4. In the Enter Bind Values dialog box, you can change the results by specifying bind value criteria. For example, if you have exported your model more than once, you can see all the designs that you exported by changing the bind value for Most Recent Design to FALSE. The default is TRUE. Click Apply.
5. The list of designs is displayed. You can drill down to display the detail in the design. Right-click the model name and select Reports > Tables.

Running Data Modeler Reports

Tool: Oracle SQL Developer



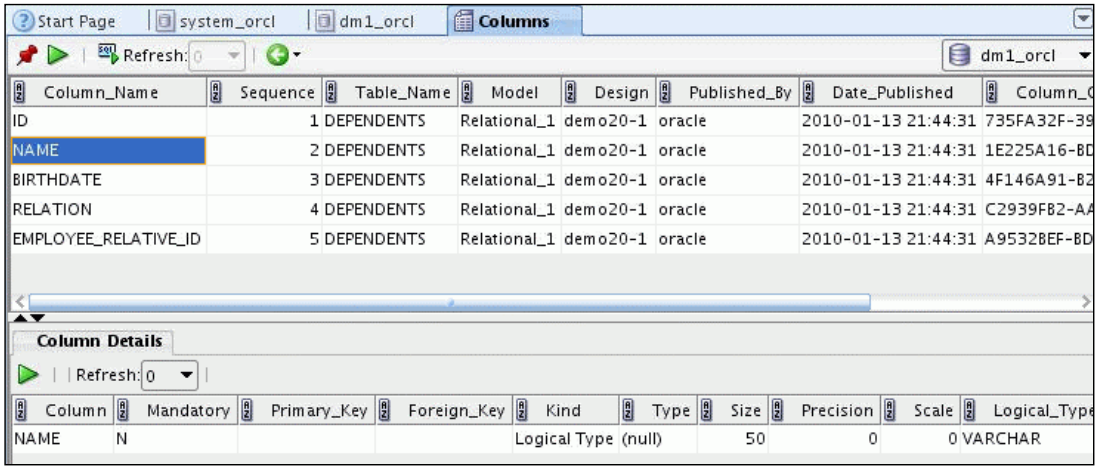
ORACLE

Running Data Modeler Reports (continued)

The list of tables is displayed. From the list of tables, you can drill down to see a list of columns. Notice that when you click a table, detailed information about that table, such as how many columns it has, is displayed in the bottom half of the window.

Running Data Modeler Reports

Tool: Oracle SQL Developer



Running Data Modeler Reports (continued)

The list of columns is displayed. If you click a column, column details are displayed in the bottom half of the window, such as the column's data type and length (size).

Quiz

Which method can you use to see design issues in the relational model? (Select all that apply.)

a. Running the design rules in Oracle SQL Developer Data Modeler
b. Exporting the DDL File
c. Running the Design Rules report in Oracle SQL Developer
d. Reviewing the Relational Model diagram

ORACLE

Answer: a, b, c, d

Summary

In this lesson, you should have learned how to:

- Import from the data dictionary
- Reverse engineer to create the logical model
- Compare and merge models
- Export your model
- Analyze your model by running Data Modeler reports

ORACLE

Summary

In this lesson, you learned how to re-engineer and existing database using import, reverse engineer, compare/merge, export and generating data modeler reports.

Practice 20-1 Overview: Re-Engineer the HR Schema

This practice covers the following topics:

- Importing the HR schema from the data dictionary
- Reverse engineering the model to create a logical data model
- Making some changes to the logical data model
- Forward engineering the model to create a new relational model
- Generating some data modeler reports.

Practice 20-1 Overview: Import and Reverse Engineer the HR Schema

In this practice, you import the HR schema from the data dictionary, then reverse engineer to create the logical data model, make some changes to the logical data model, forward engineer to a new relational model, compare the changes with what is in the database and generate some Data Modeler reports in Oracle SQL Developer.

21

Creating a Multidimensional Model

Objectives

After completing this lesson, you should be able to:

- Describe each multidimensional object
- Import a model with dimensions
- Generate a multidimensional model
- Review and modify the relational model
- Export the multidimensional model to an Oracle AW

ORACLE

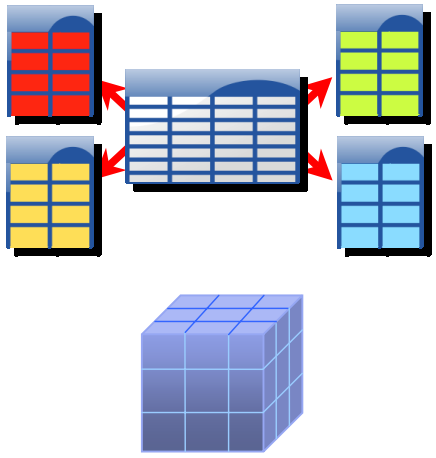
Objectives

In this lesson, you examine what each multidimensional object is and how to generate a multidimensional model by using an existing DDL file.

What Is a Multidimensional Model?

A multidimensional logical model has the following elements:

- Measures
- Dimensions
  - Hierarchies
  - Levels
  - Attributes



ORACLE

What Is a Multidimensional Model?

A multidimensional model can be implemented using tables or cubes.

Tables

In the case of tables, the dimensional model is typically implemented as a star or snowflake schema. Dimension tables (which contain information about hierarchies, levels, and attributes) join to one or more fact tables. Fact tables store quantifiable measures like sales, expenses, and inventory.

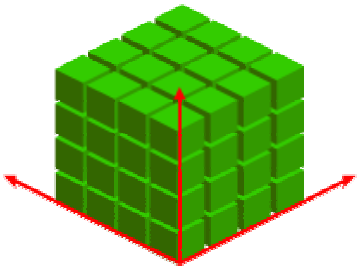
Cubes

Cubes logically represent data in a way similar to how tables do, although the data is actually stored in multidimensional arrays. Like dimension tables, cube dimensions organize members into hierarchies, levels, and attributes. The cube stores the fact data; the dimensions form the edges of the cube.

In this lesson, you will learn how to create a multidimensional model from an existing database.

Measures

- Represent factual data
- Are organized by one or more dimensions
- Populate the cells of a cube
- Can be numeric data, text, dates, Booleans, and so on



ORACLE

Measures

Measures represent factual data; they are sometimes called “facts.” Typical examples of measures are sales, cost, profit, and margin.

Measures are organized by one or more dimensions. (Dimensions are defined following this topic.) The image in the slide represents a generic measure that is organized by three dimensions.

Visualizing Measures

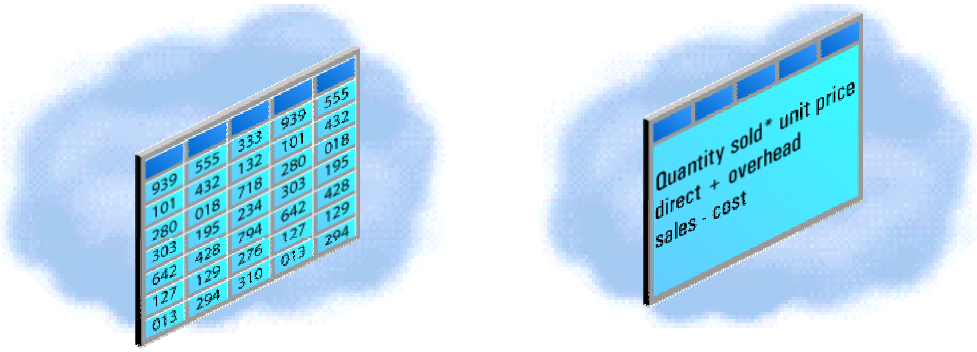
Many people visualize measures as being of a multidimensional shape, such as a cube, in which the edges of the shape are the dimensions and the contents of the shape are the measure values. In this context, the measure values populate the cells of a logical cube.

There is a tendency to draw three-dimensional (3D) shapes and refer to “cubes” (which are 3D) simply because they are easy to visualize. Of course, measures in a dimensional model can have 1, 2, 3, 4, 5, 6, 7, or many more dimensions.

Measure Types

Measures are of two types:

- *Stored* measures store the result in data cells.
- *Calculated* measures evaluate calculated data from a formula.



ORACLE

Measure Types

Measures may be stored, or calculated at query time.

Stored Measures

Stored measures are loaded and stored at the leaf level. Commonly, there is also a percentage of summary data that is stored. Summary data that is not stored is dynamically aggregated when queried.

Calculated Measures

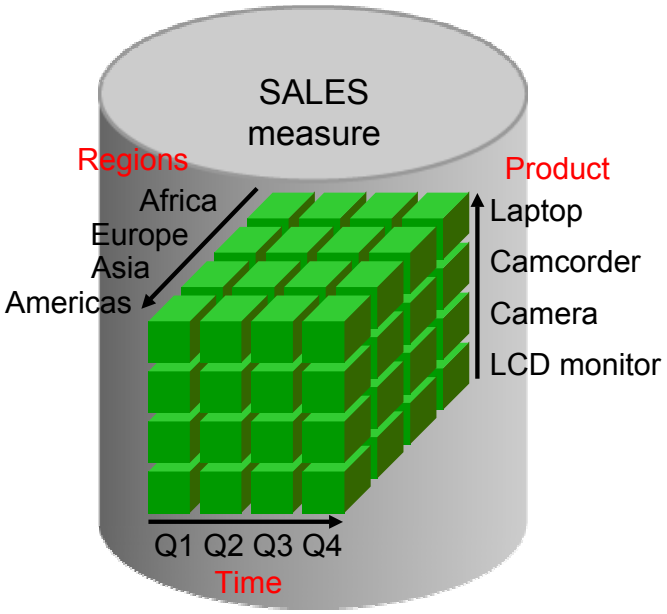
Calculated measures are measures whose values are calculated dynamically at query time. Only the calculation rules are stored in the database.

Common calculations include measures such as ratios, differences, moving totals, and averages. Calculations do not require disk storage space, and they do not extend the processing time required for data maintenance.

Dimensions

Dimensions:

- Form the “edges” of the measure
- Provide pointers to the actual cells inside the multidimensional measures



ORACLE

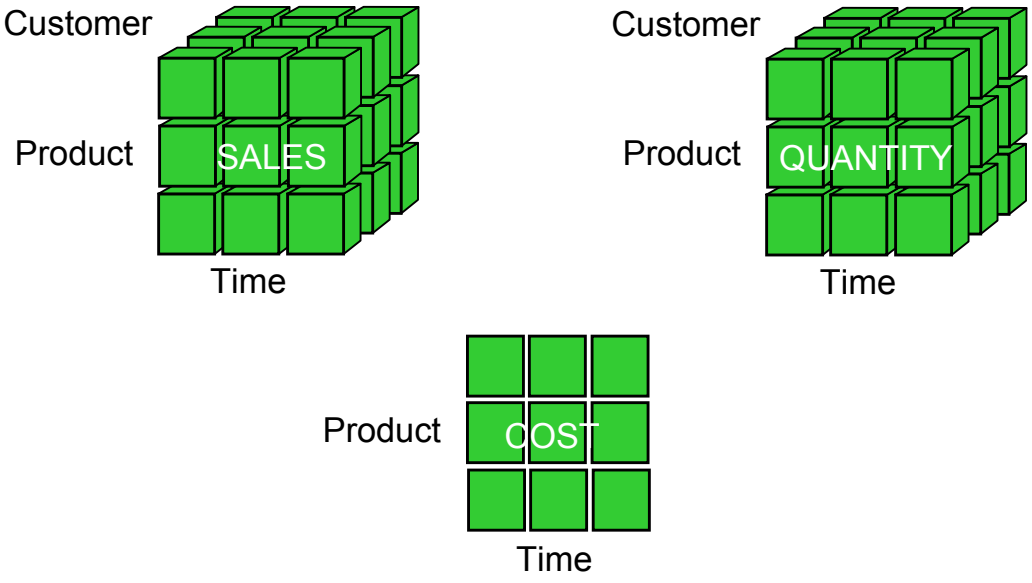
Dimensions

Dimensions provide meaning to the measure data. Dimensions are characterized by the following:

- They identify and categorize your measure data.
- They shape measures by forming the edges of the measures.

Examples of dimensions include product, geography, time, and distribution channel.

Sharing Dimensions



Sharing Dimensions

One of the characteristics of the dimensional model is that the dimensions and their members are shared by all measures, of whatever shape. In the example in the slide, there are three measures, organized in two ways:

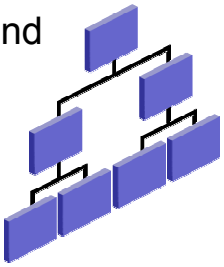
- Sales and Quantity (sold) have the same shape; they are both dimensioned by the Customer, Product, and Time dimensions.
- The Cost measure is dimensioned by Product and Time only. It does not use the Customer dimension because, in this example, the cost does not vary by customer. Therefore, the Customer dimension is not used by the Cost measure.

Dimensions are stored once and are used repeatedly. That is, there is just one Product dimension in this example, even though it appears three times in the diagram. This feature has many benefits. For example, with only one list of products to manage, a security scope may be placed on the product dimension, and all of its dependent objects are automatically scoped.

In addition, measures of different shapes that share dimensions may be combined to create new measures. The new measure is organized by the super-set of the dimensions. For example, a calculated measure named Profit may be created by combining Quantity, Cost, and Sales. The Profit measure would be dimensioned by the Customer, Product, and Time dimensions.

Hierarchy

- A *hierarchy* is a parent-child relationship between the members of a dimension.
- Hierarchies enable logical groupings of dimension members for the:
  - Navigation of data
  - Aggregation of measures
  - Allocation of data in a budgeting or planning application
  - Calculation of data, such as shares and indexes
- Dimensions can have more than one hierarchy.



ORACLE

Hierarchy

A hierarchy is a logical structure that groups like members of a dimension together for the purpose of analysis.

For example:

- A Time dimension might have a hierarchy that describes how months are grouped together to represent a quarter and how quarters are grouped together to represent a full year.
- An Organization dimension might have a hierarchy that makes it easy for you to identify the direct reports of a specific manager.

Each dimension can have multiple hierarchies if required. For example, the time dimension can have a hierarchy that represents the Julian calendar and another hierarchy that represents a fiscal calendar.

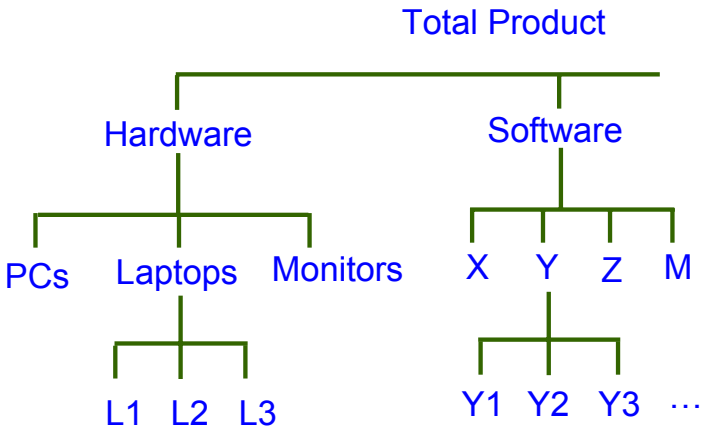
Hierarchy (continued)

A dimension's structure is organized hierarchically based on parent-child relationships. These relationships enable:

- **Navigation between levels:** Hierarchies on dimensions enable drilling down to lower levels or navigating (rolling up) to higher levels. Drilling down on the Time dimension member 2005 will likely navigate you to the quarters Q1 2005 through Q4 2005. In a calendar year hierarchy, drilling down on Q1 2005 would navigate you to the months January 05 through March 05. These kinds of relationships make it easy for users to navigate large volumes of multidimensional data.
- **Aggregation from child values to parent values:** The parent represents the aggregation of its children. Data values at lower levels aggregate into data values at higher levels. Dimensions are structured hierarchically so that data at different levels of aggregation can be manipulated together efficiently for analysis and display.
- **Allocation from parent values to child values:** The reverse of aggregation is allocation and is heavily used by planning, budgeting, and similar applications. Here, the role of the hierarchy is to identify the children and descendants of particular dimension members for "top-down" allocation of budgets (among other uses).
- **Grouping of members for calculations:** Share and index calculations take advantage of hierarchical relationships (for example, the percentage of total profit contributed by each product, or the percentage share of product revenue for a certain category, or costs as a percentage of the geographical region for a retail location).

Hierarchy: Example

- Hierarchies enable you to navigate from the lowest level to the highest level, or from the highest to the lowest.
- You can aggregate data from the lowest level to the highest level.

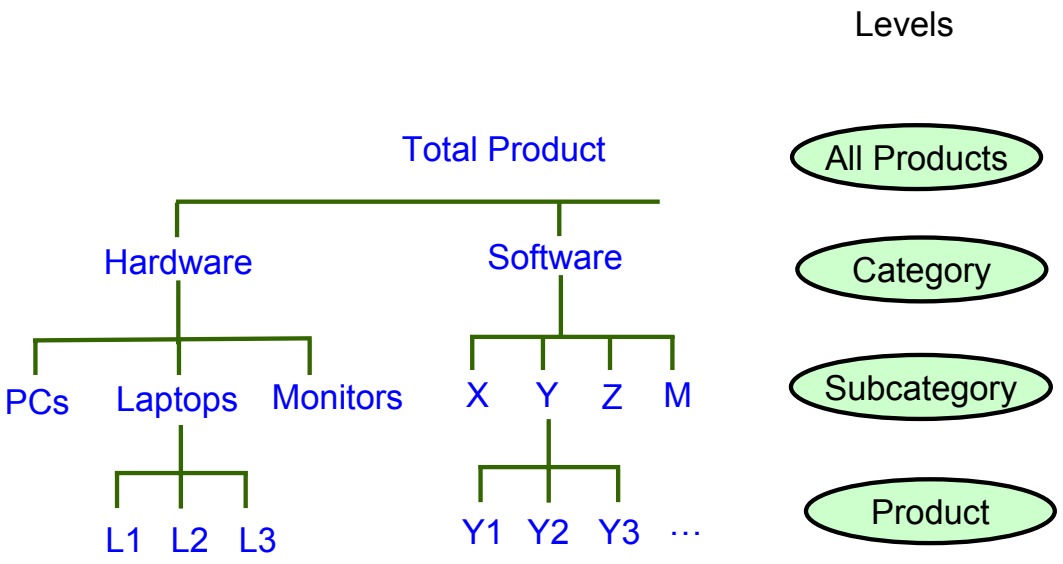


Hierarchy: Example

In this example, you can do the following in the Product hierarchy:

- Navigate up through each level in the hierarchy from the lowest level to the highest level
- Navigate down the hierarchy from the highest level to the lowest level
- Aggregate data from the lowest level (individual products) up through the hierarchy to the highest level (total product)

Level



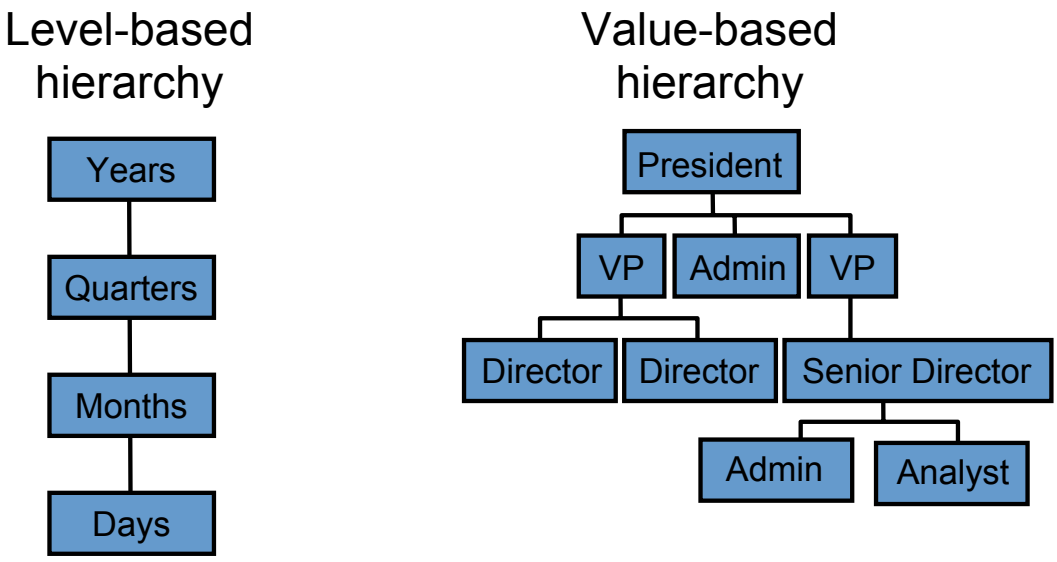
Level

Each level represents a position in the hierarchy. The members at different levels have a one-to-many parent-child relationship. A hierarchy typically contains several levels, and a single level can be included in more than one hierarchy.

If data for the Sales measure is stored at the Product level, the higher levels of the product dimension enable the sales data to be aggregated correctly into Subcategory, Category, and All Products levels.

If there are multiple hierarchies built over a dimension, it may be that a level would appear in more than one hierarchy or may exist in only one hierarchy.

Types of Hierarchy



Types of Hierarchy

Level-Based

Most of the hierarchies are level-based, including the Product dimension hierarchy shown in the previous slide and the Time dimension hierarchy shown in this slide. In the time hierarchy example, there are Day, Month, Quarter, and Year levels in the hierarchy.

Sales forces also have a level-based structure, as in the following example:

- Representative > Area > Region > Country > Continent > World

Value-Based

Other dimensions may have hierarchies that are not strictly level-based. Typical examples include Organizational, Financial, Cost Center, and Line Item hierarchies.

For example, there is clearly a hierarchy in an organization chart, but all the direct reports of the President may not be at the same level. In the example, the two VPs (vice presidents) and the President's Admin (administrative assistant) are all direct reports of the President but are not at the same level. The VPs are not at the Admin level, and the Admin is not at the VP level.

Note: In Oracle SQL Developer Data Modeler, value-based hierarchies are referred to as "ragged hierarchy links."

Attributes

- Attributes provide descriptive information about the dimension members.
- Attributes are also useful when you are selecting dimension members for analysis:
  - Select all time periods whose level is MONTH and whose Year description contains CY2007 (all months in the calendar year 2007).
  - Select the products whose size is Medium.
  - Select the customers who have two or more children.
  - Select the promotions that are of type Multipack.

ORACLE

Attributes

Dimensions may also have attributes, which are used to provide more information about members of the dimension.

Description

Unless dimension-member IDs themselves are meaningful to end users, it is usually the case that each dimension has “description” attributes that store the user-visible identifiers for the dimension members.

Data Selection

Attributes are also useful when filtering that dimension for analysis. They can be used for data selection.

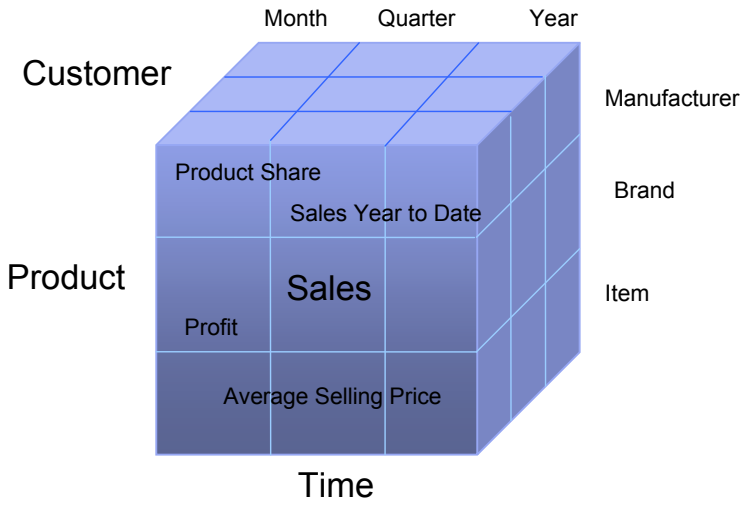
How Are Attributes Applied?

- Some attributes are valid for all the members of the dimension, regardless of level. For example, all products at all levels have a description.
- Other attributes are valid for certain levels or certain hierarchies only. For example, only individual product items have a color.

Dimensional Model Summarized

The multidimensional model has the following elements:

- Measures
- Dimensions:
  - Hierarchies
  - Levels
  - Attributes



ORACLE

Dimensional Model Summarized

In summary, the dimensional model is essentially made up of measures and dimensions. Measures contain or calculate data, and dimensions organize data.

Dimensions are mandatory in a dimensional model—if you do not have dimensions, you cannot have measures. Dimensions are what describe measures: they are fundamental to dimensional analysis.

Dimensions may contain the following elements:

- Dimensions optionally have hierarchies, which are logical structures that group like members of a dimension together for the purposes of analysis, aggregation, or allocation.
- Hierarchies may or may not have levels, because some hierarchies are not level-based.
- Dimensions may also have attributes, which are used to provide more information about members of the dimension. Attributes are useful when filtering that dimension for analysis.

Quiz

Which of the following objects populates the cells of a cube?

a. Hierarchy
b. Measures
c. Level
d. Dimension

Answer: b

Quiz

Which of the following are examples of a dimension? (Select all that apply.)

a. Sales
b. Product
c. Time
d. Month

ORACLE

Answer: b, c

Steps to Build a Multidimensional Model in Oracle SQL Developer Data Modeler

1. Import an existing database to create a relational model.
2. Reverse engineer to create a logical model.
3. Engineer from an Oracle model to create the multidimensional model.
4. Review and modify the multidimensional model.

ORACLE

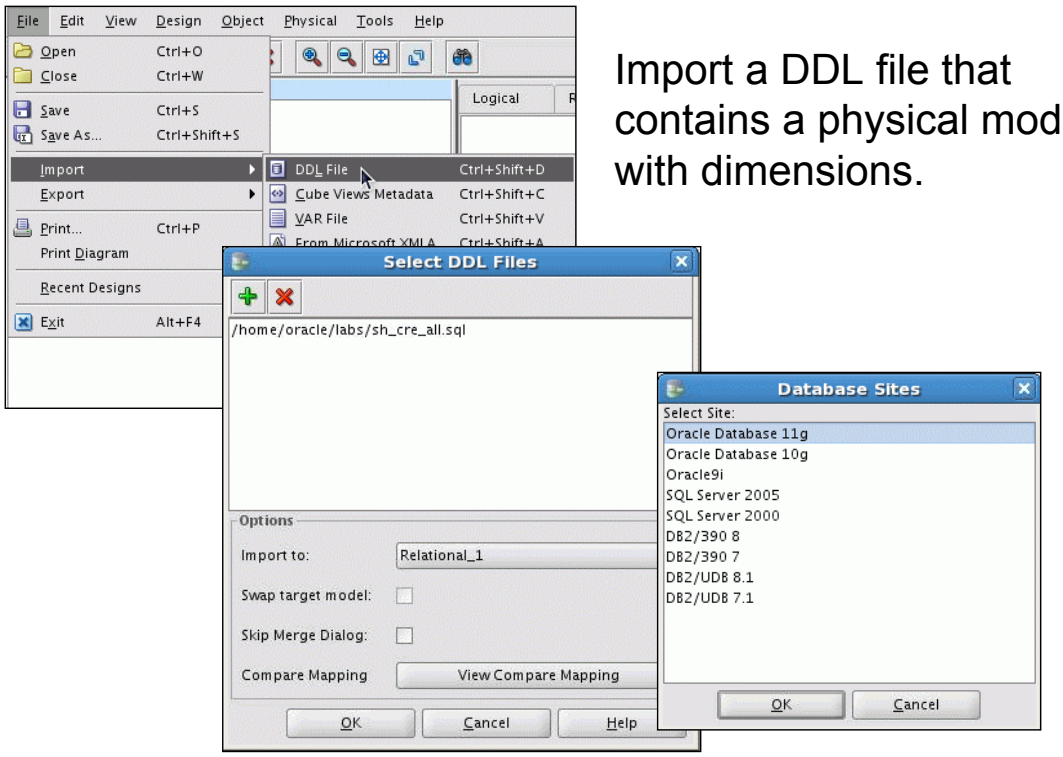
Steps to Build a Multidimensional Model in Oracle SQL Developer Data Modeler

To build a multidimensional model from an existing database or DDL file, you need to perform the following steps:

1. Create a relational model by importing from the data dictionary or importing a DDL file. You want to make sure that the physical model contains dimensions that will be used to create the multidimensional model.
2. Reverse engineer the relational model to create a logical model.
3. Create a multidimensional model and engineer it from an Oracle Model.
4. Review and modify the multidimensional model that is created.
5. Export the model to your environment of choice.

The next set of slides will walk through each of these steps.

Importing a Database with Dimensions



Import a DDL file that contains a physical model with dimensions.

ORACLE

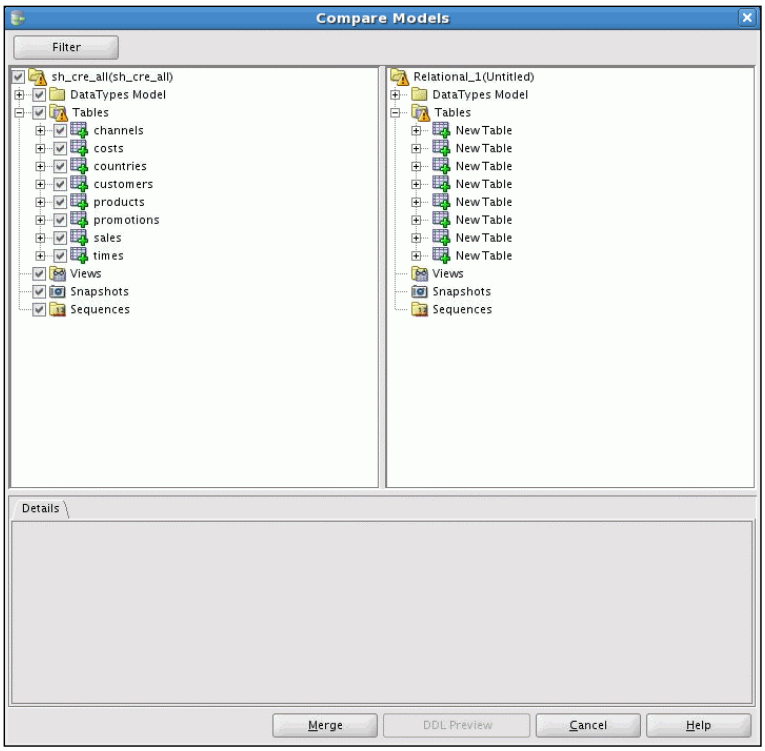
Importing a Database with Dimensions

If you already have a database with dimensions, you can import it from the data dictionary or from a DDL file. Perform the following steps:

1. Select File > Import > DDL File.
2. Click the Add '+' icon.
3. Select the SQL file with your DDL and click Open.
4. Click OK.
5. Select the database site and click OK.

Importing a Database with Dimensions

Objects will be merged into an empty model.

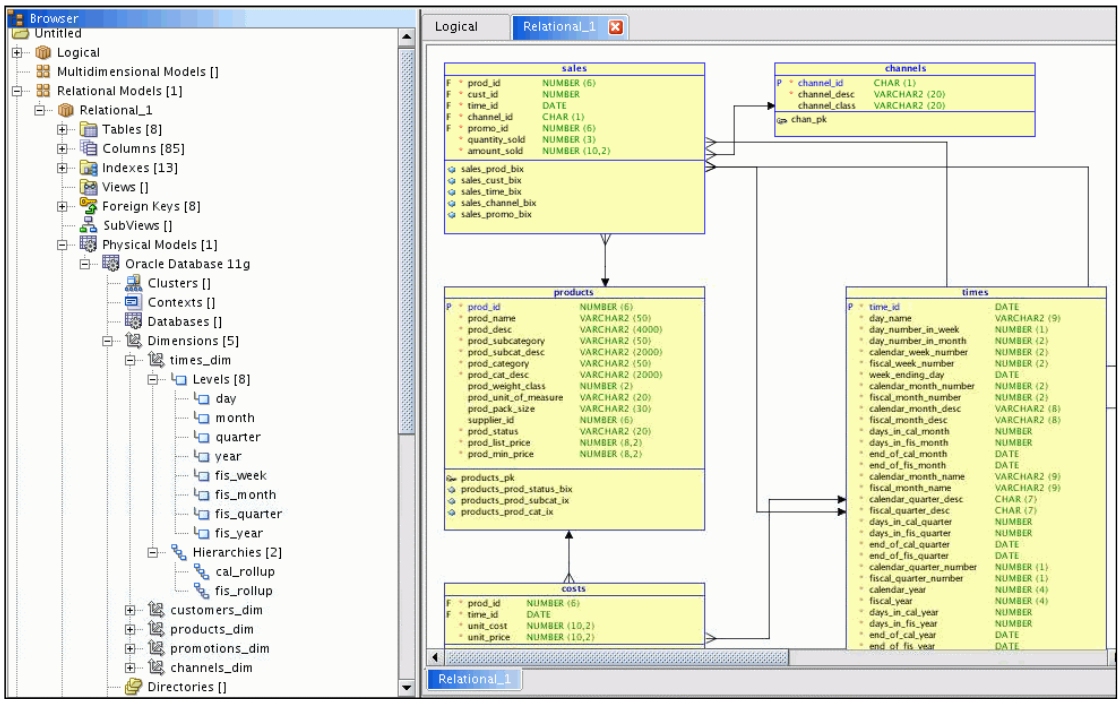


ORACLE

Importing a Database with Dimensions (continued)

The Compare Models window appears. If you expand one of the object types, you see what will happen when you click Merge. In the example in the slide, the tables on the left will be created in the empty model when you click Merge.

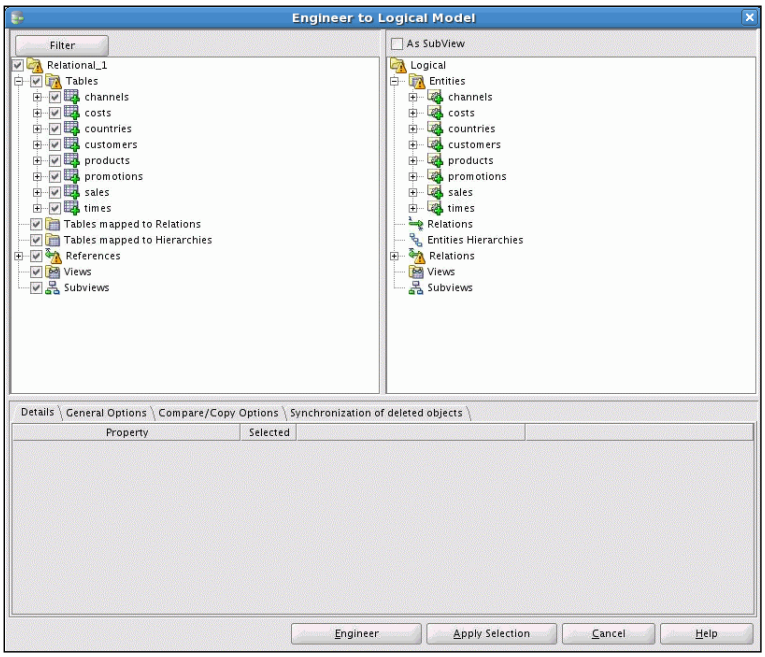
Importing a Database with Dimensions



Importing a Database with Dimensions (continued)

The relational model is created. To see the multidimensional database objects, in the object browser, expand the relational model and then the physical model. In the screenshot, you see the list of the dimensions and all the objects underneath that node. At this point you can generate a multidimensional model based on these objects.

Reverse Engineering Your Model

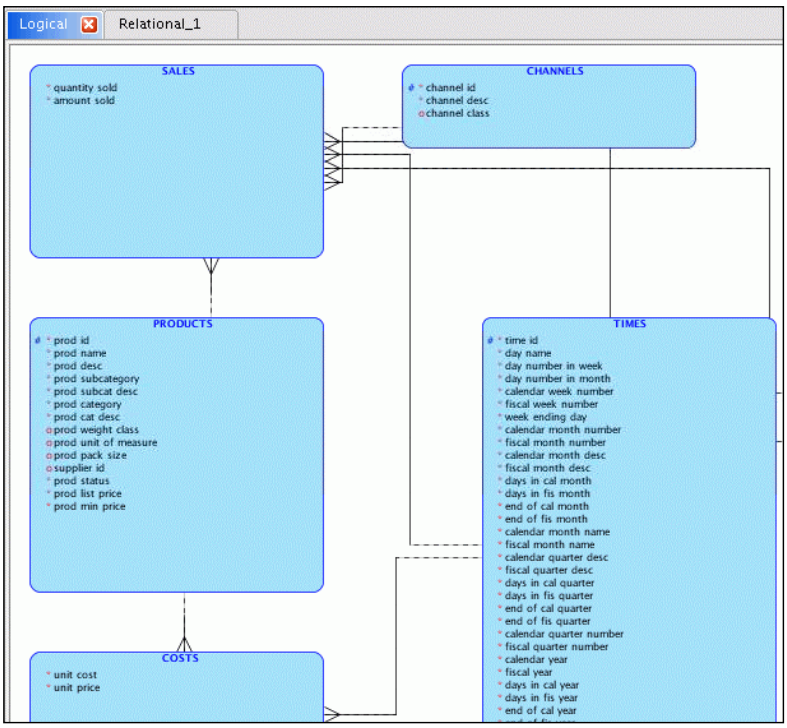


ORACLE

Reverse Engineering Your Model

Before you can create a multidimensional model, you must have a logical model, and so you must reverse engineer the relational model that you just imported. To reverse engineer, click the Engineering icon. Notice that the objects will be created because there was not an existing logical model. Click Engineer.

Reverse Engineering Your Model

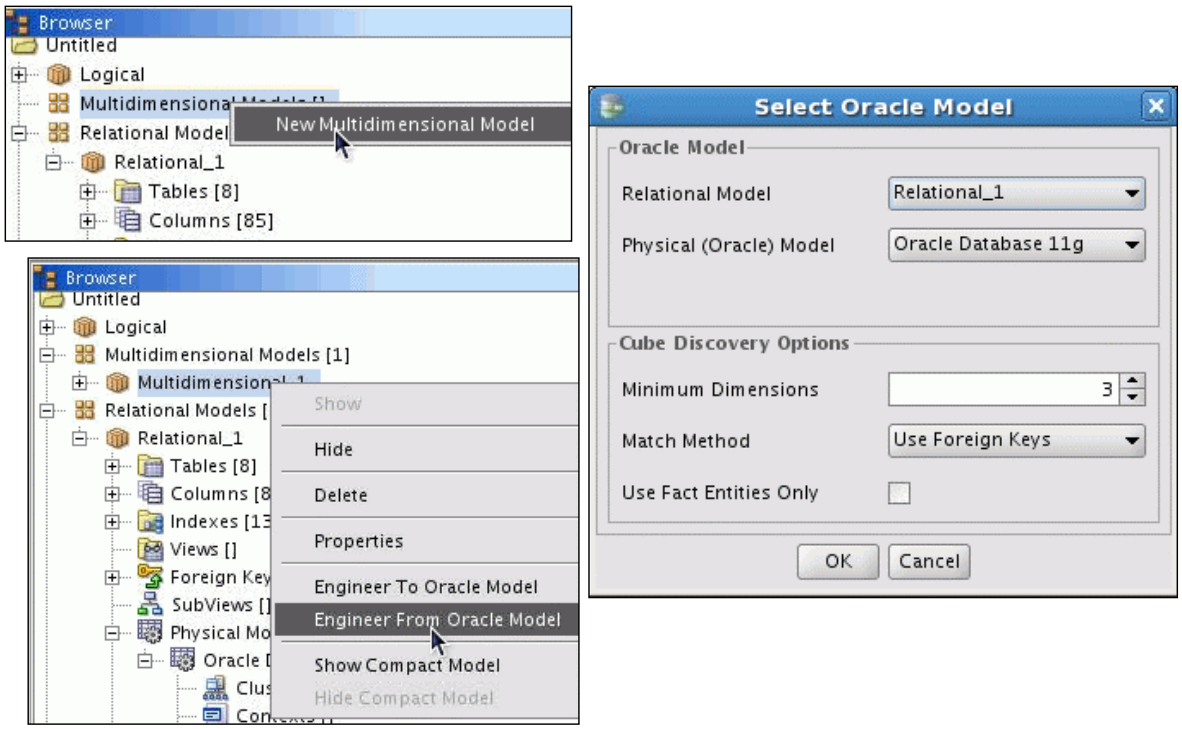


ORACLE

Reverse Engineering Your Model (continued)

The logical model is displayed. If you take a closer look at the example, you will see that the SALES entity is the fact entity and the other entities will be dimensions in the multidimensional model.

Creating Your Multidimensional Model

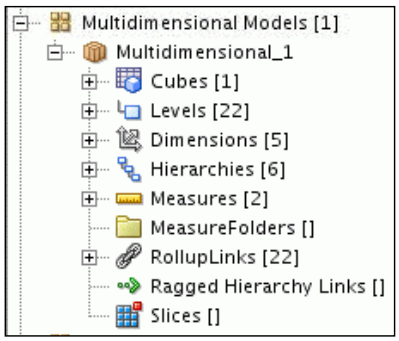


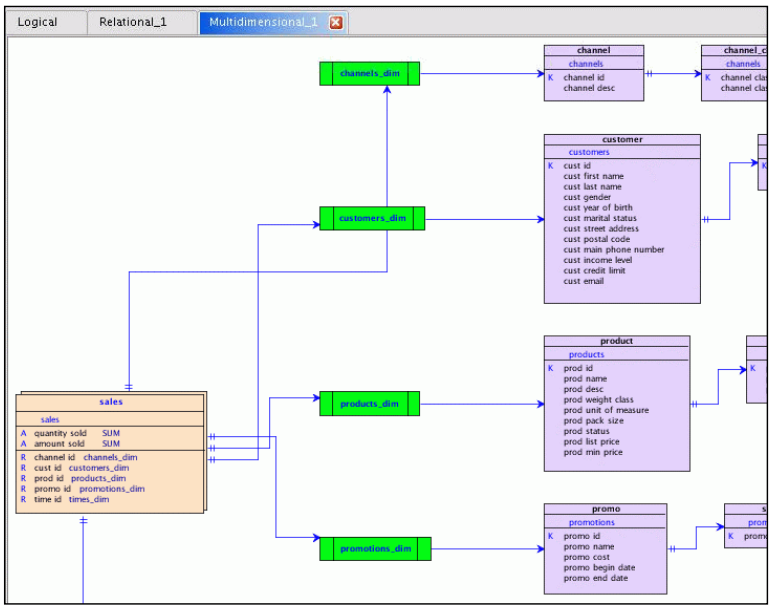
Creating Your Multidimensional Model

To create the multidimensional model, perform the following steps:

1. In the object browser, right-click Multidimensional Models and select New Multidimensional Model.
2. Right-click the Multidimensional_1 model that you just created and select Engineer From Oracle Model.
3. Select the relational and physical models that you want to engineer from. You can also change the way in which cubes will be discovered, minimum dimensions, match method, and whether to use fact entities only.
4. Click OK.

Reviewing Your Multidimensional Model



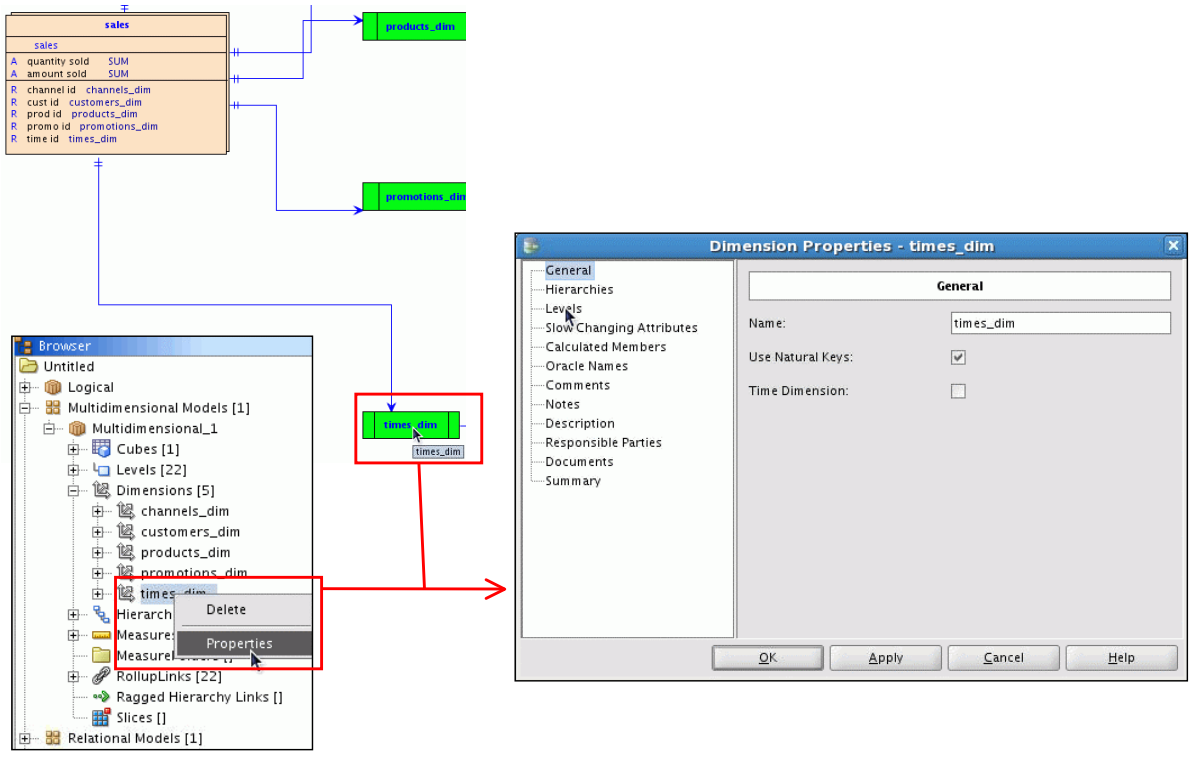


ORACLE

Reviewing Your Multidimensional Model

The multidimensional model is displayed. In the object browser, you can view the multidimensional objects that were created. In the diagram, you see the cube in peach, the dimensions in green, and the levels in purple.

Reviewing Multidimensional Object Properties



ORACLE

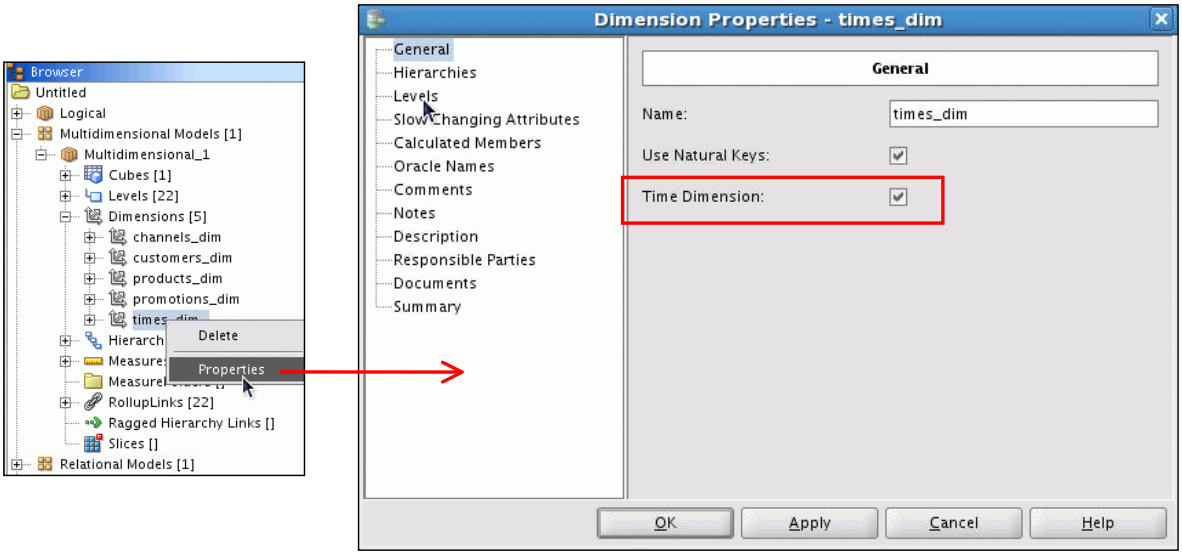
Reviewing Multidimensional Object Properties

You can view the properties of a multidimensional object by double-clicking the object in the diagram (if it is a cube, dimension, or level); otherwise, in the object browser, double-click the object, or right-click the object and select Properties.

Example

- In the example in the slide, the General properties of the times_dim Dimension object are displayed.
- You can view other properties of the multidimensional object by selecting the desired option in the left pane. An example if this is shown on page 21-27, where the Levels property for the times_dim object is selected.

Modifying Properties for the Time Dimension



ORACLE

Modifying Properties for the Time Dimension

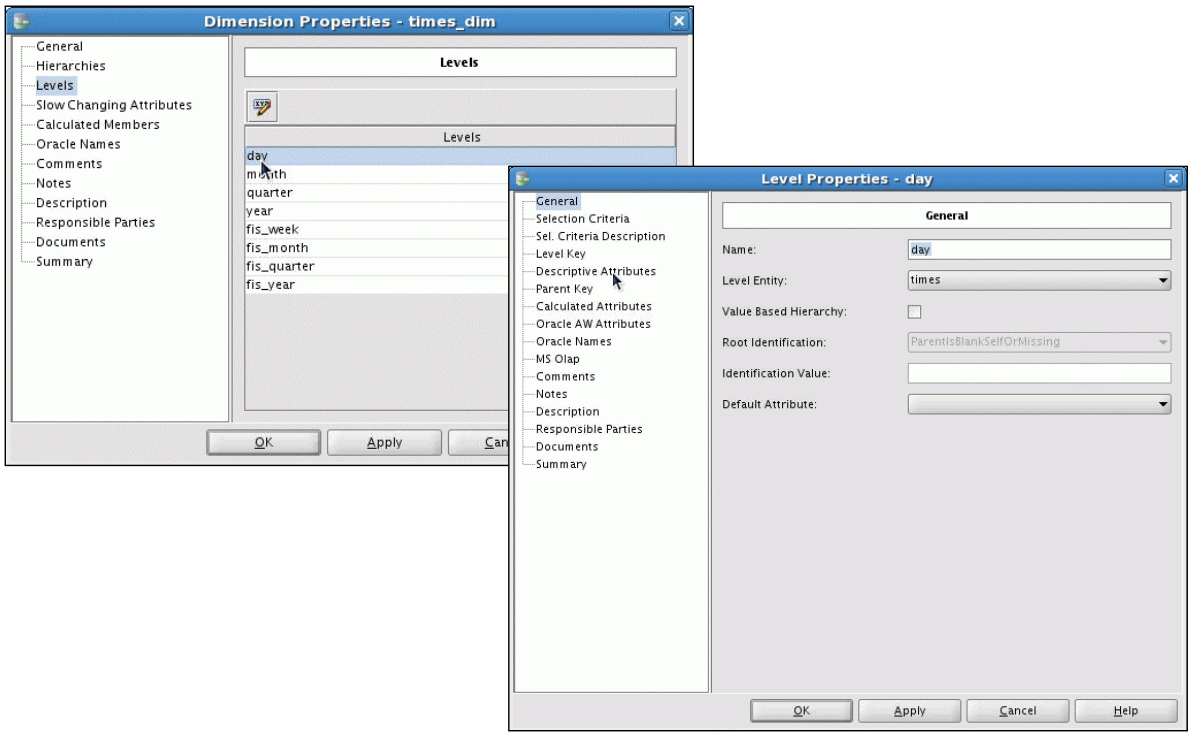
When you first create a multidimensional model, all dimension objects are considered common user dimensions.

If you want to export the multidimensional model to an Oracle AW (Oracle OLAP Analytic Workspace), you must first specify which dimension is the “Time” dimension in your model. To do this, open the Dimension Properties dialog box for the time dimension. Then, enable the Time Dimension check box. Otherwise, the Oracle OLAP Time dimension in the AW will not be defined correctly.

Example

In the example, the Time Dimension option is selected for the `times_dim` dimension object. This multidimensional model is now prepared for export to an Oracle AW.

Reviewing Properties of Multidimensional Object Components



ORACLE

Reviewing Properties of Multidimensional Object Components

You can drill down to see the properties associated with a component of a multidimensional object.

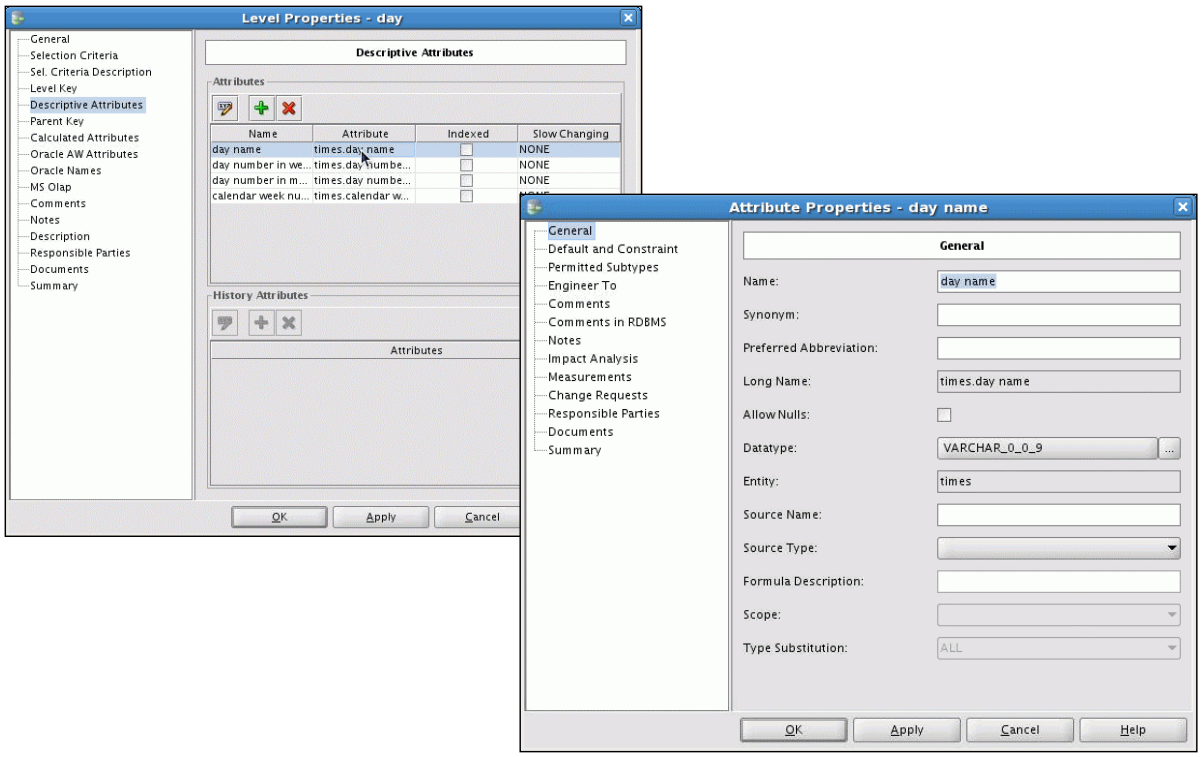
Example

In this example, the Levels property option is selected in the Dimension Properties window. To view properties for a specific level (sub-properties), double-click a value in the right pane.

Here, the day level is double-clicked to show the sub-properties for that level value.

You may then view any of the properties associated with the selected level by selecting an option in the left pane. (An example of this is shown on the next page.)

Reviewing Detailed Properties of Object Components



ORACLE

Reviewing Detailed Properties of Object Components

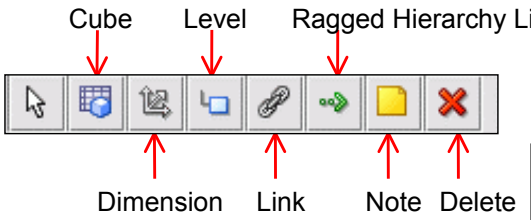
In the slide, the Descriptive Attributes option is selected for the day level in the `time_dim` dimension. From the list of attributes in the right-hand pane, you can then drill down further to see the properties associated with a specific attribute.

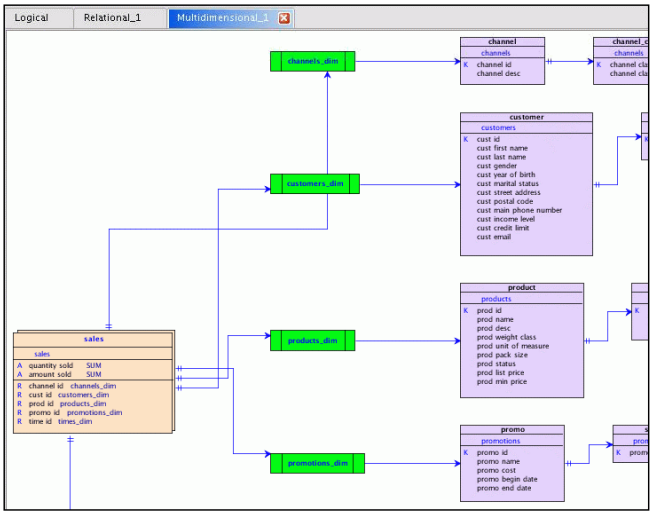
Oracle SQL Developer Data Modeler enables multiple layers of drill-down for property inspection and modification.

Example

In this case, double-click the `times.day_name` attribute name to show the Attribute Properties window, where the details are displayed.

Creating New Multidimensional Objects



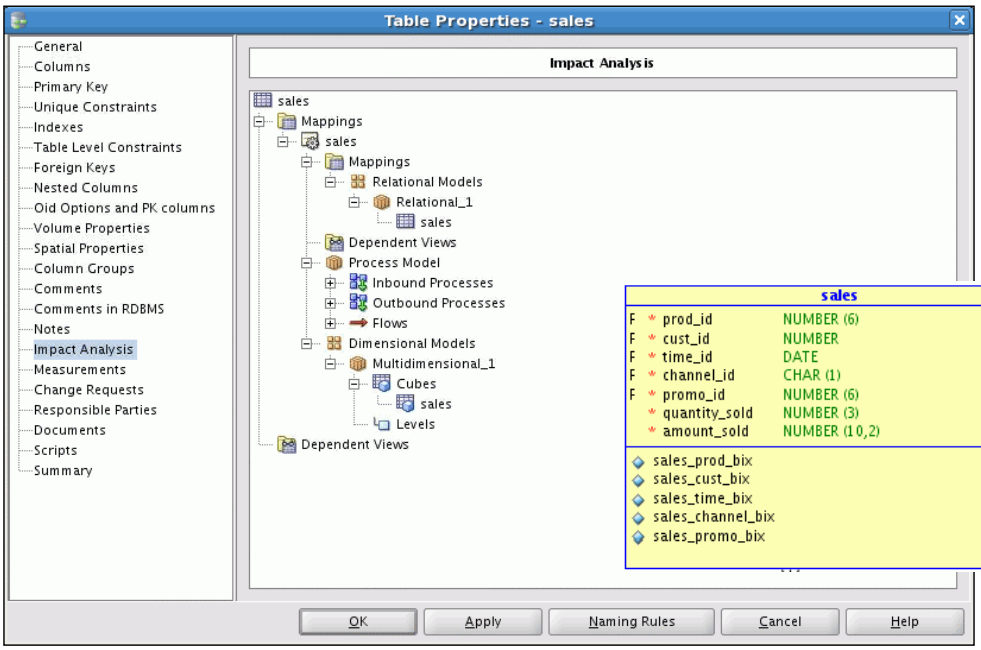


Creating New Multidimensional Objects

You can also create new multidimensional objects by using the multidimensional toolbar as shown in the slide.

Cubes, dimensions, and levels are easy to distinguish in the diagram. A link shows the relationship of one object to another (it is the blue line between objects). A ragged hierarchy link exists between levels.

Impact Analysis



Impact Analysis

To see what impact a particular table has with respect to the dimensional model and its mapping within the relational model, examine the table properties in the relational model. Select the Relational model tab, double-click the table, and select the Impact Analysis property. Expand the tree to see the details. In the example in the slide, for the sales table, you see that it maps to the sales cube in the multidimensional_1 model.

Creating an Oracle AW

Oracle SQL Developer

1. Create an Oracle AW user
   - Connect, Resource, OLAP_DBA, OLAP_USER privileges
   - Select access to relational schema

Oracle SQL Developer Data Modeler

2. Export the multidimensional model to an Oracle AW

Creating an Oracle AW

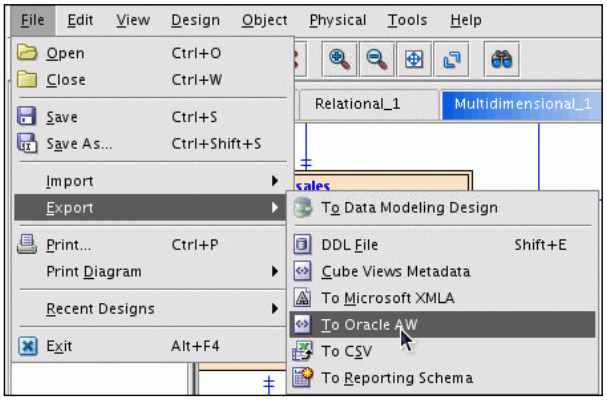
After you build a multidimensional model by using Oracle SQL Developer Data Modeler, one of the options available to you is the creation of an Oracle analytic workspace (AW). An Oracle AW is the multidimensional array-based storage model for the Oracle OLAP option. The Oracle OLAP option is the advanced multidimensional engine inside Oracle Database that interacts directly with Oracle AWs.

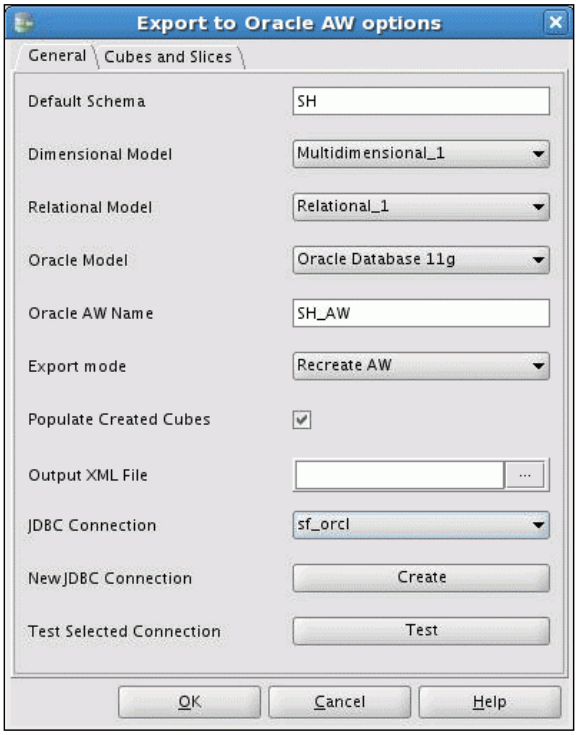
You can review and modify an AW in Oracle Analytic Workspace Manager (AWM 11*g*). Perform the following steps to produce the desired Oracle AW.

1. In Oracle SQL Developer, first create a user that will store the Oracle AW that is exported from Oracle SQL Developer Data Modeler. This user must have the access privileges indicated in the slide, as well as Select access on the tables in the schema that contains the relational model.
2. After the user is created, export the multidimensional model from Oracle SQL Developer Data Modeler to create an Oracle AW.

Note: The Oracle AW created is in OLAP 10*g* format. In order to obtain an OLAP 11*g* AW, you must perform an upgrade, which is described later in this lesson.

Exporting the Multidimensional Model





ORACLE

Exporting the Multidimensional Model

To export the multidimensional model, perform the following steps:

1. Select File > Export and select To Oracle AW.
2. Enter the default schema where your relational model is.
3. Specify the name of the Oracle AW that you want to be created.
4. Select Recreate AW from the list of export modes. Note that even though you do not have an AW, you must select this option.
5. Create a new JDBC connection to the user that you created in Oracle SQL Developer to store the AW. Alternatively, you can specify an output XML file to store the AW information that you can then import into AWM.
6. Click OK.

Upgrading Your Oracle AW by Using AWM 11*g*

Using AWM 11*g* (version 11.2.0.1), do the following:

1. Create a database connection.
2. Connect to the Oracle AW user with the OLAP 10*g* Cube type.
3. Attach the AW.
4. Create an upgrade template for Oracle 11*g*.
5. Delete the AW.
6. Disconnect from the Oracle AW user.
7. Connect to the Oracle AW user with the OLAP 11*g* Cube type.
8. Create an AW from the template.
9. Modify the AW.

ORACLE

Upgrading Your Oracle AW by Using AWM 11*g*

After you export your model to an Oracle AW, you must upgrade the resulting 10*g* format AW to an 11*g* format AW. Oracle Analytic Workspace Manager (AWM 11*g*), version 11.2.0.1 is required for this upgrade process.

To download AWM 11*g*:

1. Go to the Oracle OLAP home page on OTN at the following location: http://www.oracle.com/technology/products/bi/olap/index.html
2. Click the Analytic Workspace Manager link in the Download section. Select version 11.2.0.1.x.

Note: AWM 11*g*, version 11.2.0.1 works with both Oracle 11*g* Release 1 and Release 2 databases.

The upgrade steps are described in the slide.

For more information on how to use and modify an AW, you can perform the Oracle by Example tutorial, “Building OLAP 11*g* Cubes” at the following location:

http://www.oracle.com/technology/obe/11gr2_db_prod/bidw/olap_cube/buildcubes.htm

Summary

In this lesson, you should have learned how to:

- Describe each multidimensional object
- Import a model with dimensions
- Generate a multidimensional model
- Review and modify the relational model
- Export the multidimensional model to an Oracle AW

ORACLE

Summary

In this lesson, you examined multidimensional objects and generated a multidimensional model.

Practice 21-1 Overview: Build a Multidimensional Model

This practice covers the following topics:

- Importing a DDL file that contains dimensions
- Reverse engineering to create the logical model
- Creating the multidimensional model
- Engineering the multidimensional model
- Reviewing the multidimensional model

Additional Information

The SQL Developer Data Modeler home page contains whitepapers, documentation, Oracle by Example tutorials, and other useful information.

The URL is:

http://www.oracle.com/technology/products/database/sql_developer/index.html.

Additional Information

The links in the slide provide you with additional information related to this lesson.